XR806 PSRAM 方案

使用指南

版本号：1.0

发布时间：2020-02-24

版本历史

版本	日期	责任人	版本描述
1.0	2020-02-24	AWA 1680	创建文档。

目录

表格目录

1 前言

1.1 文档简介

本文档介绍了 XR806 平台上 PSRAM 方案的使用方法及其注意事项。

1.2 目标读者

使用 XR806 SDK 的开发人员。

1.3 适用范围

此文档适用于 XR806 SDK，支持 XR806 系列芯片产品。

1.4 文档约定

1.4.1 标志说明

本文档采用各种醒目的标志来表示在操作过程中应该特别注意的地方，这些标志的含义如下：

标识	说明
警告	该标志后的说明应给予格外关注，如果不遵守，可能会导致人员受伤或死亡。
注意	提醒操作中应注意的事项。不当的操作可能会损坏器件，影响可靠性、降低性能等。
说明	为准确理解文中指令、正确实施操作而提供的补充或强调信息。
窍门	一些容易忽视的小功能、技巧。了解这些功能或技巧能帮助解决特定问题或者节省操作时间。

1.4.2 地址与数据描述方法约定

本文档在描述地址、数据时遵循如下约定：

符号	例子	说明
0x	0x0200，0x79	地址或数据以 16 进制表示。
0b	0b010，0b00 000 111	数据采用二进制表示(寄存器描述除外)。
X	00X，XX1	数据描述中，X 代表 0 或 1。 例如，00X 代表 000 或 001；XX1 代表 001，011，101 或 111。

1.4.3 数值单位约定

本文档在描述数据容量（如 NAND 容量）时，单位词头代表的是 1024 的倍数；描述频率、数据速率等时则代表的是 1000 的倍数。具体如下：

类型	符号	对应数值
数据容量（如 NAND 容量）	1 K	1024
	1 M	1 048 576
	1 G	1 073 741 824
频率，数据速率等	1 k	1000
	1 M	1 000 000
	1 G	1 000 000 000

2 概述

2.1 背景说明

MCU 主频越来越高，处理能力越来越强大，需要的 RAM 资源相应增加。相较于传统的 SRAM 与 DRAM，PSRAM 以其小封装、大容量、低成本等特性在 IoT 领域的应用日趋广泛。因此，在 XR806 芯片中，会内部 SiP 一片 PSRAM（SQPI）进行使用。

在 PSRAM 驱动初始化完成后，即可把 PSRAM 当成 SRAM 使用（存放代码、数据等），但 PSRAM 的访问速度比 SRAM 慢。

2.2 规格特性

除 PSRAM 外，芯片还支持的可运行代码的介质有：Flash、SRAM、ROM。四者的对比如下：

表 2-1 XR806 可运行代码介质列表

存储介质	属性	执行中断	速度	支持型号
ROM	RX	Y	快，与 CPU 同频	ALL
SRAM	RWX	Y	快，与 CPU 同频	ALL
Flash	RX	N	慢，小于等于 96MHz，可 Cache 缓存	ALL
PSRAM	RWX	N	中，小于等于 120MHz，可 Cache 缓存	XR806BM2I

属性 R 表示可读，W 表示可写，X 表示可执行。

ROM 区域中存放的是固化的代码，不能修改；SRAM 和 ROM 都是系统上电即可使用；XIP 和 PSRAM 功能需要先初始化后才能使用，内置 Flash 或 PSRAM 的容量可参见《XR806_Product_Brief.pdf》文档。

注意

XR806 中，Flash 和 PSRAM 共用一个 Controller，在 Flash 进行擦写时，Controller 需要切换为 SBUS 模式，此时不能执行 PSRAM 和 Flash 中的代码，也不能读写 PSRAM 中的数据。因此，中断处理函数中执行的代码和访问的数据均不能存储在 PSRAM 中。

2.3 文件位置

以 SDK 包为根目录，本技术方案涉及到的主要文件位置如下。

```
.
└──── include
    ├──── sys
    │     └──── psram_heap.h
...
```

```
└──── project
    ├──── linker_script
    │     └──── gcc
    │           └──── appos.ld
    └──── image_cfg
          └──── image.cfg
```

关键文件说明如下。

表 2-2 PSRAM 技术方案文件说明

文件名	文件说明
appos.ld	用于编译链接时 PSRAM 的内容分布配置，用户主要需要检查/确认 PSRAM 对应的 memory 和 section 两个字段，section 中的 text/data/bss 等部分。
image.cfg	用于定义“app_psram.bin”文件在镜像文件中的位置，用户主要需要检查/确认该文件 section 段中的“app_psram.bin”大小和起始偏移地址。 “app_psram.bin”默认存储了“3.3 链接说明”章节中描述的.psram_text 段和.psram_data 段中的内容。
psram_heap.h	SDK 提供的 PSRAM 内存申请与释放接口。

注意

- 对于应用而言，除了是中断调用的函数和对性能有高要求的代码外，其他所有的代码都可以放到 PSRAM 中。
- 各自工程的编译出的.o 文件以及使用的.a 文件也需要在 ld 文件中指定存放的位置，此处未列出。
- ld 和 cfg 文件使用公共文件为示例进行介绍，若各工程指定自己的 ld 和 cfg 文件，涉及文件还包括工程指定的文件。

说明

XR806 SDK 可在以下 GitHub 仓库获取：https://github.com/XradioTech/xr806_sdk.git

3 应用说明

3.1 应用简述

在工程配置开启 PSRAM 后就会影响到 SDK 的编译、链接和系统启动流程，当通过虚拟地址访问 PSRAM 时，PSRAM/Flash Controller 会将虚拟地址翻译为在 PSRAM 中的实际地址并完成相应的数据访问操作。

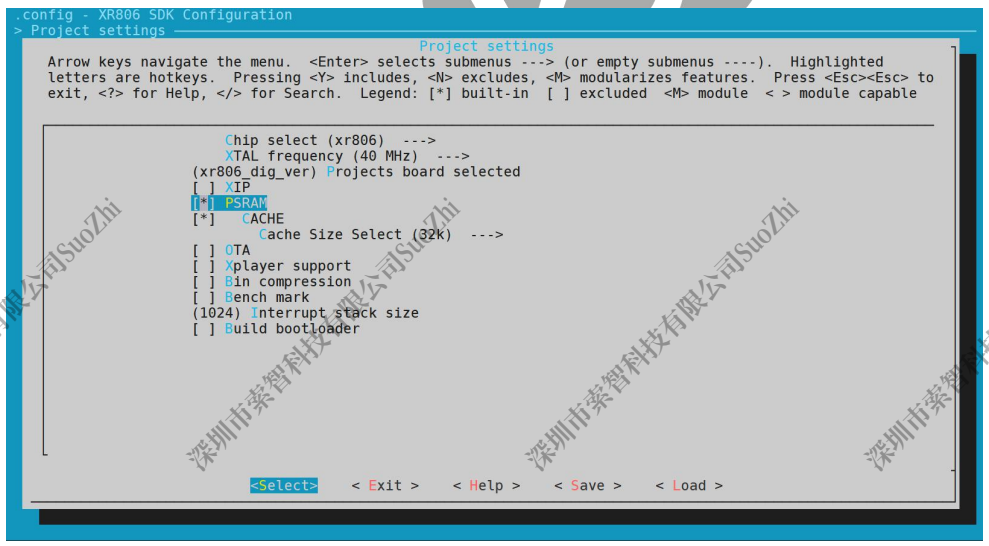
PSRAM 技术方案已经内嵌到 XR806 SDK，通过以下步骤即可启用。

1. 检查/确认 PSRAM 的 pinmux 配置与开发板原理图中的实际硬件连接是否一致（默认配置一般无需修改）。
2. 完成工程配置（menuconfig 中开启 PSRAM），详细可参考“3.2 配置说明”章节。
3. 设置相应的 ld 文件，详细可参考“3.3 链接说明”章节。
4. 检查/确认 image.cfg 文件中 PSRAM 对应 bin 文件大小，参考“3.4 镜像配置文件说明”章节。

3.2 配置说明

打开 PSRAM 功能，按如下配置：执行 make menuconfig，进入 Project settings，选中 PSRAM，同样打开 XIP 功能，只需选中 XIP。

图 3-1 打开 PSRAM 功能



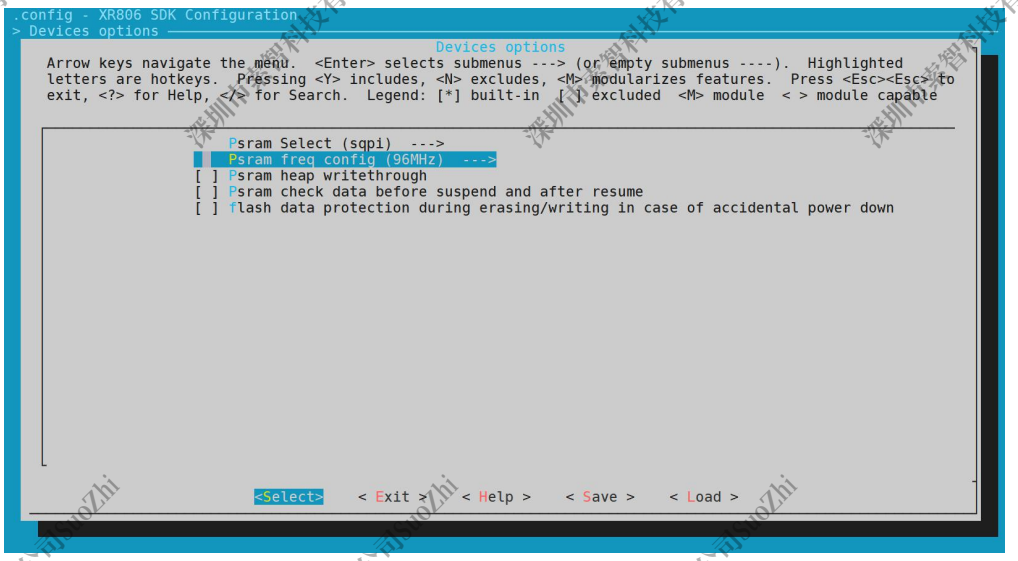
选中 PSRAM 后，即可在 PSRAM 中存放相应的代码和数据，剩余空间可以做 heap 使用。若选中 XIP 后，也可在 XIP 中存放相应的代码。

说明

开启 PSRAM 功能后，可选中 CACHE 选项以开启 Cache 功能搭配使用，Cache 的大小可选为：8KB、16KB、32KB，SDK 默认选择 32KB。

XR806 SDK 开启 PSRAM 功能后，在 menuconfig 中进入 Device options，可对 PSRAM 进行工作频率和类型等配置，如下。

图 3-2 PSRAM 配置选项



各配置说明如下（需要打开 Project settings 中的 PSRAM 功能，以上配置才会显示）：

表 3-1 XR806 PSRAM 配置列表

配置项	配置说明
PSRAM 类型	设置说明： 此项配置不同通信协议类型的 PSRAM 选择 设置位置： menuconfig 中 Device options 选项下，条目名称为“Psram Select” 设置方式： 回车进入此条目，选择 PSRAM 类型（XR806 SDK 当前只支持 SQPI 型 PSRAM）
PSRAM 频率	设置说明： 此项配置 PSRAM 的工作频率 设置位置： menuconfig 中 Device options 选项下，条目名称为“Psram freq config” 设置方式： 回车进入此条目，选择 PSRAM 工作频率（默认选 96MHz）
PSRAM 堆空间 writethrough 模式	设置说明： 此项配置 PSRAM 堆空间的 write through 模式，PSRAM 堆空间大小为存放代码后剩余的空间大小。开启此项后，对 PSRAM 堆空间的数据更新不会经过 Cache 设置位置： menuconfig 中 Device options 选项下，条目名称为“Psram heap writethrough” 设置方式：

配置项	配置说明
	按“Y”键选中此选项，使用 Cache 的 write through 模式（默认关闭）。
PSRAM 休眠唤醒后数据检查	设置说明： 此项配置开启 PSRAM 在进行休眠唤醒后的数据一致性检查。此检查功能开启后，耗时较多，建议仅在 debug 时使用。 设置位置： menuconfig 中 Device options 选项下，条目名称为“Psram check data before suspend and after resume” 设置方式： 按“Y”键选中此选项，使用休眠唤醒后数据检查（默认关闭）。

说明

1．当 PSRAM 搭配 Cache 使用时，默认使用 write back 模式，即更新数据时，只写入缓存 Cache，只有在数据被替换出缓存时，被修改的数据才会被写入 PSRAM，以此可以提高读写效率。

2．若用户开启 Psram heap writethrough 选项，则会选择使用 write through 模式，对应的 PSRAM 堆空间区域会被配置为 non-cacheable，即在对 PSRAM 进行数据更新时，直接将数据写入 PSRAM，不会经过 Cache。

3.3 链接说明

控制链接过程有两种方式，一种是修改符号链接属性，另一种是修改链接脚本。

1）符号链接属性可以在符号定义的时候进行确定，SDK 中自定义 PSRAM 的符号属性如下：

表 3-2 PSRAM 编译属性

文件名	文件说明
__psram_text	表示放在 PSRAM 中执行的代码段
__psram_rodata	表示放在 PSRAM 中的只读数据段
__psram_data	表示放在 PSRAM 中的可读写数据段
__psram_bss	表示放在 PSRAM 中的 BSS 段，该段的数据在初始化后会自动清零

其中__psram_text、__psram_rodata 最后链接到.psram_text 段中，__psram_data 会链接到.psram_data 段中，__psram_bss 会链接到.psram_bss 段中。

说明

- 对于 XIP，在 SDK 中也会有对应的符号属性“__xip_text”、“__xip_rodata”等，详细可参考 SDK/include/compiler.h

- 符号链接属性的功能可以在 menuconfig 的 Feature options 中进行关闭，SDK 默认为打开状态

2）通过链接脚本将函数以及数据分布在 PSRAM 和 XIP 中，可修改工程中使用的 ld 文件，参考 SDK/project/linker_script/gcc/appos.ld：

```
SECTIONS
{
...
#if (defined(CONFIG_PSRAM))
    .psram_text :
    {
        . = ALIGN(16);
        __psram_start__ = .;
        __psram_text_start__ = .;

        /* MUST not put IRQ handler/callback in .psram section */
        *(.psram_text* .psram_rodata*)
        *project/common/cmd/cmd_psram.o (.text* .rodata*)
        *libmp3.a: (.text .text.* .rodata .rodata.*)
        *libaac.a: (.text .text.* .rodata .rodata.*)
        ...
        . = ALIGN(16);
        __psram_text_end__ = .;
    } > PSRAM

    .psram_data :
    {
        . = ALIGN(16);
        __psram_data_start__ = .;

        /* MUST not put IRQ handler/callback used data in .psram section */
        *libmp3.a: ( .data .data.* vtable )
        *libaac.a: ( .data .data.* vtable )
        ...
        *command.o ( .data .data.* vtable )
        *(.psram_data*)

        . = ALIGN(16);
        __psram_data_end__ = .;
    } > PSRAM
```

```
    .psram_bss :
    {
        . = ALIGN(16);
        __psram_bss_start__ = .;

        /* MUST not put IRQ handler/callback used data in .psram section */
        *libmp3.a: ( .bss .bss.* COMMON )
        *libaac.a: ( .bss .bss.* COMMON )
        ...
        *command.o ( .bss .bss.* COMMON )
        *(.psram_bss*)
        . = ALIGN(16);
        __psram_bss_end__ = .;
        __psram_end__ = .;
    } > PSRAM
#endif /* CONFIG_PSRAM */
...
}
```

以上示例将 cmd_psram.o 的代码段和只读数据段放入 PSRAM 中，将 libmp3.a、libaac.a 的代码和只读数据段、data 段、bss 段全部放入 PSRAM 中；XIP 段的设置方式也类似，不同的是 XIP 不可以放入 data 和 bss 段。用户可参考此写法在 PSRAM 和 XIP 中加入自己的代码和数据。

说明

若指定某个.a 文件下的.o 文件可以加上其.a 的名字，例如：*libfs.a:lfs.o (.text *.text.*)代表 libfs.a 文件下的 lfs.o 文件的 text 段。

注意

不可以将中断中访问的资源放在 PSRAM 或 XIP 中，包括：中断处理代码、中断处理函数中调用的函数、中断回调函数、以及访问到的字符串常量、data、bss 等数据。否则在 Flash 擦写期间，由于 PSRAM 和 XIP 不能被 CPU 访问，此时若发生中断，将造成系统挂死。另外在 PSRAM 和 XIP 初始化完成前，也不能访问，即在此之前运行的代码和访问的数据等要在 ROM 或 SRAM 中。

例如，在 PSRAM 初始化完成之前的 GPIO 初始化、PM 初始化、PSRAM 驱动初始化等代码均不能放在 PSRAM 中。

将中断回调函数放入 SRAM 的方式是在函数前加“__sram_test”，示例如下：

```
__sram_text
void *dma_malloc( size_t size , uint32_t flag )
{
    return malloc(size);
}
```

3.4 镜像配置文件说明

使能 PSRAM 和 XIP 时，会产生相应的 bin 文件，若采用自己的打包文件，打包时，不要将 bin 文件漏打包，可参考 SDK/project/image_cfg/image.cfg：

```
    "image"     : {"max_size": "1020K"},
    "section" :
    [
        ...
#if (defined(CONFIG_PSRAM))
        {"id": "0xa5f65a09", "bin": "app_psram.bin", "cert": "null", "max_len": "110K",    "sram_offs":
        PRJ_PSRAM_START_OFFS, "ep": "0x00008000", "attr": "0x1"},
#endif
        ...
        {}
    ],
```

用户打包报错时：

1.检查/确认"max_len"字段是否设置太小，实际 app_psram.bin 已经超过了此大小

2.检查/确认总固件大小是否小于"max_size"字段值，若超过 max_size，则用户需要调整各 bin 文件的分布，详细可参考 SDK/project/image_cfg/readme.md

说明

PRJ_PSRAM_START_OFFS 地址可在 SDK/config.mk 文件中查看，默认为 0x01400000。

3.5 地址说明

XR806 PSRAM 地址映射空间为 0x01400000~0x023fffff（CPU 访问的地址），可参考《XR806_Datasheet.pdf》文档。

目前 XR806 仅支持 2M PSRAM，PSRAM 范围地址映射范围为 0x01400000~0x01600000，这 2M 区域均通过 CPU 的 CBUS 访问，其中起始区域存放.psram_text、.psram_data、.psram_bss 等内容，剩余区域用作 PSRAM 堆空间。

3.5.1 静态地址确认

当开启 PSRAM 后添加了新函数，可以通过 map 文件或者 objdump 文件来确认是否达到预期效果：

- 查看 map 文件：在 out 目录下会在编译的时候自动生成 xxx.map 文件，可以查看目标函数/变量保存的区域。
- 查看 objdump 文件：使用 make objdump，在 out 目录下，产生 xxx.objdump 文件，找到对应的函数/变量的地址，判断是否在指定的链接脚本的 PSRAM 或 XIP 区域中。

图 3-3 确认打开 PSRAM 功能

```
Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .xip          000d5b20  00400000  00400000  00010000  2**4
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  1 .psram_text   0001d7a0  01400000  01400000  000f0000  2**4
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  2 .psram_data   00000760  0141d7a0  0141d7a0  0010d7a0  2**4
                  CONTENTS, ALLOC, LOAD, DATA
  3 .psram_bss    000109b0  0141df00  0141df00  0010df00  2**3
                  ALLOC
  4 .text         0000a170  00201000  00201000  00001000  2**3
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  5 .ARM.exidx    00000008  0020b170  0020b170  0000b170  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  6 .data         00000e34  0020b178  0020b178  0000b178  2**3
                  CONTENTS, ALLOC, LOAD, DATA
  7 .bss          00000bc8  0020bfac  0020bfac  0000bfac  2**2
                  ALLOC
  8 .ram_table    00000400  0020cb74  0020bfac  0000cb74  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  9 .ARM.attributes 00000034  00000000  00000000  0010df00  2**0
                  CONTENTS, READONLY
 10 .comment      000000e6  00000000  00000000  0010df34  2**0
```

PSRAM 段地址在 0x01400000 开始，当前.psram_text 段大小为 0x1d7a0，.psram_data 段大小为 0x760，.psram_bss 段大小为 0x109b0。

图 3-4 确认函数在 PSRAM 中

```
0140293c <MP3FrameHeaderSync>:
 140293c:   6883        ldr r3, [r0, #8]
 140293e:   e92d 41f0   stmdb   sp!, {r4, r5, r6, r7, r8, lr}
 1402942:   681b        ldr r3, [r3, #0]
 1402944:   6e1e        ldr r6, [r3, #96]   ; 0x60
 1402946:   f506 5782   add.w   r7, r6, #4160   ; 0x1040
 140294a:   f506 5482   add.w   r4, r6, #4160   ; 0x1040
 140294e:   f8d6 8000   ldr.w   r8, [r6]
 1402952:   3710        adds    r7, #16
 1402954:   3404        adds    r4, #4
 1402956:   f44f 6100   mov.w   r1, #2048   ; 0x800
 140295a:   4630        mov r0, r6
 140295c:   f7ff faba   bl  1401ed4 <BufferDataCheck>
 1402960:   683b        ldr r3, [r7, #0]
 1402962:   2b01        cmp r3, #1
 1402964:   d020        beq.n   14029a8 <MP3FrameHeaderSync+0x6c>
 1402966:   6821        ldr r1, [r4, #0]
```

此函数的地址在 0x0140293c，属于 PSRAM 段。

说明

也可通过编译后的 out 目录，查看 app_psram.bin 文件大小是否为 0 来判断 PSRAM 中是否存在数据，如果非零则证明已经按照 ld 形式链接。

3.5.2 动态地址确认

调用 psram_malloc 接口申请 PSRAM 内存后，可以通过打印返回值看申请到的内存是否为 PSRAM 的内存地址空间。XR806 PSRAM 地址映射空间为 0x01400000~0x023fffff，因此直接查看 psram_malloc 接口的返回值是否在此范围内即可，详细可参见“3.6 接口说明”章节。

3.6 接口说明

使用 PSRAM 进行堆内存动态分配调用时，其操作必须在 platform_init_level0()初始化完后才能进行，且需要包含头文件 psram_heap.h，其路径为"SDK/include/sys/psram_heap.h"。

PSRAM 堆空间使用接口简介如下表所示。

表 3-3 PSRAM 堆空间使用接口简介

接口名	简要介绍
堆空间申请与释放接口	包括 4 个接口函数，用于提供 PSRAM 内存的分配与释放操作。
堆空间信息查看接口	包括 3 个接口函数，用于提供 PSRAM 堆内存的使用信息。

上述接口均于 SDK/src/sys/heap/psram_heap/psram_heap.c 文件中实现，XR806 PSRAM 堆空间使用接口的详细说明如下。

3.6.1 psram_malloc

表 3-4 psram_malloc 接口函数说明

信息项	说明
原型	void *psram_malloc(size_t size);
功能	分配所需的 PSRAM 内存空间，并返回一个指向它的指针
参数	size_t size 含义解释：需要申请的 PSRAM 内存块大小，以字节为单位 使用说明：size_t 为标准 C 库中定义的数据类型，在 32 位系统中 size_t 表示 long unsigned int 型，下同
返回值	若请求成功，返回一个指针，指向已分配大小的 PSRAM 内存； 若请求失败，则返回 NULL

说明

XR806 中 DMA 传输所用的内存空间均默认从 SRAM 申请，即使用 dma_malloc 时不支持选择从 PSRAM 中获取 RAM 资源。

注意

目前 XR806 PSRAM 默认配置整片地址区域均为 cacheable 属性，PSRAM 中的数据不一定与 Cache 中的数据实时同步，因此使用 DMA 从 PSRAM 向 SRAM 传输数据时，DMA 驱动内部操作为：CPU 会先将 Cache 中的数据同步到 PSRAM（进行 Cache Clean 操作），再由 DMA 从 PSRAM 向 SRAM 进行数据传输。

3.6.2 psram_realloc

表 3-5 psram_realloc 接口函数说明

信息项	说明
原型	void *psram_realloc(void *ptr, size_t size);
功能	尝试重新调整之前调用 psram_malloc 或 psram_calloc 所分配的 ptr 所指向的内存块的大小。
参数	void *ptr 含义解释：指向一个要重新分配内存的内存块（即要调整的 PSRAM 内存地址） 使用说明：若 ptr 输入为空，则会分配一个新的内存块，且函数返回一个指向它的指针。 size_t size 含义解释：内存块的新的大小，以字节为单位 使用说明：如果大小为 0，且 ptr 指向一个已存在的内存块，则 ptr 所指向的内存块会被释放，并返回一个空指针。
返回值	若请求成功，返回一个指针，指向重新分配大小的内存； 若请求失败，则返回 NULL

说明

1. 若原空间后面有足够大小的空间，则扩展内存直接从原有空间后面追加，原来空间数据不发生变化。
2. 若原空间后面没有足够大小的空间，则会在 PSRAM 堆上另找一个合适大小的连续空间来使用，这样函数返回的是一个新的内存地址。
3. 使用此接口时，不能保证每次都请求成功，因此为防止内存泄漏，用户应避免将此函数的返回值直接赋值给原 ptr 变量。

3.6.3 psram_calloc

表 3-6 psram_calloc 接口函数说明

信息项	说明
原型	void *psram_calloc(size_t nmemb, size_t size);
功能	在 PSRAM 内存中分配 nmemb 个长度为 size 的连续空间，并将这段连续内存空间清零，返回指向起始地址的指针
参数	size_t nmemb 含义解释：对象个数 使用说明：N/A size_t size 含义解释：对象占据 PSRAM 内存的字节数

信息项	说明
	使用说明：N/A
返回值	若请求成功，返回一个指针，指向所分配内存空间的起始地址； 若请求失败，则返回 NULL

3.6.4 psram_free

表 3-7 psram_free 接口函数说明

信息项	说明
原型	void psram_free(void *ptr);
功能	释放指针调用 psram_malloc、psram_realloc 或 psram_calloc 所分配的 PSRAM 内存空间
参数	void *ptr 含义解释：指针指向一个要释放的 PSRAM 内存块 使用说明：该内存块之前是通过调用 psram_malloc、psram_realloc 或 psram_calloc 进行分配内存的。如果传递的参数是一个空指针，则不会执行任何动作。
返回值	无

3.6.5 psram_total_heap_size

表 3-8 psram_total_heap_size 接口函数说明

信息项	说明
原型	size_t psram_total_heap_size();
功能	获取 PSRAM 可用堆空间总大小（除去.psram_text、.psram_data、.psram_bss 等各段占用之后）
参数	无
返回值	返回 PSRAM 堆空间总字节数

3.6.6 psram_free_heap_size

表 3-9 psram_free_heap_size 接口函数说明

信息项	说明
原型	size_t psram_free_heap_size();
功能	获取调用时，PSRAM 堆中空闲内存的大小
参数	无
返回值	返回当前 PSRAM 堆空间空闲内存总字节数

3.6.7 psram_minimum_ever_free_heap_size

表 3-10 psram_minimum_ever_free_heap_size 接口函数说明

信息项	说明
原型	size_t psram_minimum_ever_free_heap_size();
功能	获取在 OS 应用程序开始运行之后，PSRAM 堆中曾经存在的最小的未被分配的存储空间字节数。
参数	无
返回值	返回 PSRAM 堆中曾经存在的最小未被分配的存储空间字节数，主要用于估算 PSRAM 中剩余的完全空闲的堆空间大小。

4 示例说明

PSRAM 方案主要用于将用户代码放入 PSRAM 中运行，减轻 SRAM 资源压力，同时 PSRAM 剩余的堆空间也可供用户动态使用。

将用户代码放入 PSRAM 中运行的方法已在“3 应用说明”章节中进行了介绍，本示例则主要介绍与演示 PSRAM 堆空间接口的使用。

4.1 示例简介

本示例演示的目的是简要介绍 PSRAM 堆空间接口的基本使用方法（PSRAM 内存申请、读、写、堆信息查看等操作），通过串口输出打印信息。

4.1.1 获取方式

PSRAM 堆空间接口使用示例基于 hello_demo 工程，hello_demo 工程位于 XR806 SDK 的 /project/demo/hello_demo 目录。

说明

XR806 SDK 可在以下 GitHub 仓库获取：https://github.com/XradioTech/xr806_sdk.git

---

4.1.2 准备工作

本示例的硬件准备有如下。

1. 评估板：运行示例工程代码。
2. 串口线：连接评估板的 UART0 插针，用于 console 控制台的输入输出。
3. PC 机：用于镜像烧录和 console 控制的输入输出。

本示例的软件准备，包括烧写工具、代码编译和烧写操作，请参见《XR806_SDK_快速入门指南》。

基于 hello_demo 工程，需要准备的代码如下：

1. 在 main.c 文件的 main 函数中注释掉打印“Hello world”的语句，并添加例程代码，如下：

```
#include "common/framework/platform_init.h"
#include "sys/psram_heap.h"
#include <stdio.h>
#include <stddef.h>

#define PSRAM_BUF_SIZE 4096

void print_psram_heap_info()
{
    size_t total_heap, free_heap, minimum_ever_free_heap;
```

```
    total_heap = psram_total_heap_size();
    free_heap = psram_free_heap_size();
    minimum_ever_free_heap = psram_minimum_ever_free_heap_size();

    printf("[*PSRAM*]total_heap(%u)(%uKB), free_heap(%u)(%uKB),
minimum_ever_free_heap(%u)(%uKB)\n", \
            total_heap, total_heap / 1024, free_heap, free_heap / 1024, minimum_ever_free_heap, \
            minimum_ever_free_heap / 1024);
}

void psram_heap_example()
{
    uint8_t *w_buf = NULL;
    uint8_t *r_buf = NULL;

    print_psram_heap_info();

    w_buf = psram_malloc(PSRAM_BUF_SIZE);
    if (!w_buf) {
        printf("[*err*]no mem\n");
        return;
    }
    printf("[*PSRAM*]w_buf malloc success\n");

    r_buf = psram_calloc(PSRAM_BUF_SIZE, sizeof(uint8_t));
    if (!r_buf) {
        printf("[*err*]no mem\n");
        psram_free(w_buf);
        return;
    }
    printf("[*PSRAM*]r_buf malloc success\n");

    print_psram_heap_info();

    for (int i = 0; i < PSRAM_BUF_SIZE; ++i) {
        w_buf[i] = i & 0xff;
    }

    for (int i = 0; i < PSRAM_BUF_SIZE; ++i) {
        if (w_buf[i] != (i & 0xff)) {
            printf("data err, w_buf[%d] is %d, not %d\n", i, w_buf[i], (i & 0xff));
        }
    }
```

```
    printf("[*PSRAM*]w_buf check success\n");

    for (int i = 0; i < PSRAM_BUF_SIZE; ++i) {
        if (r_buf[i] != 0) {
            printf("data err, r_buf[%d] is %d, not 0\n", i, w_buf[i]);
        }
    }
    printf("[*PSRAM*]r_buf check success\n");

    psram_free(w_buf);
    psram_free(r_buf);

    printf("[*PSRAM*]psram buf free OK\n");
    print_psram_heap_info();
}

int main(void)
{
    platform_init();

    printf("[*PSRAM*]psram heap example begin\n");
    psram_heap_example();
    printf("[*PSRAM*]psram heap example end\n");

    return 0;
}
```

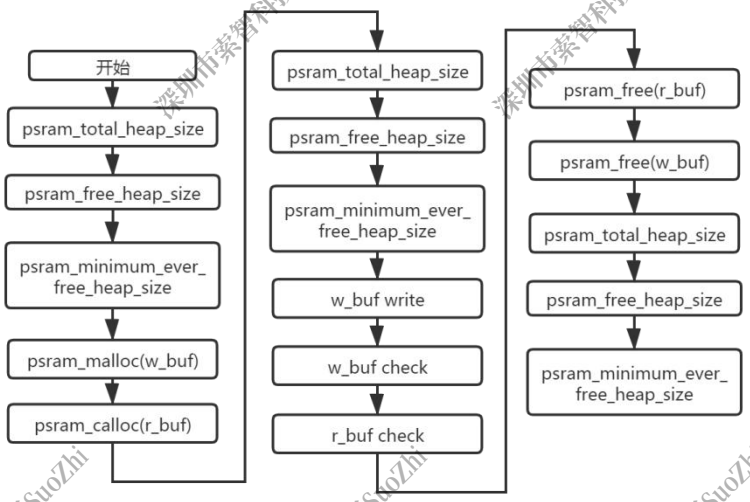
4.1.3 操作步骤

1. 在 menuconfig 中打开 Project settings 中的 PSRAM 功能和 Cache 功能；
2. 进行编译，将生成的固件编译烧写进开发板运行。

4.2 关键实现

上述示例代码的主要调用流程如下：

图 4-1 PSRAM 堆接口示例流程图



4.3 效果展示

在评估板中运行各命令后，控制台中打印输如下所示。

```
[*PSRAM*]psram heap example begin
[*PSRAM*]total_heap(2096495)(2047KB), free_heap(2096464)(2047KB), minimum_ever_free_heap(2096464)(2047KB)
[*PSRAM*]w_buf malloc success, addr(0x1400290)
[*PSRAM*]r_buf malloc success, addr(0x14012a0)
[*PSRAM*]total_heap(2096495)(2047KB), free_heap(2088240)(2039KB), minimum_ever_free_heap(2088240)(2039KB)
[*PSRAM*]w_buf check success
[*PSRAM*]r_buf check success
[*PSRAM*]psram buf free OK
[*PSRAM*]total_heap(2096495)(2047KB), free_heap(2096464)(2047KB), minimum_ever_free_heap(2088240)(2039KB)
[*PSRAM*]psram heap example end
```

调用 psram_malloc 接口前，查看总堆空间大小为 2096464（2047KB），申请两个 4KB 内存后，剩余 2088240（2039KB），内存地址分别为 0x01400290，0x014012a0，属于 PSRAM 堆内存范围，符合预期。

对 buf 中进行写值并校验，输出显示校验成功，最终释放两个 4KB 内存后，PSRAM 空闲堆空间回到 2096464（2047KB），即 psram_malloc 前的状态，且曾经存在的最小的空闲的 PSRAM 堆空间字节数为 2088240，即整个程序运行过程中的最小值，符合预期。

附录 A： 术语表

表 A-1 术语表

P		
PSRAM	Pseudo Static Random Access Memory	伪随机静态存取存储器
S		
SiP	System-in-Package	系统单封装
X		
XIP	eXecute In Place	在 Flash 中执行代码

著作权声明

商标声明

免责声明