Application Note

AWRL6432 在不同共享内存设置下的深度睡眠实现

Chris Meng *mmWave Central FAE*

摘要

业界对于毫米波雷达的低功耗应用的呼声越来越高。TI 新推出的 AWRL6432 是一款车规级低成本低功耗单芯片毫米波产品，正好能满足这些低功耗应用。该芯片支持多种低功耗模式，以及灵活的内存配置。本文以两种不同于默认实例的共享内存配置为例，深入分析了深度睡眠应用中代码需要的修改以及背后的原因，实现了在不同共享内存模式下的低功耗特性。为客户灵活使用 TI 新一代低功耗毫米波芯片提供了实用的指导和代码参考。

内容

插图清单

表格清单

1 前言

AWRL6432 是 TI 新推出的一款 60GHz 的车规级低成本低功耗单芯片毫米波产品。芯片基于低功耗工艺设计，支持多种省电模式，包括深度睡眠，实现更少功率消耗，从而满足各种对低功耗有要求的应用，例如防盗等应用。AWRL6432 还提供灵活的共享内存分配，满足不同用户的应用需求。本文以共享内存模式 3 和共享内存模式 1 为例（默认共享内存模式为 0），介绍如何在不同共享内存模式下实现实例的深度睡眠。

2 AWRL6432 的共享内存

AWRL6432 芯片上共有 1MB 的片上内存。 其中有 512KB 是专门给 APPSS（APP subsystem）也就是 Cortex-M4F（后面简称 M4F）使用。160KB 是在 HWASS（HWA subsystem）内部，给 HWASS 使用的（M4F 也可以通过芯片内部总线访问该区域内存）。剩下有 256KB 共享内存可以分配给 HWASS 或者是 APPSS 使用，还有 96KB 的共享内存可以分配给 HWASS 或者 FECSS（frontend controller subsystem）也就是射频子系统使用。具体如 图 2-1 所示。不同共享内存模式下 APSSS 内存地址和 HWA 的内存地址是不同的，请参考表 2-1。

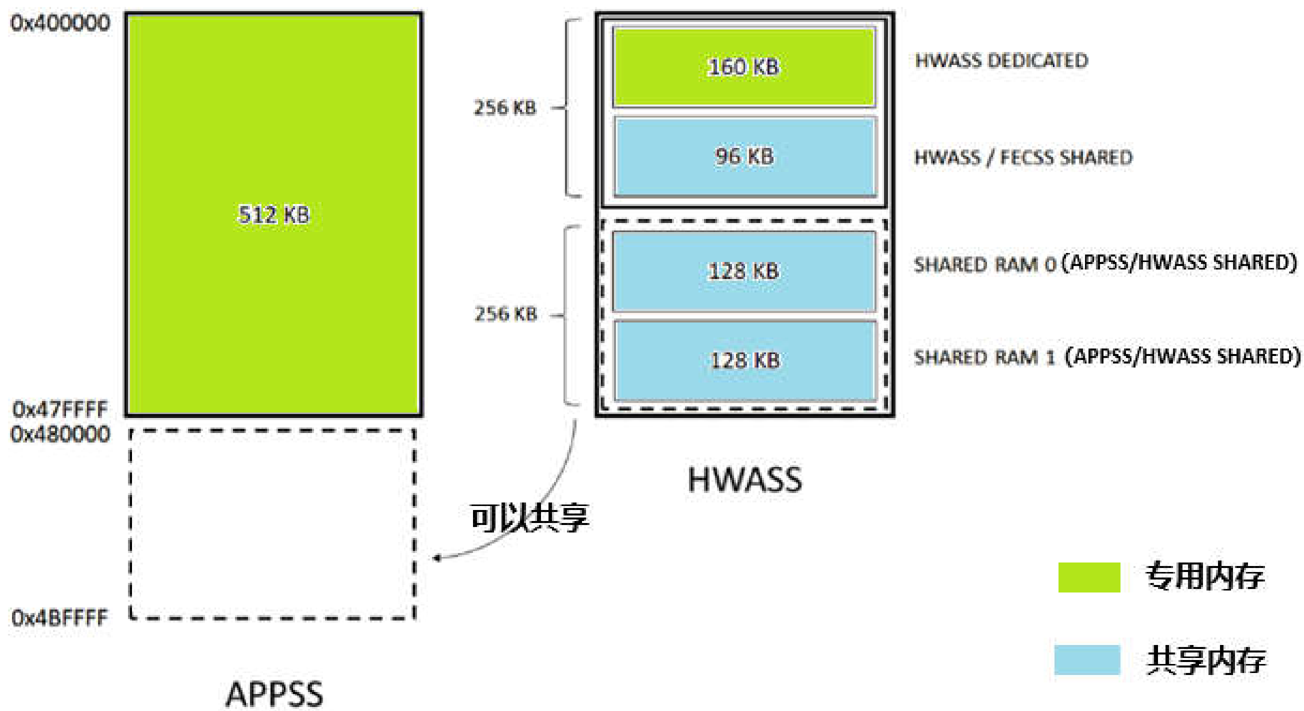

图 **2-1. AWRL6432** 内存共享示意图

表 **2-1. AWRL6432** 在不同共享内存模式下的内存地址

共享内存模式	**APPSS** 内存地址 （包含 **APSPSS/HWASS** 共享内存）	**HWASS** 内存地址 （默认 **HWASS/FECSS** 共享内存分配给 **HWASS** 使用）
SH_MEM_CONFIG=0	0x00400000 - 0x0047FFFF (512 KB)	0x60000000 - 0x6007FFFF (512 KB)
SH_MEM_CONFIG=1	0x00400000 - 0x0049FFFF (640 KB)	0x60000000 - 0x6005FFFF (384 KB)
SH_MEM_CONFIG=2	0x00400000 - 0x0047FFFF 0x004A0000 - 0x004BFFFF (640KB)	0x60000000 - 0x6005FFFF (384 KB)
SH_MEM_CONFIG=3	0x00400000 - 0x004BFFFF (768 KB)	0x60000000 - 0x6003FFFF (256 KB)

AWRL6432 的共享内存分配是通过控制寄存器 TOP_PRCM：HWA_PD_MEM_SHARE_REG（地址：0x5A040500）和 APP_CTRL：APPSS_SHARED_MEM_CLK_GATE（地址：0x56060398）两个寄存器的设置来实现。

表 2-2. 不同共享内存模式下控制寄存器值

共享内存模式	HWA_PD_MEM_SHARE_REG 寄存器值	APPSS_SHARED_MEM_CLK_GATE 寄存器值
SH_MEM_CONFIG=0	0x0	0x5
SH_MEM_CONFIG=1	0x7	0x6
SH_MEM_CONFIG=2	0x38	0x9
SH_MEM_CONFIG=3	0x3F	0xA

3 AWRL6432 的深度睡眠

AWRL6432 支持多种省电模式，其中深度睡眠（deep sleep）是最省电的状态。用户可以让芯片在发波和数据处理结束后的帧空闲时间处于深度睡眠，以获得最低的平均功耗。在深度睡眠下，除了始终开启电源域（包含实时时钟 RTC）还在工作外，用户可以设定需要保持内容的内存处于刷新状态，其他模块几乎都处于关闭/断电状态。虽然在深度睡眠模式下，芯片的大部分模块都处于完全断电的状态，但当芯片唤醒的时候，是不需要重启的，因为应用程序代码和射频设置还是保持在芯片的内存上。在 motion and presence detections 实例中当配置 lowpowercfg 1 时，在每帧处理结束后就会进入深度睡眠模式，通过帧定时器，在下一帧开始时唤醒芯片，芯片可以继续发波和进行信号处理。

4 AWRL6432 在不同共享内存设置下的深度睡眠实现

下面以 MMWAVE-L-SDK 中的 motion and presence detection 例程为例，介绍在不同共享内存模式下深度睡眠的实现。注，不同的例程修改可能有不同。

4.1 共享内存模式 SH_MEM_CONFIG=3

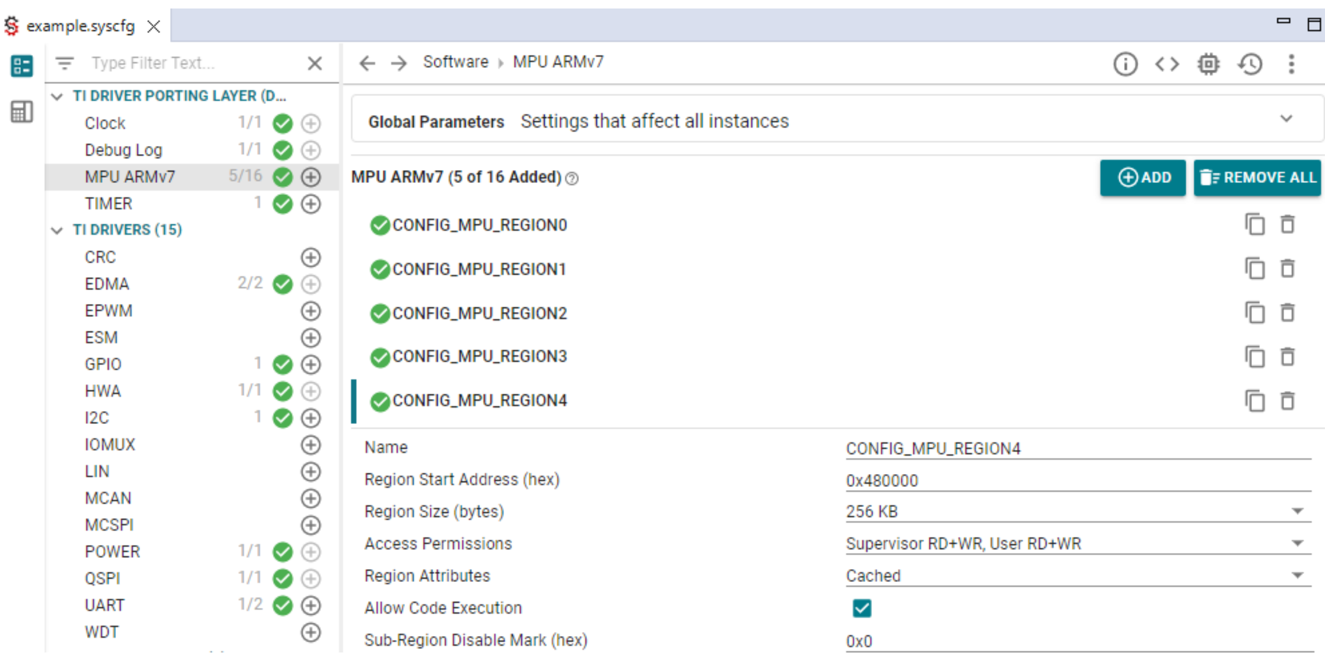

图 4-1. SH_MEM_CONFIG=3 下新增 MPU 区域配置

当共享内存模式 SH_MEM_CONFIG 由默认 0 修改为 3 时，共享内存的分配发生了变化，SHARED RAM 0/1 都配置给 M4F 使用，使得 M4F 的内存大小从默认 512KB 增加到 768KB，增加了 256KB。

例程代码需要的第一个修改是 M4F 的 MPU 配置。用户需要在例程的 syscfg 文件里添加一个 MPU 的区域，起始地址是 0x480000，也就是新增的内存起始地址，大小是 256KB，具体请参考 图 4-1。

第二个修改是工程的 cmd 文件 linkder.cmd。这个文件里需要给 M4F 增加一块 256KB 的内存，还需要减少 HWASS_SHM_MEM 的大小。

```
MEMORY
{
…
        M4F_RAM33 : ORIGIN = 0x00480000, LENGTH = 0x00040000
        HWASS_SHM_MEM : ORIGIN = 0x60000000, LENGTH = 0x00040000
}
SECTIONS
{
…
        .text: {} align(8) >> M4F_RAM33| M4F_RAM12 | M4F_RAM3
…
}
```

如果用户需要把会被 M4F 写入的区域放到新增加的 M4F_RAM33（例如下面 cmd 文件里.data 段的配置），第三个修改就是用户需要调用 ECC_disable_shared_memory 函数关闭共享内存的 ECC 功能，以规避勘误表里 DIG #14 Corrupted Data Store for Partial Write in Shared Memory 的问题。注意，所有的 AWRL6432 芯片都是支持功能安全的，默认 ECC 都是使能的。如果是不支持功能安全的其他 xWRLx432 芯片，默认 ECC 是关闭的，就不需要和 ECC 相关的修改。

```
.data: {} align(8) >> M4F_RAM33| M4F_RAM12 | M4F_RAM3
```

下面的 ECC 关闭代码注释了 FEC 共享内存的 ECC 的关闭，原因是在 SH_MEM_CONFIG=3 的配置下，M4F 是不会访问这块区域的。用户需要 driver_open 前调用这个函数，例如可以在 main 函数最开始的地方。ECC 函数源码请参考文献 2。

```
void ECC_disable_shared_memory()
{
    appEccAggEccVector = (volatile uint32_t*)0x56F7EC08;
    appEccAggEccControl = (volatile uint32_t*)0x56F7EC14;
    topEfuseRamEccCfg = (volatile uint32_t*)0x5A0201AC;
    // Disable APP SHARED MEM0 ECC
    ECC_controlWrite(ECC_VEC_APP_SHM_RAM0_ID, ECC_CON_ECC_EN_MASK, 0, 0);
    // Verify respective ECC is disabled. Throw assertion if not disabled.
    if(ECC_enableStatus(ECC_VEC_APP_SHM_RAM0_ID)) DebugP_assert(0);
    // Disable APP SHARED MEM1 ECC
    ECC_controlWrite(ECC_VEC_APP_SHM_RAM1_ID, ECC_CON_ECC_EN_MASK, 0, 0);
    // Verify respective ECC is disabled. Throw assertion if not disabled.
    if(ECC_enableStatus(ECC_VEC_APP_SHM_RAM1_ID)) DebugP_assert(0);
#if 0 //FEC share memory doesn't used by M4F with SH_MEM_CONFIG=3
    // Disable FEC SHARED MEM ECC
    ECC_controlWrite(ECC_VEC_FEC_SHM_RAM_ID, ECC_CON_ECC_EN_MASK, 0, 0);
    // Verify respective ECC is disabled. Throw assertion if not disabled.
    if(ECC_enableStatus(ECC_VEC_FEC_SHM_RAM_ID)) DebugP_assert(0);
#endif
    // Disable RAM ECC configuration for deep sleep exit
*topEfuseRamEccCfg = ((*topEfuseRamEccCfg & ~TOP_EFUSE_ALL_SHM_EN_MASK) | 0);
    return;
}
main.c
int main()
{
    ECC_disable_shared_memory();
    System_init();
    Board_init();
…
}
```

第四个修改是修改工程的 makefile 或者是 makefile_ccs_bootimage_gen 文件里的 SH_MEM_CONFIG 设置，修改其为 SH_MEM_CONFIG=3。这样编译出来的 image 文件，RBL（ROM Bootloader）在引导的时候就知道应用程序的共享内存配置，正确设置 HWA_PD_MEM_SHARE_REG 和 APPSS_SHARED_MEM_CLK_GATE 寄存器，以及初始化内存。

第五个修改和应用程序相关。RBL 是不会初始化给 HWASS 使用的内存的，包括 HWASS 专用内存和 HWASS 使用的共享内存，所以需要用户在应用程序里初始化这些内存后再使用。在 SH_MEM_CONFIG=3 下，APPSS

Shared RAM0/1 已经在 RBL 里配置给 M4F 使用了，就不需要再在 APP 里再初始化。所以需要修改 HWASS 内存初始化代码，同时例程里分配给 HWASS 使用的内存大小也需要修改为 256KB（0x40000）。

motion_detect.c

```
#define L3_MEM_SIZE (0x40000)
SOC_memoryInit(SOC_RCM_MEMINIT_HWA_SHRAM_INIT|SOC_RCM_MEMINIT_TPCCA_INIT|SOC_RCM_MEMINIT_TPCCB_INIT|
SOC_RCM_MEMINIT_FECSS_SHRAM_INIT);
```

SOC_memoryInit 函数里宏定义的含义，请参考 表 4-1。

表 4-1. 内存初始化函数使用的宏定义

宏定义	含义
SOC_RCM_MEMINIT_HWA_SHRAM_INIT	HWASS Shared RAM (160KB)
SOC_RCM_MEMINIT_TPCCA_INIT	初始化 TPCC A 内存
SOC_RCM_MEMINIT_TPCCB_INIT	初始化 TPCC B 内存
SOC_RCM_MEMINIT_FECSS_SHRAM_INIT	FECSS SHRAM (96KB)
SOC_RCM_MEMINIT_APPSS_SHRAM0_INIT	APPSS Shared RAM0 (128KB)
SOC_RCM_MEMINIT_APPSS_SHRAM1_INIT	APPSS Shared RAM1 (128KB)

如果用户没有使用深度睡眠模式，完成上面的修改代码就可以正常运行了。但是如果使能了深度睡眠模式，还需要下面三个修改才能保证芯片能正常唤醒和在唤醒后可以正常工作。

第六个修改是需要修改在低功耗模式下保持的内存区域，如 图 4-2 所示。

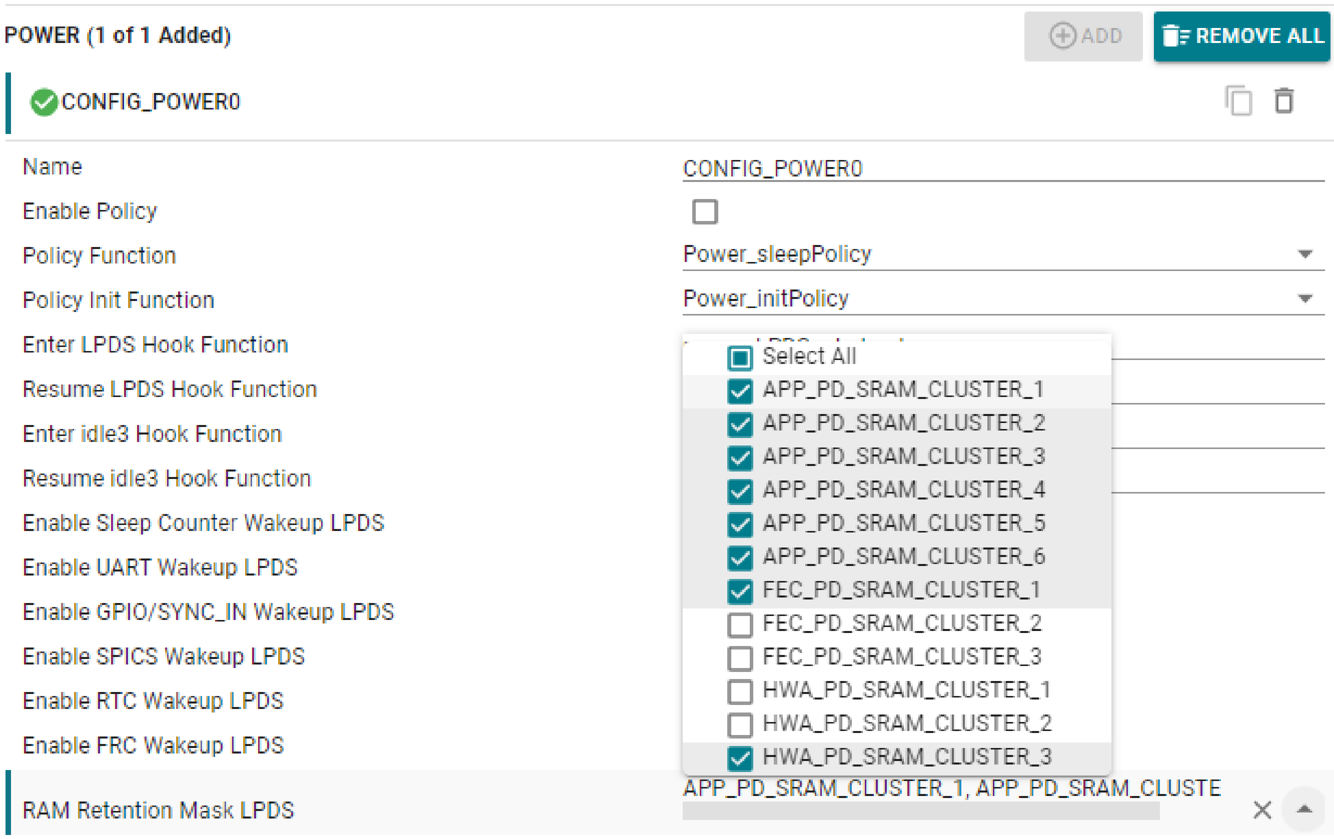

图 **4-2.** 低功耗模式下保持的内存区域配置修改（**SH_MEM_CONFIG=3**）

不同的实例和应用内存区域设置可能会不同。表 4-2 说明了图 4-2 中的 power 选项设置里选择的 memory 区域（内存簇）的具体信息。用户可以根据自己的实际应用使能或者关闭内存在深度睡眠的保持，以获得深度睡眠模式下的较低功耗。更多共享内存区域信息请参考文献 3 和 4。

表 4-2. 内存簇

	APP_PD					
簇号	1	2	3	4	5	6
内存簇	64KB BANK#1 (RAM_1A)	16KB BANK#2 (RAM_2A)	64KB BANK#1 (RAM_1B)	128KB BANK#1 (RAM_1C)	112KB BANK#2 (RAM_2B), 128KB BANK#3 (RAM_3)	256KB SHARED RAM (APP_SHMEM_1，APP_SHMEM_2)
	FECSS_PD			HWA_PD		
簇号	1	2	3	1	2	3
内存簇	FEC_RAM_1A 16KB BANK#1	FEC_RAM_1B 16KB BANK#2	96KB SHARED RAM	HWA PARAM RAM	FFT ENGINE RAM, LOCAL BUFFERS	160KB SHARED RAM

第七个修改是在深度睡眠唤醒后 APPSS_SHARED_MEM_CLK_GATE 寄存器值发生了变化，需要在使用共享内存前修改为正确的值。寄存器在深度睡眠下不一定保留进入睡眠前的寄存器值的状态。

第八个修改是在唤醒后之前的 ECC 关闭功能失效了，需要在唤醒后再次配置。

这两个修改都需要在应用代码运行前，建议添加在 power_LPDSresumehook 函数最开始的地方。

power_management.c

```
#define APPSS_SHARED_MEM_CLK_GATE        (volatile uint32_t *)(0x56060398U)
void power_LPDSresumehook(void)
{
    static uint8_t ledState = 0;
    ECC_disable_shared_memory();
    // Re-Init MPU
    MpuP_init();
    *APPSS_SHARED_MEM_CLK_GATE=0xA;// SH_MEM_CONFIG=3
```

用户在 motion and presence detection 例程里做完上述所有修改后，代码就可以在 SH_MEM_CONFIG=3 和 lowpowercfg 1 配置下正常运行了。

对于不同类型的芯片，以及不同的低功耗模式，可以参考 表 4-3 的总结，修改相应代码。

表 4-3. SH_MEM_CONFIG=3 下不同芯片类型和低功耗模式下的修改

xWRLx432 芯片类型	低功耗模式	需要的修改
不支持功能安全	lowpowercfg 1	修改一、二、四、五、六、七
	lowpowercfg 0	修改一、二、四、五
支持功能安全（AWRL6432）	lowpowercfg 1	修改一、二、三、四、五、六、七、八
	lowpowercfg 0	修改一、二、三、四、五

4.2 共享内存模式 SH_MEM_CONFIG=1

在共享内存模式 SH_MEM_CONFIG=1 下代码的修改和 SH_MEM_CONFIG=3 的修改类似。但 SH_MEM_CONFIG=1 下，仅 SHARED RAM 0 内存设置给 M4F 使用，M4F 可使用内存共为 640KB，相比于默认 SH_MEM_CONFIG=0 增加了 128KB 的内存，这和 SH_MEM_CONFIG=3 是不同的，所以修改还是有一些差异的。

修改一在 SH_MEM_CONFIG=1 下，增加的 MPU 区域配置内存大小设置为 128KB，具体设置如 图 4-3 所示。

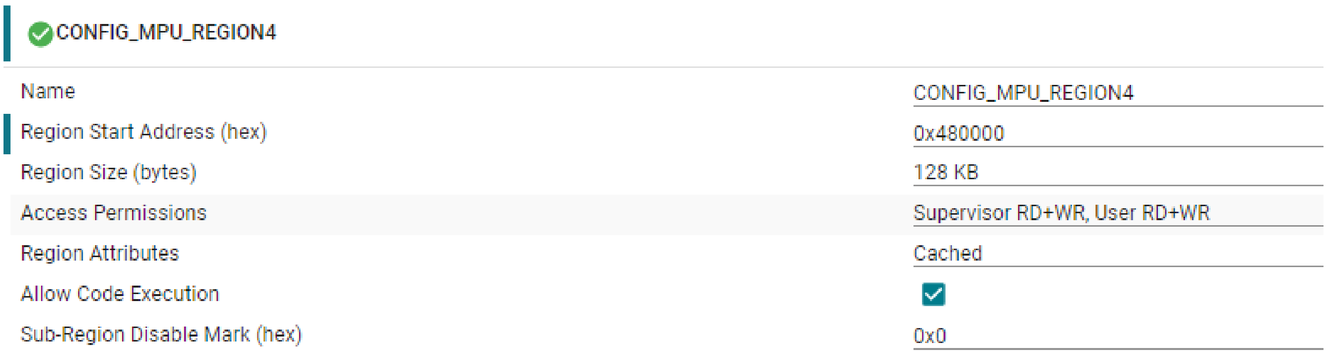

图 4-3. SH_MEM_CONFIG=1 下新增 MPU 区域配置

第二个 cmd 的修改，需要修改 M4F 增加 128KB 的内存和减少 HWASS_SHM_MEM 的大小到 384KB。

```
MEMORY
{
…
        M4F_RAM33 : ORIGIN = 0x00480000, LENGTH = 0x00020000
        HWASS_SHM_MEM : ORIGIN = 0x60000000, LENGTH = 0x00060000
}
```

第三个修改和 ECC 相关，修改节 4.1 第三个修改基本一致。不一样的地方是可以注释 SHARED RAM 1 的 ECC 关闭相关代码，因为 M4F 在 SH_MEM_CONFIG=1 设置下无法访问 SHARED RAM 1 内存。

```
    #if 0
    // Disable APP SHARED MEM1 ECC
    ECC_controlWrite(ECC_VEC_APP_SHM_RAM1_ID, ECC_CON_ECC_EN_MASK, 0, 0);
    // Verify respective ECC is disabled. Throw assertion if not disabled.
if(ECC_enableStatus(ECC_VEC_APP_SHM_RAM1_ID)) DebugP_assert(0);
#endif
```

第四个修改和共享内存模式相关，是修改工程的 makefile 或者是 makefile_ccs_bootimage_gen 文件里的 SH_MEM_CONFIG 设置为 SH_MEM_CONFIG=1。

第五个修改和内存相关，SH_MEM_CONFIG=1 配置下需要做如下修改。

motion_detect.c

```
#define L3_MEM_SIZE (0x60000)
SOC_memoryInit(SOC_RCM_MEMINIT_HWA_SHRAM_INIT|SOC_RCM_MEMINIT_TPCCA_INIT|SOC_RCM_MEMINIT_TPCCB_INIT|
SOC_RCM_MEMINIT_FECSS_SHRAM_INIT |SOC_RCM_MEMINIT_APPSS_SHRAM1_INIT);
```

在 SH_MEM_CONFIG=1 下，不需要 节 4.1 第六个修改。

第七个修改和共享内存设置相关，在 SH_MEM_CONFIG=1 下也需要在唤醒后恢复共享内存设置。第八个修改和 ECC 相关，节 4.1 一致。

power_management.c

```
#define APPSS_SHARED_MEM_CLK_GATE        (volatile uint32_t *)(0x56060398U)
void power_LPDSresumehook(void)
{
    static uint8_t ledState = 0;
    ECC_disable_shared_memory();
    // Re-Init MPU
    MpuP_init();
    *APPSS_SHARED_MEM_CLK_GATE=0x6;// SH_MEM_CONFIG=1
```

表 4-4 给出了在 SH_MEM_CONFIG=1 下，对于不同类型的芯片和功耗模式的修改总结。

表 **4-4. SH_MEM_CONFIG=1** 下不同芯片类型和低功耗模式下的修改

xWRLx432 芯片类型	低功耗模式	需要的修改
不支持功能安全	lowpowercfg 1	修改一、二、四、五、七
	lowpowercfg 0	修改一、二、四、五
支持功能安全（AWRL6432）	lowpowercfg 1	修改一、二、三、四、五、七、八
	lowpowercfg 0	修改一、二、三、四、五

5 总结

本文通过两种不同于默认例程的共享内存配置，深入分析了深度睡眠应用中代码需要的修改，背后的原因和具体实现，为客户灵活使用 TI 新一代低功耗毫米波芯片提供了实用的指导和代码参考。

本文的实现也可以同样应用到 TI 其他低功耗毫米波芯片，例如 AWRL1432，IWRL6432，IWRL1432。

6 参考文献

1. xWRL6432 MMWAVE-L-SDK: Shared Memory Usage (MMWAVE_L_SDK_05_05_02_00/docs/api_guide_xwrL64xx/SHARED_MEMORY.html)
2. Using Shared Memory With xWRLx432(radar_toolbox_3_00_00_05\software_docs\using_shared_memory_with_xwrlx432.html)
3. xWRL6432 Power Consumption
4. AWRL6432, IWRL6432, AWRL1432, IWRL1432 Technical Reference Manual (Rev. B)
5. MMWAVE-L-SDK