Application Note

Deep Sleep Implementation on AWRL6432 with Different Shared Memory Configurations

Chris Meng *mmWave Central FAE*

ABSTRACT

This document was translated from a simplified Chinese source. (ZHCAFE4)

The demand for low-power mmWave radar solutions continues to grow across the industry. TI's newly launched AWRL6432 is an automotive-grade, low-cost, low-power single-chip mmWave device that is well suited to meet the requirements of such low-power applications. The device supports multiple low-power modes as well as flexible memory configurations. Using two shared memory configurations different from the default example as case studies, this article provides an in-depth analysis of the code modifications required for deep sleep applications and the reasons behind them, enabling low-power operation under different shared memory modes. It also provides practical guidance and code references for customers to flexibly utilize TI's new generation of low-power mmWave devices.

Table of Contents

List of Figures

List of Tables

1 Introduction

AWRL6432 is a newly introduced 60GHz automotive-grade, low-cost, low-power single-chip mmWave device from TI. The device is designed using a low-power process and supports multiple power-saving modes, including deep sleep, to minimize power consumption and satisfy various applications with stringent low-power requirements, such as anti-theft applications. AWRL6432 also provides flexible shared memory allocation to meet the needs of different user applications. Using shared memory mode 3 and shared memory mode 1 as examples (the default shared memory mode is 0), this article describes how to implement deep sleep for applications under different shared memory modes.

2 Shared Memory of AWRL6432

AWRL6432 contains a total of 1MB of on-chip memory. Among this memory, 512KB is dedicated to APPSS (APP subsystem), namely the Cortex-M4F (hereinafter referred to as M4F). Another 160KB resides within HWASS (HWA subsystem) and is used by HWASS (M4F can also access this memory region through the internal bus of the device). Of the remaining memory, 256KB of shared memory can be allocated either to HWASS or APPSS, while another 96KB of shared memory can be allocated either to HWASS or FECSS (Frontend Controller Subsystem), namely the RF subsystem. Details are shown in Figure 2-1. The APSSS memory addresses and HWA memory addresses differ under different shared memory modes. Please refer to Table 2-1.

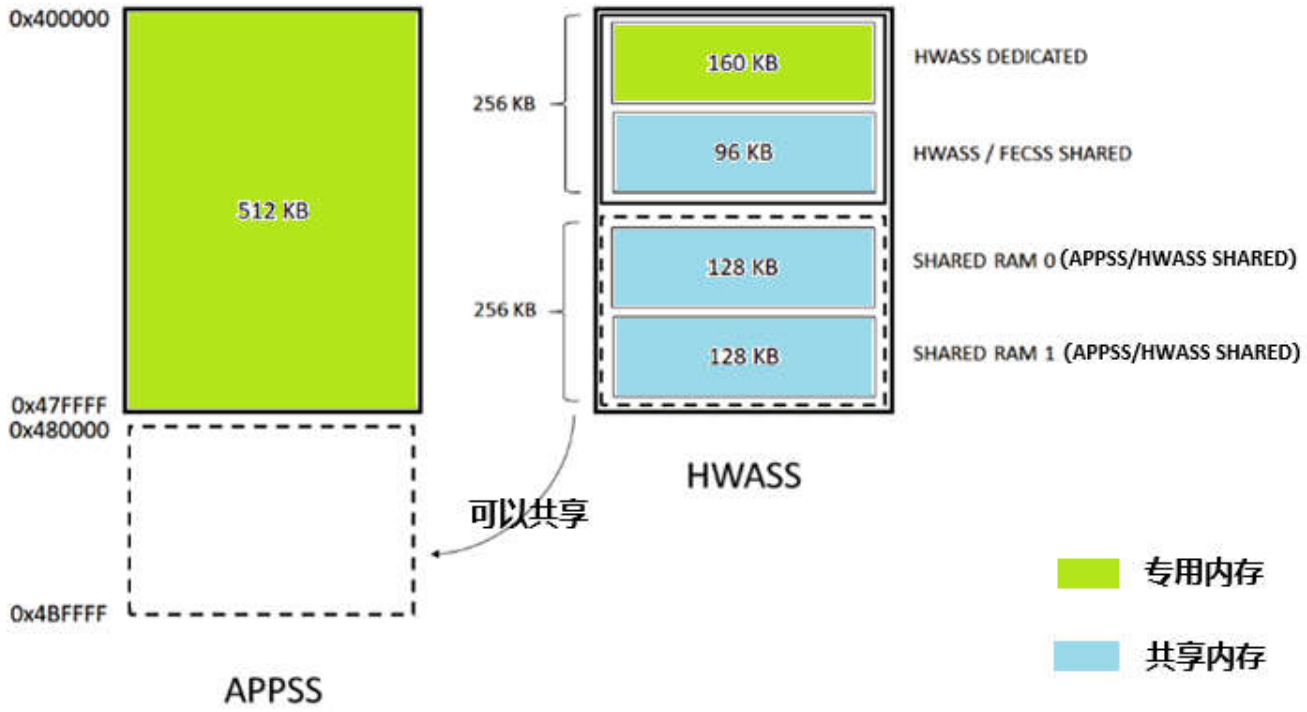

Figure 2-1. Schematic Diagram of AWRL6432 Shared Memory Architecture

Table 2-1. Memory Addresses of AWRL6432 Under Different Shared Memory Modes

Shared Memory Mode	APPSS Memory Address (Including APSPSS/HWASS Shared Memory)	HWASS Memory Address (Default HWASS/FECSS Shared Memory Assigned to HWASS)
SH_MEM_CONFIG=0	0x00400000 - 0x0047FFFF (512KB)	0x60000000 - 0x6007FFFF (512KB)
SH_MEM_CONFIG=1	0x00400000 - 0x0049FFFF (640KB)	0x60000000 - 0x6005FFFF (384KB)
SH_MEM_CONFIG=2	0x00400000 - 0x0047FFFF 0x004A0000 - 0x004BFFFF (640KB)	0x60000000 - 0x6005FFFF (384KB)
SH_MEM_CONFIG=3	0x00400000 - 0x004BFFFF (768KB)	0x60000000 - 0x6003FFFF (256KB)

Shared memory allocation on AWRL6432 is implemented through configuration of two control registers: TOP_PRCM:HWA_PD_MEM_SHARE_REG (Address: 0x5A040500) and APP_CTRL:APPSS_SHARED_MEM_CLK_GATE (Address: 0x56060398).

Table 2-2. Control Register Values Under Different Shared Memory Modes

Shared Memory Mode	HWA_PD_MEM_SHARE_REG Register Value	APPSS_SHARED_MEM_CLK_GATE Register Value
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 Deep Sleep on AWRL6432

AWRL6432 supports multiple power-saving modes, among which deep sleep is the lowest-power state. Users can place the device into deep sleep during the frame idle time after chirp transmission and data processing are completed, thereby achieving the lowest average power consumption. During deep sleep, except for the always-on power domain (which includes the real-time clock RTC), users can configure specific memory regions to retain their contents through refresh, while almost all other modules are shut down or powered off. Although most modules of the device are completely powered down during deep sleep mode, the device does not need to reboot upon wake-up because the application code and RF configurations remain stored in memory. In the motion and presence detection example, when lowpowercfg 1 is configured, the device enters deep sleep mode after each frame is processed. Through the frame timer, the device wakes up at the beginning of the next frame and continues chirp transmission and signal processing.

4 Deep Sleep Implementation on AWRL6432 with Different Shared Memory Configurations

Using the motion and presence detection example in MMWAVE-L-SDK as a reference, the following sections describe the implementation of deep sleep under different shared memory modes. Note that modifications may vary for different examples.

4.1 Shared Memory Mode SH_MEM_CONFIG=3

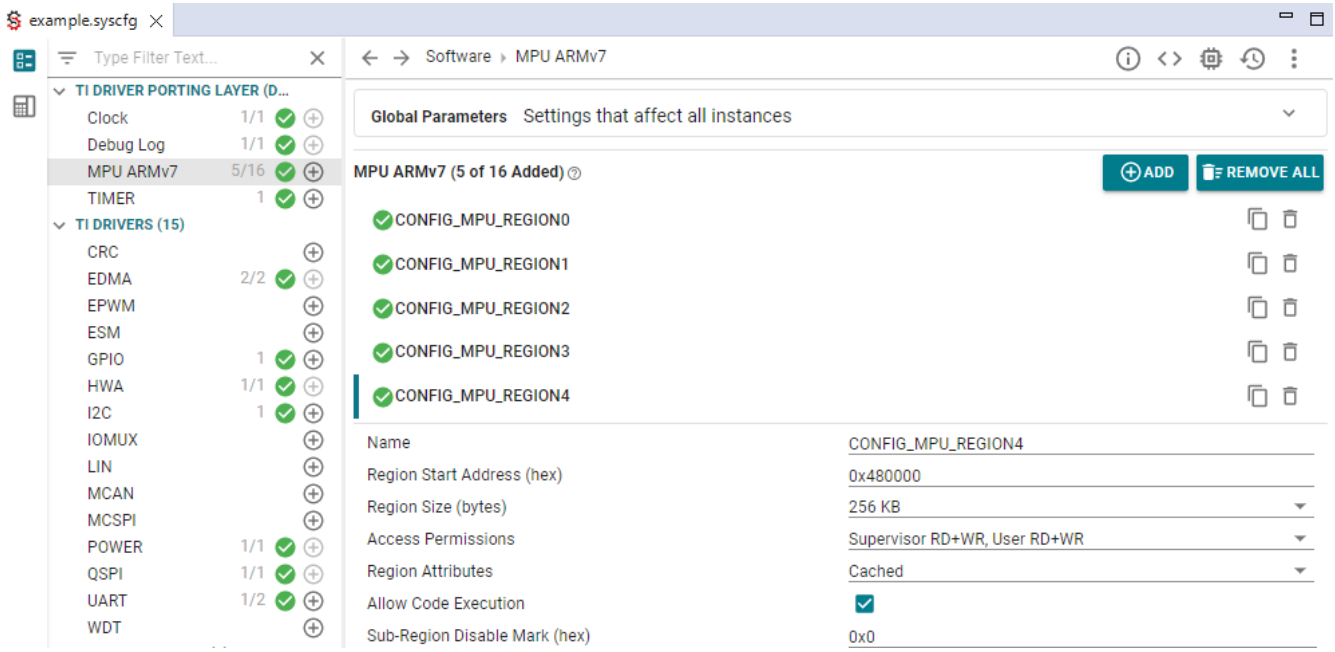

Figure 4-1. Additional MPU Region Configuration for SH_MEM_CONFIG=3

When SH_MEM_CONFIG is changed from the default value of 0 to 3, the shared memory allocation changes. Both SHARED RAM 0 and SHARED RAM 1 are assigned to the M4F, increasing available M4F memory from the default 512KB to 768KB, an increase of 256KB.

The first required modification to the example code is the MPU configuration of the M4F. Users need to add an MPU region in the example's syscfgfile. The start address should be 0x480000, which is the starting address of the newly added memory, and the size should be 256KB. Refer to Figure 4-1 for details.

The second modification involves the project's linker.cmd file. In this file, an additional 256KB memory region must be added for the M4F, and the size of HWASS_SHM_MEM must be reduced accordingly.

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

If users need to place regions that will be written by the M4F into the newly added M4F_RAM33 region (for example, the .data section shown in the linker.cmd file below), the third modification is to call the ECC_disable_shared_memory function to disable ECC protection for shared memory. This avoids the issue described in the errata item DIG #14: Corrupted Data Store for Partial Write in Shared Memory. Note that all AWRL6432 devices support functional safety, and ECC is enabled by default. For other xWRLx432 devices that do not support functional safety, ECC is disabled by default, and no ECC-related modifications are required.

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

In the code example below, ECC disabling for FEC shared memory is commented out because M4F does not access this region when SH_MEM_CONFIG=3 is used. Users should call this function before driver_open, for example at the beginning of the main function. Refer to the reference 2 for the ECC function source code.

```
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();
…
}
```

The fourth modification is to change the SH_MEM_CONFIG setting in the project makefile or makefile_ccs_bootimage_gen file to SH_MEM_CONFIG=3. This ensures that the generated image file contains the correct shared memory configuration, allowing the RBL (ROM Bootloader) to correctly configure the

HWA_PD_MEM_SHARE_REG and APPSS_SHARED_MEM_CLK_GATE registers and initialize memory during boot.

The fifth modification is application-related. The RBL does not initialize memory assigned to HWASS, including dedicated HWASS memory and HWASS shared memory. Therefore, users must initialize these memory regions within the application before using them. Under SH_MEM_CONFIG=3, APPSS Shared RAM0/1 has already been assigned to M4F by the RBL and therefore does not need to be initialized again in the APP. As a result, the HWASS memory initialization code must be modified, and the amount of memory allocated to HWASS in the example must be changed to 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);
```

Please refer to Table 4-1 for the meanings of the macros used in the SOC_memoryInit function.

Table 4-1. Macros Definition Used by the Memory Initialization Function

Macro Definition	Meaning
SOC_RCM_MEMINIT_HWA_SHRAM_INIT	HWASS Shared RAM (160KB)
SOC_RCM_MEMINIT_TPCCA_INIT	Initialize TPCC A Memory
SOC_RCM_MEMINIT_TPCCB_INIT	Initialize TPCC B Memory
SOC_RCM_MEMINIT_FECSS_SHRAM_INIT	FECSS SHRAM (96KB)
SOC_RCM_MEMINIT_APPSS_SHRAM0_INIT	APSS Shared RAM0 (128KB)
SOC_RCM_MEMINIT_APPSS_SHRAM1_INIT	APSS Shared RAM1 (128KB)

If deep sleep mode is not used, the application can operate correctly after completing the above modifications. However, when deep sleep mode is enabled, three additional modifications are required to ensure proper wake-up and operation after wake-up.

The sixth modification is to adjust the memory regions retained during low-power mode, as shown in Figure 4-2.

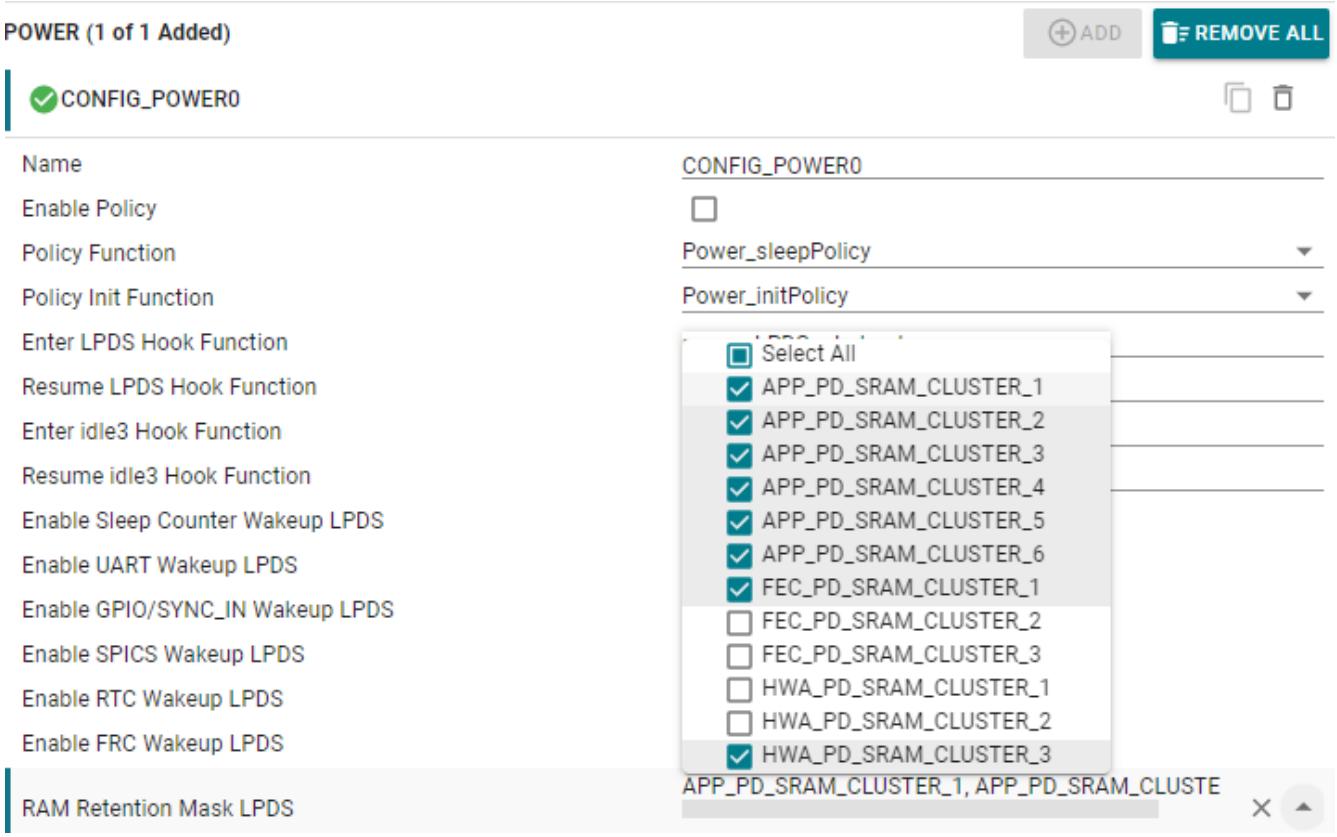

Figure 4-2. Memory Retention Configuration Changes in Low-Power Mode (SH_MEM_CONFIG=3)

Memory region settings may vary depending on the example and application.Table 4-2 That explains the specific memory cluster information corresponding to the memory regions selected in the power configuration settings of Figure 4-2. Users can enable or disable memory retention during deep sleep according to their application requirements to achieve lower power consumption in deep sleep mode. For more information about shared memory regions, refer to the reference 3 and 4.

Table 4-2. Memory Clusters

	APP_PD					
Cluster Number	1	2	3	4	5	6
Memory Clusters	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**		
Cluster Number	1	2	3	1	2	3
Memory Clusters	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

The seventh modification is that the APPSS_SHARED_MEM_CLK_GATE register value changes after wake-up from deep sleep, and it must be restored to the correct value before accessing shared memory. Register values are not necessarily retained across deep sleep transitions.

The eighth modification is that the previously disabled ECC functionality becomes ineffective after wake-up and must be configured again.

Both modifications should be performed before application code execution. It is recommended to place them at the beginning of the power_LPDSresumehook function.

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
```

After all of the above modifications are completed in the motion and presence detection example, the code can operate correctly with SH_MEM_CONFIG=3 and lowpowercfg 1 enabled.

For different device types and low-power modes, users can refer to the summary in Table 4-3 and modify the corresponding code accordingly.

Table 4-3. Modifications Required Under SH_MEM_CONFIG=3 for Different Device Types and Low-Power Modes

xWRLx432 Device Type	Low-Power Mode	Required Modifications
Functional-safety not supported	lowpowercfg 1	Modifications 1, 2, 4, 5, 6, 7
	lowpowercfg 0	Modifications 1, 2, 4, 5
Functional-safety supported (AWRL6432)	lowpowercfg 1	Modifications 1, 2, 3, 4, 5, 6, 7, 8
	lowpowercfg 0	Modifications 1, 2, 3, 4, 5

4.2 Shared Memory Mode SH_MEM_CONFIG=1

The code modifications required under shared memory mode SH_MEM_CONFIG=1 are similar to those required for SH_MEM_CONFIG=3. However, under SH_MEM_CONFIG=1, only SHARED RAM 0 is assigned to the M4F. Therefore, the total memory available to the M4F is 640KB, which is an increase of 128KB compared to the default SH_MEM_CONFIG=0. This differs from SH_MEM_CONFIG=3, resulting in some modification differences.

For Modification 1, the additional MPU region size should be configured as 128KB under SH_MEM_CONFIG=1, as shown in Figure 4-3.

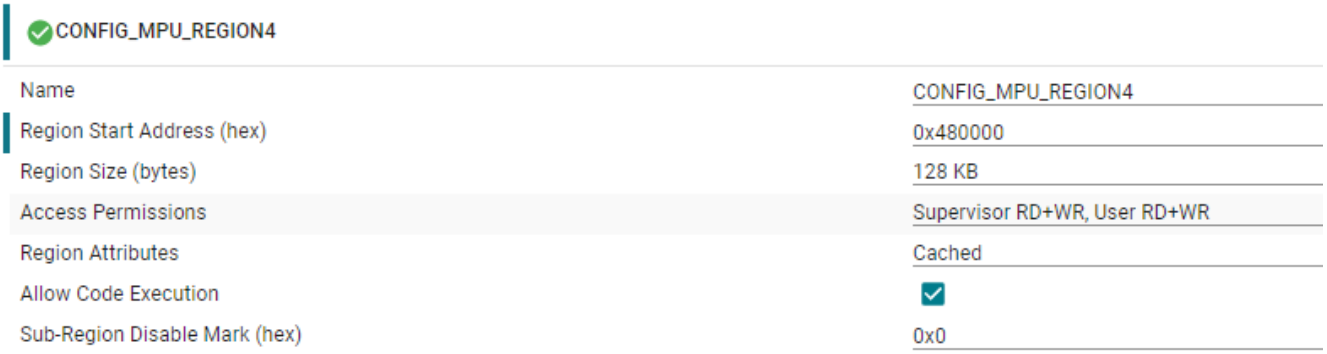

Figure 4-3. Additional MPU Region Configuration for SH_MEM_CONFIG=1

For the second modification involving the linker.cmd file, an additional 128KB memory region must be allocated to the M4F, and the size of HWASS_SHM_MEM must be reduced to 384KB.

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

The third modification is related to ECC and is generally the same as the third Section 4.1 modification described earlier. The difference is that ECC disabling for SHARED RAM 1 can be commented out because M4F cannot access SHARED RAM 1 under SH_MEM_CONFIG=1.

```
    #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
```

The fourth modification is related to shared memory mode and requires changing SH_MEM_CONFIG in the project makefile or makefile_ccs_bootimage_gen file to SH_MEM_CONFIG=1.

The fifth modification is memory-related. Under SH_MEM_CONFIG=1, the following modifications are required.

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);
```

Under SH_MEM_CONFIG=1, the sixth modification shown in Section 4.1 is not required.

The seventh modification is related to shared memory settings. Under SH_MEM_CONFIG=1, shared memory settings must also be restored after wake-up. The eighth modification is ECC-related and Section 4.1 remains unchanged.

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
```

Table 4-4 also provides a summary of the modifications required under SH_MEM_CONFIG=1 for different device types and power modes.

Table 4-4. Modifications Required Under SH_MEM_CONFIG=1 for Different Device Types and Low-Power Modes

xWRLx432 Device Type	Low-Power Mode	Required Modifications
Functional-safety not supported	lowpowercfg 1	Modifications 1, 2, 4, 5, 7
	lowpowercfg 0	Modifications 1, 2, 4, 5
Functional-safety supported (AWRL6432)	lowpowercfg 1	Modifications 1, 2, 3, 4, 5, 7, 8
	lowpowercfg 0	Modifications 1, 2, 3, 4, 5

5 Summary

Through two shared memory configurations different from the default example, this article provides an in-depth analysis of the code modifications required for deep sleep applications, the reasons behind them, and their implementation details, offering practical guidance and code references for customers using TI's new generation of low-power mmWave devices.

The implementation described in this article can also be applied to other TI low-power mmWave devices, such as AWRL1432, IWRL6432, and IWRL1432.

6 References

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

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you fully indemnify TI and its representatives against any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale, TI’s General Quality Guidelines, or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products. Unless TI explicitly designates a product as custom or customer-specified, TI products are standard, catalog, general purpose devices.

TI objects to and rejects any additional or different terms you may propose.

Last updated 10/2025