# FreeRTOS SMP Audio Processing Implementation Guide for AM62x A53

## Table of Contents



---

## 1. Starting Point - Reference Examples

Rather than building a project from scratch, start from these SDK examples:

### SMP Boilerplate and Main Structure

```
examples/hello_world/am62x-sk/a53ss0-0_freertos-smp/
```

### SMP Scheduler and Task Affinity

```
examples/kernel/freertos/smp_task_switch/am62x-sk/a53ss0-0_freertos-smp/
```

### McASP with DMA on A53

```
examples/drivers/mcasp/mcasp_loopback/am62x-sk/a53ss0-0_freertos/
```

### McASP with AIC31 Codec

```
examples/drivers/mcasp/mcasp_playback_codec_aic31/
```

***Important Note:** The McASP examples currently target `a53ss0-0_freertos` (single-core), not `freertos-smp`. When combining McASP with SMP, use the SMP `main.c` pattern and add McASP via SysConfig.*

---

# 2. SMP Specific Considerations

For comprehensive understanding of FreeRTOS SMP, refer to: https://www.freertos.org/Documentation/02-Kernel/02-Kernel-features/13-Symmetric-multiprocessing-introduction

## Overview

Most FreeRTOS SMP APIs are identical to single-core FreeRTOS. Four new APIs are added for SMP:

• `vTaskCoreAffinitySet()` - Set which cores a task can run on

• `vTaskCoreAffinityGet()` - Query task core affinity

• `vTaskPreemptionDisable()` - Disable preemption on current core

• `vTaskPreemptionEnable()` - Re-enable preemption on current core

## FreeRTOS SMP Configuration

Configuration file location:

```
source/kernel/freertos/config/am62x/a53-smp/FreeRTOSConfig.h
```

### Key Configuration Settings

|---------|-------|-------------|

### configRUN_MULTIPLE_PRIORITIES Explained

• **Value 0 (Default):** Maintains single-core paradigm where a lower priority task never runs if a higher priority task is ready. Multiple tasks run simultaneously only if they have equal priority.

• **Value 1:** Allows higher and lower priority tasks to run concurrently on different cores, enabling more parallelism but requiring careful priority design.

---

# 3. Core Initialization Sequence (Critical)

The multi-core boot pattern from `main.c` is essential for proper SMP operation:

```
#include <kernel/dpl/SystemP.h>

// Global flag for scheduler synchronization
extern volatile uint64_t ullPortSchedularRunning;

int main(void)
{
    Drivers_open();

    if (0 == Armv8_getCoreId())  // Core 0 only
    {
        // Core 0: Create init task and START the scheduler
        gMainTask = xTaskCreateStatic(
            freertos_main,          // Task function
            "main",                 // Task name
```

```
            MAIN_TASK_STACK_SIZE,   // Stack size
            NULL,                   // Task parameter
            MAIN_TASK_PRIORITY,     // Priority
            gMainTaskStack,         // Stack buffer
            &gMainTaskObj           // Task structure
        );

        // Set affinity BEFORE scheduler starts
        vTaskCoreAffinitySet(gMainTask, (1 << 0));  // Run on Core 0 only

        // Core 0 starts the scheduler
        vTaskStartScheduler();
    }
    else  // Core 1, 2, 3
    {
        // Other cores: Wait for scheduler signal
        while (ullPortSchedularRunning == 0)
        {
            ;  // Spin until Core 0 initializes scheduler
        }

        // Core N joins the scheduler
        xPortStartScheduler();
    }

    // Should never reach here
    return 0;
}
```

### Why This Matters for SMP

1. **Only Core 0** calls `vTaskStartScheduler()` to initialize the kernel
2. **Other cores** wait for the `ullPortSchedularRunning` flag set by Core 0
3. Each core independently calls `xPortStartScheduler()` to join the scheduler
4. This ensures synchronized startup across all cores and prevents race conditions

### Critical Points

• Never call `vTaskStartScheduler()` on multiple cores
• Always set task affinity BEFORE the scheduler starts
• Use the `ullPortSchedularRunning` flag for synchronization
• Core 0 must complete scheduler initialization before other cores proceed

---

## 4. How to Structure Tasks (Core Affinity, Prioritization, etc.)

### Key Principles for SMP Audio Applications

1. Pin audio-related tasks to a fixed core to maintain real-time performance
2. Use core affinity to control which cores run which tasks
3. Prioritize based on real-time requirements
4. Keep ISR handlers lightweight and on the same core as the peripheral

## Setting Core Affinity

### Pin a task to Core 0 only

```
// Bitmask: bit 0 = Core 0
vTaskCoreAffinitySet(xAudioTask, (1 << 0));
```

### Pin a task to Core 0 or Core 1

```
// Bitmask: bit 0 = Core 0, bit 1 = Core 1
vTaskCoreAffinitySet(xAudioTask, (1 << 0) | (1 << 1));
```

### Allow a task to run on any core

```
vTaskCoreAffinitySet(xBackgroundTask, tskNO_AFFINITY);
```

### Pin a task to Core 2

```
// Bitmask: bit 2 = Core 2
vTaskCoreAffinitySet(xLoggingTask, (1 << 2));
```

## Recommended Core Assignment for Audio Applications

|---------|---------|----------|----------|

## Example: Complete Audio Task Setup

```
void create_audio_tasks(void)
{
    // Audio callback task (highest priority, Core 0 only)
    gAudioCallbackTask = xTaskCreateStatic(
        audio_callback_task,
        "AudioCallback",
        AUDIO_CALLBACK_STACK_SIZE,
        NULL,
        AUDIO_PRIORITY_HIGHEST,
        gAudioCallbackStack,
        &gAudioCallbackObj
    );
    vTaskCoreAffinitySet(gAudioCallbackTask, (1 << 0));  // Core 0 only

    // Audio processing task (high priority, Core 0 or 1)
    gAudioProcessTask = xTaskCreateStatic(
        audio_process_task,
        "AudioProcess",
        AUDIO_PROCESS_STACK_SIZE,
        NULL,
        AUDIO_PRIORITY_HIGH,
        gAudioProcessStack,
        &gAudioProcessObj
    );
    vTaskCoreAffinitySet(gAudioProcessTask, (1 << 0) | (1 << 1));  // Core 0 or 1

    // Logging task (low priority, any core)
    gLoggingTask = xTaskCreateStatic(
        logging_task,
        "Logging",
        LOGGING_STACK_SIZE,
        NULL,
        LOGGING_PRIORITY,
        gLoggingStack,
        &gLoggingObj
    );
```

```
    vTaskCoreAffinitySet(gLoggingTask, tskNO_AFFINITY);  // Any core
}
```

---

# 5. McASP Interrupt & Callback Handling

## Important Warning

SMP applications use SGI (Software Generated Interrupt) number 0 for communication between A53 cores. **Do not use SGI interrupt 0 in user applications.**

## Recommended Architecture: ISR-to-Task Pattern

This is the recommended approach for audio processing because it keeps ISR handlers lightweight and real-time safe.

### McASP Completion Callback

```
// Binary semaphore for synchronization
static SemaphoreHandle_t audioFrameReady = NULL;

// McASP completion callback - keep this lightweight
static void mcasp_callback(MCASP_Handle handle, void *arg)
{
    BaseType_t xHigherPriorityTaskWoken = pdFALSE;

    // Minimal work in ISR - just signal that a frame is ready
    xSemaphoreGiveFromISR(audioFrameReady, &xHigherPriorityTaskWoken);

    // Yield to higher priority task if needed
    portYIELD_FROM_ISR(xHigherPriorityTaskWoken);
}

// Initialize the semaphore (call once at startup)
void audio_init(void)
{
    audioFrameReady = xSemaphoreCreateBinary();

    // Register callback with McASP
    MCASP_registerCallback(mcaspHandle, mcasp_callback, NULL);
}
```

### Audio Processing Task

```
void audioProcessingTask(void *arg)
{
    MCASP_Transaction txn;

    while (1)
    {
        // Wait for frame completion signal from ISR
        xSemaphoreTake(audioFrameReady, portMAX_DELAY);

        // Process the audio frame
        process_audio_buffer(audioBuffer);

        // Submit next DMA transfer if needed
        MCASP_submitTransaction(mcaspHandle, &txn);
    }
}
```

### ~~Alternative Architecture: Direct ISR Processing (Not Recommended)~~

~~Only viable for trivial operations (buffer swaps, status flags) with latency less than 1ms.~~

~~**Risks:**~~

~~• ISR can block real-time tasks on other cores~~

~~• Reduced responsiveness of other cores during ISR execution~~

~~• Difficult to debug timing issues~~

~~**Use only if:**~~

~~• Operation takes less than 1 microsecond~~

~~• Only reading or writing simple status flags~~

~~• No FreeRTOS calls other than binary semaphores~~

## Critical ISR Considerations

1. **Keep ISRs on the same core as the peripheral** - McASP peripheral is typically tied to Core 0
2. **Use lightweight primitives** - Prefer binary semaphores, avoid mutexes
3. **Never call blocking FreeRTOS APIs in ISR** - No delays, no queue receives, only post operations
4. **Set ISR priority appropriately** - Should not conflict with kernel operations or other high-priority interrupts
5. **Minimize ISR execution time** - Aim for microsecond-level duration
6. **Test with worst-case scenario** - Ensure ISR + audio task don't exceed real-time deadlines

## ISR Priority Guidelines

• **McASP ISR Priority:** Should be lower than kernel tick interrupt but higher than application tasks

• **Typical setting:** Use `MCASP_ISR_PRIORITY` constant from driver

• **Verify:** Check interrupt priority configuration in SysConfig

---

# 6. SysConfig Settings and Linker Considerations

## Using Reference Examples

Start with SMP and McASP examples as your base configuration. The SMP example demonstrates shared memory allocation, and the McASP example shows SysConfig and DMA buffer setup.

## Key Linker Script Configuration for SMP

The following configurations are already present in the `hello_world_smp` example. Copy this structure to your linker script:

```
MEMORY {
    DDR : ORIGIN = 0x80080000, LENGTH = 0x2000000

    /* Shared memory segments */
    /* Important: On A53, these regions must have MMU entries configured as non-cached */
    USER_SHM_MEM : ORIGIN = 0x82000000, LENGTH = 0x80
    LOG_SHM_MEM  : ORIGIN = 0xA1000000, LENGTH = 0x40000
}

SECTIONS {
    /* Shared memory accessible by all cores */
    .bss.user_shared_mem (NOLOAD) : {
        KEEP(*(.bss.user_shared_mem))
    } > USER_SHM_MEM

    .bss.log_shared_mem (NOLOAD) : {
        KEEP(*(.bss.log_shared_mem))
    } > LOG_SHM_MEM

    /* Per-core stacks MUST be separate for SMP */
    .stack (NOLOAD) : ALIGN(16) {
        __TI_STACK_BASE = .;       /* Core 0 stack */
        . = . + __TI_STACK_SIZE;

        __TI_STACK_BASE1 = .;      /* Core 1 stack */
        . = . + __TI_STACK_SIZE;

        __TI_STACK_BASE2 = .;      /* Core 2 stack */
        . = . + __TI_STACK_SIZE;

        __TI_STACK_BASE3 = .;      /* Core 3 stack */
        . = . + __TI_STACK_SIZE;
    } > DDR
}
```

## SMP Linker Requirements

1. **Each core needs its own stack** - Allocated separately with unique base addresses
2. **Stack sizes must be adequate** - Reserve enough for interrupt context saves on each core
3. **Shared memory must be marked non-cached** - Configured via MMU entries in SysConfig
4. **Symbol naming convention** - `__TI_STACK_BASE`, `__TI_STACK_BASE1`, `__TI_STACK_BASE2`, `__TI_STACK_BASE3`

## SysConfig Considerations for SMP

1. **MMU Configuration** - Add non-cached entries for shared memory regions (USER_SHM_MEM, LOG_SHM_MEM)
2. **Clock Configuration** - Ensure all cores can run at same or compatible frequencies
3. **Interrupt Configuration** - Avoid SGI interrupt 0 (reserved for kernel)
4. **Peripheral Assignment** - Pin McASP and DMA to Core 0 for deterministic behavior

## Shared Memory Declaration

Use the `__attribute__` directive to place variables in shared memory:

```
// Shared data accessible by all cores (non-cached)
```

```
uint32_t shared_buffer[256] __attribute__((section(".bss.user_shared_mem")));

// Per-core data (cached, in regular DDR)
uint32_t core_local_data[256];
```

**Important:** Only place truly shared data in USER_SHM_MEM. Per-core buffers should remain in regular DDR for performance.

---

# 7. Inter-Task Communication

## Message Queues (Recommended for SMP)

**Best for:** Multiple tasks consuming same data, buffering asynchronous events, decoupling producer-consumer tasks

### Example: Sensor Data Pipeline

```
// Message definition
typedef struct {
    uint8_t data[256];
    uint32_t length;
    uint32_t timestamp;
} AudioMessage_t;

// Create queue (one-time initialization)
void audio_queue_init(void)
{
    // Create queue for 10 messages of AudioMessage_t size
    gAudioQueue = xQueueCreate(10, sizeof(AudioMessage_t));
}

// Producer task (runs on any core, e.g., Core 1)
void producer_task(void *arg)
{
    AudioMessage_t msg;

    while (1)
    {
        // Acquire audio data
        msg.length = read_audio_data(msg.data);
        msg.timestamp = Clock_getTicks();

        // Send message to queue
        // Will block if queue is full (wait for consumer to process)
        xQueueSend(gAudioQueue, &msg, portMAX_DELAY);
    }
}

// Consumer 1 (Core 0)
void consumer1_task(void *arg)
{
    AudioMessage_t msg;

    while (1)
    {
        // Receive message from queue
        // Will block until message is available
        if (xQueueReceive(gAudioQueue, &msg, portMAX_DELAY) == pdTRUE)
        {
            process_audio_message(&msg);
        }
    }
}
```

```
// Consumer 2 (Core 2)
void consumer2_task(void *arg)
{
    AudioMessage_t msg;

    while (1)
    {
        if (xQueueReceive(gAudioQueue, &msg, portMAX_DELAY) == pdTRUE)
        {
            analyze_audio_message(&msg);
        }
    }
}
```

### Queue Characteristics

• Each task that receives from the queue gets its own copy of the message

• ISR-safe through `xQueueSendFromISR()` and `xQueueReceiveFromISR()`

• Both blocking and non-blocking modes available

• Automatic buffer management (no manual allocation)

• Overhead per message copy (consider for very high-frequency data)

## Binary Semaphores (For Simple Synchronization)

**Best for:** ISR-to-task signaling, lightweight event notification

```
// Create semaphore
SemaphoreHandle_t audioReady = xSemaphoreCreateBinary();

// ISR signals completion
static void mcasp_isr(void *arg)
{
    BaseType_t xHigherPriorityTaskWoken = pdFALSE;
    xSemaphoreGiveFromISR(audioReady, &xHigherPriorityTaskWoken);
    portYIELD_FROM_ISR(xHigherPriorityTaskWoken);
}

// Task waits for signal
void audio_task(void *arg)
{
    while (1)
    {
        xSemaphoreTake(audioReady, portMAX_DELAY);
        // Process audio
    }
}
```

## Mutexes (For Resource Protection)

**Best for:** Protecting shared resources accessed by multiple tasks on different cores

```
// Create mutex
SemaphoreHandle_t audioBufferLock = xSemaphoreCreateMutex();

// Task 1 accessing shared buffer
void task1(void *arg)
{
    while (1)
    {
        xSemaphoreTake(audioBufferLock, portMAX_DELAY);

        // Safe to access shared_audio_buffer
```

```
        process_buffer(shared_audio_buffer);

        xSemaphoreGive(audioBufferLock);

        vTaskDelay(pdMS_TO_TICKS(10));
    }
}

// Task 2 accessing same buffer
void task2(void *arg)
{
    while (1)
    {
        xSemaphoreTake(audioBufferLock, portMAX_DELAY);

        // Safe to access shared_audio_buffer
        update_buffer(shared_audio_buffer);

        xSemaphoreGive(audioBufferLock);

        vTaskDelay(pdMS_TO_TICKS(10));
    }
}
```

---

# 8. Debugging

## Recommended Approach: UART Logging

CCS (Code Composer Studio) is **not recommended** for A53 SMP debugging. The debugger cannot halt properly at `main()` due to the multi-core entry sequence.

### Use UART Logging via DebugP

```
#include <kernel/dpl/DebugP.h>

void audio_task(void *arg)
{
    DebugP_log("Core %d: audio task started\r\n", Armv8_getCoreId());

    while (1)
    {
        // Wait for frame
        xSemaphoreTake(audioFrameReady, portMAX_DELAY);

        DebugP_log("Core %d: processing frame %d\r\n",
                   Armv8_getCoreId(), frameCount++);

        // Process frame
        process_audio_buffer();
    }
}
```

### Route DebugP to UART0

Configure in SysConfig:

1. Open SysConfig
2. Navigate to Kernel > DPL > Debug
3. Set Debug Log Device to UART0

4. Set baud rate to 115200

### External Debugger

For deeper inspection and breakpoint debugging:

• Use **Lauterbach TRACE32** or similar external debugger

• Connect to JTAG port on AM62x-SK board

• Can stop and inspect all cores simultaneously

## Configuration for Development

Keep `configOPTIMIZE_FOR_LATENCY` set to 0 in `FreeRTOSConfig.h`:

```
#define configOPTIMIZE_FOR_LATENCY 0
```

This enables:

• Runtime assertions

• Stack overflow detection

• Task statistics and task list queries

• Better debugging information

## Debug Output Example

```
[UART] Core 0: audio task started
[UART] Core 1: processing task started
[UART] Core 2: logging task started
[UART] Core 3: utility task started
[UART] Core 0: processing frame 1
[UART] Core 0: processing frame 2
[UART] Core 1: frame analysis complete
```

## Common Debugging Scenarios

### Issue: Task runs on wrong core

**Debug method:** Add log before and after `vTaskCoreAffinitySet()`

```
uint32_t before = Armv8_getCoreId();
vTaskCoreAffinitySet(audioTask, (1 << 0));
uint32_t after = Armv8_getCoreId();
DebugP_log("Core affinity set: was on %d, now on %d\r\n", before, after);
```

### Issue: Audio processing missing deadline

**Debug method:** Log timestamp at each stage

```
uint64_t t1 = Clock_getTicks();
xSemaphoreTake(audioReady, portMAX_DELAY);
uint64_t t2 = Clock_getTicks();
process_audio_buffer();
uint64_t t3 = Clock_getTicks();

DebugP_log("Wait: %lu, Process: %lu (total %lu cycles)\r\n",
```

```
            t2-t1, t3-t2, t3-t1);
```

### Issue: Queue overflow

**Debug method:** Log queue high-water mark

```
UBaseType_t spacesLeft = uxQueueSpacesAvailable(audioQueue);
if (spacesLeft < 2)
{
    DebugP_log("WARNING: Audio queue nearly full! %d spaces left\r\n", spacesLeft);
}
```

---

# Quick Reference

## Core IDs

• Core 0: `Armv8_getCoreId() == 0`

• Core 1: `Armv8_getCoreId() == 1`

• Core 2: `Armv8_getCoreId() == 2`

• Core 3: `Armv8_getCoreId() == 3`

## Common Affinity Masks

• Core 0: `(1 << 0)` or `0x1`

• Core 1: `(1 << 1)` or `0x2`

• Core 0 or 1: `(1 << 0) | (1 << 1)` or `0x3`

• Any core: `tskNO_AFFINITY`

## Priority Levels (Example)

• Highest: Audio callback ISR handler

• High: Audio processing task

• Medium: System tasks, logging

• Low: Background, non-real-time tasks

---

# Support and Resources

• FreeRTOS SMP Documentation: https://www.freertos.org/Documentation/02-Kernel/02-Kernel-features/13-Symmetric-multiprocessing-introduction

• AM62x Technical Reference Manual: Check TI documentation site

• SDK Examples: Located in `examples/` directory

• SysConfig Tool: Part of TI Code Composer Studio

---

**Last Updated:** June 2026

**Target Platform:** AM62x A53 with FreeRTOS SMP

**SDK Version:** 10.01.00.33