**RTOS流和消息缓冲区**

**[**[**任务间通信和同步**](https://www.freertos.org/Inter-Task-Communication.html)**]**

**介绍**

流缓冲区是[RTOS任务](https://www.freertos.org/a00015.html)到RTOS任务，是对任务通信原语的中断。与大多数其他FreeRTOS通信原语不同，它们针对单读取器单写入器方案进行了优化，例如将数据从中断服务例程传递到任务，或从一个微控制器内核传递到双核CPU上的另一个内核。数据通过复制传递–发送者将数据复制到缓冲区中，并通过读取将其复制出缓冲区。

流缓冲区传递连续的字节流。消息缓冲区传递可变大小但离散的消息。消息缓冲区使用流缓冲区进行数据传输。

**重要说明**：在FreeRTOS对象中，流缓冲区实现（消息缓冲区实现也是如此，因为消息缓冲区建立在流缓冲区的顶部）是唯一的，它假定只有一个任务或中断将写入缓冲区（写程序），并且只有一个任务或中断将从缓冲区（读取器）读取。对于作者和读者来说，不同的任务或中断是安全的，但与其他FreeRTOS对象不同，拥有多个不同的作者或多个不同的读者是不安全的。如果存在多个不同的编写器，则应用程序编写器必须将每个调用都置于编写API函数（例如[xStreamBufferSend（）](https://www.freertos.org/xStreamBufferSend.html)）在临界区中，并使用0的发送块时间。同样，如果要有多个不同的读取器，则应用程序编写者必须[将对](https://www.freertos.org/xStreamBufferReceive.html)读取API函数（例如[xStreamBufferReceive（）](https://www.freertos.org/xStreamBufferReceive.html)）的每次调用放在临界区中并使用接收块时间为0。

在本页面：

1. [**介绍**](https://www.freertos.org/RTOS-stream-buffer-example.html#introduction)
2. [**入门**](https://www.freertos.org/RTOS-stream-buffer-example.html#getting_started)
3. [**阻止读取和触发电平**](https://www.freertos.org/RTOS-stream-buffer-example.html#blocking_reads)
4. [**阻止写入**](https://www.freertos.org/RTOS-stream-buffer-example.html#blocking_writes)
5. [**发送和接收完整的宏（供多核使用）**](https://www.freertos.org/RTOS-stream-buffer-example.html#macros)

**介绍**

流缓冲区允许将字节流从中断服务例程传递到任务，或从一个任务传递到另一任务。字节流可以具有任意长度，并且不一定具有开头或结尾。可以一次写入任意数量的字节，并且可以一次读取任意数量的字节。数据通过复制传递–发送者将数据复制到缓冲区中，并通过读取将其复制出缓冲区。

与大多数其他FreeRTOS通信原语不同，流缓冲区针对单读取器单写入器场景进行了优化，例如将数据从中断服务例程传递到任务，或从一个微控制器内核传递到双核CPU上的另一个内核。

通过 在构建中包括FreeRTOS / source / stream\_buffer.c源文件来启用流缓冲区功能。

流缓冲区实现[直接使用任务通知](https://www.freertos.org/RTOS-task-notifications.html)。因此，调用将调用任务置于“阻塞”状态的流缓冲区API函数可以更改调用任务的通知状态和值。

**重要说明**：在FreeRTOS对象中，流缓冲区实现（[消息缓冲区](https://www.freertos.org/RTOS-message-buffer-example.html) 实现也是如此，因为消息缓冲区建立在流缓冲区的顶部）是唯一的，它假定只有一个任务或中断将写入缓冲区（写程序），并且只有一个任务或中断将从缓冲区（读取器）读取。对于作者和读者来说，不同的任务或中断是安全的，但与其他FreeRTOS对象不同，拥有多个不同的作者或多个不同的读者是不安全的。如果存在多个不同的编写器，则应用程序编写器必须将每个调用都置于编写API函数（例如[xStreamBufferSend（）](https://www.freertos.org/xStreamBufferSend.html)）在临界区中，并使用0的发送块时间。同样，如果要有多个不同的读取器，则应用程序编写者必须[将对](https://www.freertos.org/xStreamBufferReceive.html)读取API函数（例如[xStreamBufferReceive（）](https://www.freertos.org/xStreamBufferReceive.html)）的每次调用放在临界区中并使用接收块时间为0。

**入门**

所述**FreeRTOS的/演示/普通/最小/ StreamBufferInterrupt.c**源文件提供了如何使用流缓冲器从中断服务例程的数据传递到一个任务重注释的示例。

有关与[流缓冲区](https://www.freertos.org/RTOS-stream-buffer-API.html)相关的API函数的列表，请参见用户文档的[流缓冲区部分](https://www.freertos.org/RTOS-stream-buffer-API.html)，在许多情况下，包括演示所使用功能的代码段。

**阻止读取和触发电平**

[xStreamBufferReceive（）](https://www.freertos.org/xStreamBufferReceive.html)用于从RTOS任务从流缓冲区中读取数据。 [xStreamBufferReceiveFromISR（）](https://www.freertos.org/xStreamBufferReceiveFromISR.html)用来从中断服务例程（ISR）从流缓冲区中读取数据。

xStreamBufferReceive（）允许指定块时间。如果在任务使用xStreamBufferReceive（）从碰巧为空的流缓冲区中读取时指定了非零的块时间，则该任务将被置于“阻塞”状态（因此它不会消耗任何CPU时间，其他任务也可以运行）直到指定数量的数据在流缓冲区中变得可用，或者块时间到期为止。从阻塞状态中删除等待数据的任务之前，流缓冲区中必须存在的数据量称为流缓冲区的触发级别。例如：

* 如果在读取触发器级别为1的空流缓冲区时阻止了任务，则当将单个字节写入缓冲区或任务的阻止时间到期时，任务将被取消阻止。
* 如果在读取触发器级别为10的空流缓冲区时阻止了任务，则直到该流缓冲区包含至少10个字节或任务的阻止时间到期后，该任务才会被取消阻止。

如果读取任务的阻止时间在达到触发级别之前已过期，则该任务仍将接收，但是实际上有许多字节可用。

**笔记：**

* 将触发级别设置为0是无效的。尝试将触发级别设置为0将导致使用触发级别1。
* 指定大于流缓冲区大小的触发级别也是无效的。

流缓冲区的触发级别最初是[在创建](https://www.freertos.org/xStreamBufferCreate.html)流缓冲区时设置的 ，然后可以使用[xStreamBufferSetTriggerLevel（）](https://www.freertos.org/xStreamBufferSetTriggerLevel.html) API函数进行更改。

**阻止写入**

[xStreamBufferSend（）](https://www.freertos.org/xStreamBufferSend.html)）用于将数据从RTOS任务发送到流缓冲区。 [xStreamBufferSendFromISR（）](https://www.freertos.org/xStreamBufferSendFromISR.html)用来将数据从中断服务程序（ISR）发送到流缓冲区。

如果在任务使用xStreamBufferSend（）写入恰好已满的流缓冲区时指定了非零的块时间，则该任务将被置于“阻塞”状态（这样就不会占用任何CPU时间，其他任务也可以运行）直到流缓冲区中的任何空间变为可用，或者块时间到期为止。

**发送完整的和接收完整的宏**

[另请参阅[有关将消息缓冲区用于双核-核心到核心通信](https://www.freertos.org/articles/001_simple_freertos_core_to_core_communication/simple_freertos_core_to_core_communication_AMP.html)的[博客](https://www.freertos.org/articles/001_simple_freertos_core_to_core_communication/simple_freertos_core_to_core_communication_AMP.html)。]

**sbSEND\_COMPLETED（）（和sbSEND\_COMPLETED\_FROM\_ISR（））**

sbSEND\_COMPLETED（）是一个宏，在将数据写入流缓冲区时会被调用（在FreeRTOS API函数内部）。它采用单个参数，这是已更新的流缓冲区的句柄。

sbSEND\_COMPLETED（）检查是否在流缓冲区上等待某个任务以等待数据，如果是，则将该任务从“阻止”状态中删除。

应用程序编写者可以通过在FreeRTOSConfig.h中提供自己的sbSEND\_COMPLETED（）实现来更改此默认行为。当使用流缓冲区在多核处理器上的内核之间传递数据时，这很有用。在那种情况下，可以实现sbSEND\_COMPLETED（）以在另一个CPU内核中生成中断，然后中断的服务例程可以使用xStreamBufferSendCompletedFromISR（）API函数来检查并在必要时取消阻止正在等待数据的任务。 。该**FreeRTOS的/演示/通用/最小/ MessageBufferAMP.c**源文件提供正是场景的大量注释的例子。

**sbRECEIVE\_COMPLETED（）（和sbRECEIVE\_COMPLETED\_FROM\_ISR（））**

sbRECEIVE\_COMPLETED（）是sbSEND\_COMPLETED（）的接收等效项。从流缓冲区中读取数据时（在FreeRTOS API函数内部）将调用它。宏检查以查看是否有任务在流缓冲区中被阻塞，以等待缓冲区中的空间变得可用，如果有，则将该任务从“阻塞”状态中删除。与sbSEND\_COMPLETED（）一样，通过在FreeRTOSConfig.h中提供替代实现，sbRECEIVE\_COMPLETED（）的默认行为

# RTOS消息缓冲区 [ [流缓冲区和消息缓冲区](https://www.freertos.org/RTOS-stream-message-buffers.html) ]

在本页面：

1. [介绍](https://www.freertos.org/RTOS-message-buffer-example.html#introduction)
2. [入门](https://www.freertos.org/RTOS-message-buffer-example.html#getting_started)
3. [调整消息缓冲区的大小](https://www.freertos.org/RTOS-message-buffer-example.html#sizing)
4. [阻止读写](https://www.freertos.org/RTOS-message-buffer-example.html#blocking_writes)
5. [发送和接收完整的宏（供多核使用）](https://www.freertos.org/RTOS-message-buffer-example.html#macros)

## 介绍

消息缓冲区允许将可变长度的离散消息从中断服务例程传递到任务，或从一个任务传递到另一任务。例如，长度为10、20和123字节的消息都可以写入和读取同一消息缓冲区。与使用[流缓冲区不同](https://www.freertos.org/RTOS-stream-buffer-example.html)，10字节的消息只能作为10字节的消息读出，而不能作为单个字节读出。消息缓冲区建立在流缓冲区之上（也就是说，它们使用流缓冲区实现）。

数据通过副本传递通过消息缓冲区–数据被发送方复制到缓冲区，并通过读取从缓冲区复制出。

另请参阅[configMESSAGE\_BUFFER\_LENGTH\_TYPE](https://www.freertos.org/a00110.html" \l "configMESSAGE_BUFFER_LENGTH_TYPE) 配置参数的定义 。

与大多数其他FreeRTOS通信原语不同，流缓冲区以及消息缓冲区针对单读取器单写入器场景进行了优化，例如将数据从中断服务例程传递到任务，或从一个微控制器内核传递到双核CPU上的另一个内核。

通过 在构建中包含FreeRTOS / source / stream\_buffer.c源文件来启用消息缓冲区功能。

消息缓冲区实现将[直接用于任务通知](https://www.freertos.org/RTOS-task-notifications.html)。因此，调用将调用任务置于“阻塞”状态的消息缓冲区API函数可以更改调用任务的通知状态和值。

**重要说明**：在FreeRTOS对象中，流缓冲区实现（消息缓冲区实现也是如此，因为消息缓冲区建立在流缓冲区的顶部）是唯一的，它假定只有一个任务或中断将写入缓冲区（写程序），并且只有一个任务或中断将从缓冲区（读取器）读取。对于作者和读者来说，不同的任务或中断是安全的，但与其他FreeRTOS对象不同，拥有多个不同的作者或多个不同的读者是不安全的。如果存在多个不同的编写器，则应用程序编写器必须将每个调用都置于编写API函数（例如[xStreamBufferSend（）](https://www.freertos.org/xStreamBufferSend.html)）在临界区中，并使用0的发送块时间。同样，如果要有多个不同的读取器，则应用程序编写者必须[将对](https://www.freertos.org/xStreamBufferReceive.html)读取API函数（例如[xStreamBufferReceive（）](https://www.freertos.org/xStreamBufferReceive.html)）的每次调用放在临界区中并使用接收块时间为0。

### 入门

所述**FreeRTOS/Demo/Common/Minimal/StreamBufferInterrupt.c**源文件提供了如何使用一个消息缓冲器从一个MCU芯通过可变长度的数据到另一个上的多芯MCU重注释的示例。那是相当高级的场景，但是在较简单的单核场景中，用于创建消息，从消息缓冲区发送消息以及从消息缓冲区接收消息的机制是相同的，与演示中的情况不同，无需重写[sbSEND\_COMPLETE（）](https://www.freertos.org/RTOS-stream-buffer-example.html#macros)宏。

有关与[消息缓冲区](https://www.freertos.org/RTOS-message-buffer-API.html)相关的API函数的列表，请参见用户文档的[消息缓冲区部分](https://www.freertos.org/RTOS-message-buffer-API.html)，在许多情况下，包括演示所使用功能的代码段。

## 调整消息缓冲区的大小

为了使消息缓冲区能够处理大小可变的消息，每条消息的长度都要在消息本身之前写入消息缓冲区中（这在FreeRTOS API函数内部发生）。长度存储在类型为size\_t的变量中，在32字节体系结构中，该变量通常为4字节。因此，作为示例，将10个字节的消息写入消息缓冲区实际上将消耗14个字节的缓冲区空间。同样，将100字节的消息写入消息缓冲区实际上将存储104字节的缓冲区空间。

## 阻止读写

[xMessageBufferReceive（）](https://www.freertos.org/xMessageBufferReceive.html)）用于从RTOS任务的消息缓冲区中读取数据。 [xMessageBufferReceiveFromISR（）](https://www.freertos.org/xMessageBufferReceiveFromISR.html)用来从中断服务程序（ISR）的消息缓冲区中读取数据。 [xMessageBufferSend（）](https://www.freertos.org/xMessageBufferSend.html)）用于将数据从RTOS任务发送到消息缓冲区。 [xMessageBufferSendFromISR（）](https://www.freertos.org/xMessageBufferSendFromISR.html)）用于将数据从中断服务例程（ISR）发送到消息缓冲区。

如果在任务使用xMessageBufferReceive（）从碰巧为空的消息缓冲区中读取时指定了非零的阻止时间，则该任务将被置于“阻止”状态（因此它不会消耗任何CPU时间，并且其他任务可以运行）直到消息缓冲区中的数据可用或阻止时间到期为止。

如果在任务使用xMessageBufferSend（）写入恰好已满的消息缓冲区时指定了非零的块时间，则该任务将被置于“阻塞”状态（因此它不会消耗任何CPU时间，并且其他任务可以运行）直到消息缓冲区中的任何一个空间可用或阻止时间到期为止。

## 发送完整的和接收完整的宏

由于消息缓冲区建立在流缓冲区上，因此sbSEND\_COMPLETE（）和sbRECEIVE\_COMPLETE（）宏的行为与[描述流缓冲区](https://www.freertos.org/RTOS-stream-buffer-example.html)的[页面上的描述](https://www.freertos.org/RTOS-stream-buffer-example.html)完全相同 。

**RTOS流缓冲区  
[**[**API**](https://www.freertos.org/a00106.html)**]**

[**RTOS流缓冲区**](https://www.freertos.org/RTOS-stream-buffer-example.html)**API函数：**

* [**xStreamBufferCreate（）**](https://www.freertos.org/xStreamBufferCreate.html)
* [**xStreamBufferCreateStatic（）**](https://www.freertos.org/xStreamBufferCreateStatic.html)
* [**xStreamBufferSend（）**](https://www.freertos.org/xStreamBufferSend.html)
* [**xStreamBufferSendFromISR（）**](https://www.freertos.org/xStreamBufferSendFromISR.html)
* [**xStreamBufferReceive（）**](https://www.freertos.org/xStreamBufferReceive.html)
* [**xStreamBufferReceiveFromISR（）**](https://www.freertos.org/xStreamBufferReceiveFromISR.html)
* [**vStreamBufferDelete（）**](https://www.freertos.org/vStreamBufferDelete.html)
* [**xStreamBufferBytesAvailable（）**](https://www.freertos.org/xStreamBufferBytesAvailable.html)
* [**xStreamBufferSpacesAvailable（）**](https://www.freertos.org/xStreamBufferSpacesAvailable.html)
* [**xStreamBufferSetTriggerLevel（）**](https://www.freertos.org/xStreamBufferSetTriggerLevel.html)
* [**xStreamBufferReset（）**](https://www.freertos.org/xStreamBufferReset.html)
* [**xStreamBufferIsEmpty（）**](https://www.freertos.org/xStreamBufferIsEmpty.html)
* [**xStreamBufferIsFull（）**](https://www.freertos.org/xStreamBufferIsFull.html)

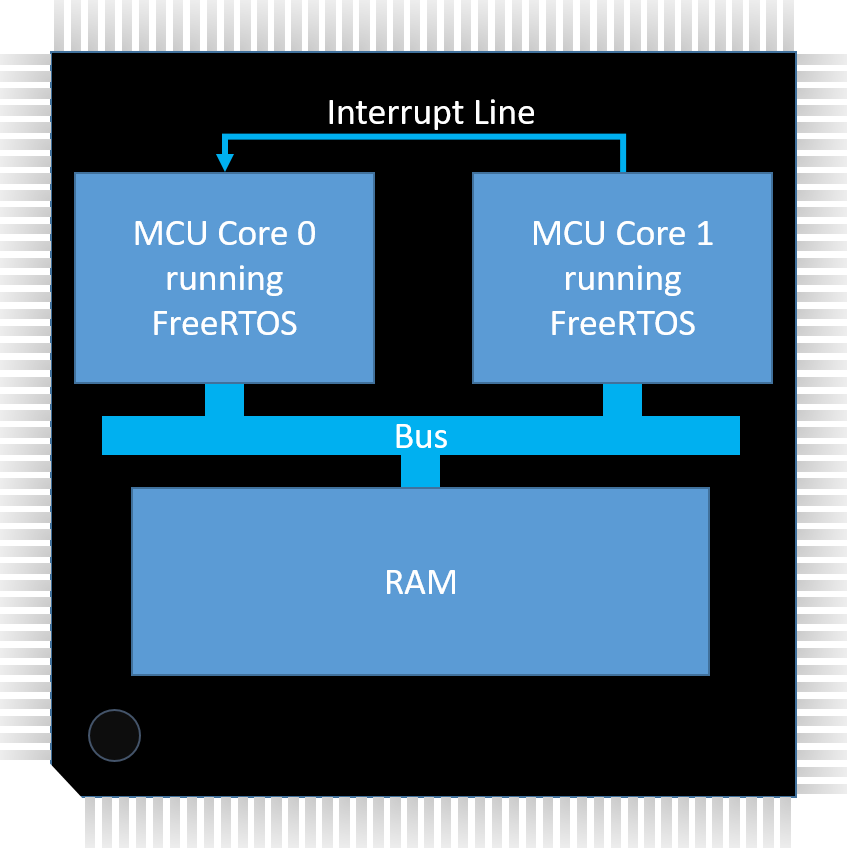
**使用FreeRTOS消息缓冲区的简单多核核心到核心通信**

由[Richard Barry](https://www.freertos.org/author/ribarry.html)在2020年2月18日

[ FreeRTOS下载中的[STM32H745I](https://www.freertos.org/STM32H7_Dual_Core_AMP_RTOS_demo.html)演示提供了下面描述的控制缓冲区方案的有效示例。]

在本文中，我将介绍如何使用[FreeRTOS消息缓冲区](https://www.freertos.org/RTOS-stream-message-buffers.html)实现基本且轻巧的核心到核心通信方案，该[消息缓冲区](https://www.freertos.org/RTOS-stream-message-buffers.html)是无锁循环缓冲区，可以将大小不同的数据包从单个发送方传递到单个接收方。消息缓冲区仅提供数据的传输-它们不施加数据必须遵循的任何格式或更高级别的协议。

在下面描述的用例中，发送和接收任务位于非对称多处理器（AMP）配置中的多核微控制器（MCU）的不同内核上–这意味着每个内核都运行自己的FreeRTOS实例。唯一的硬件要求（除了一个以上的内核外）是一个内核在另一个内核中生成中断的能力，以及两个内核都可以访问的内存区域（共享内存）。消息缓冲区以每个内核上运行的应用程序已知的地址放置在共享内存中。见图1。理想情况下，还将有一个内存保护单元（MPU），以确保只能通过内核的[消息缓冲区API](https://www.freertos.org/RTOS-message-buffer-API.html)来访问[消息缓冲区](https://www.freertos.org/RTOS-message-buffer-API.html)，并最好将共享内存标记为不可缓存。

[](https://www.freertos.org/wp-content/uploads/2019/05/multicore_amp_hardware_configuration.png)  
**图1：硬件拓扑。点击放大。**

以下两个伪代码清单显示了用于发送和接收消息缓冲区的API函数的结构。可以看出，在两种情况下，调用任务都可以选择进入阻塞状态（因此不占用任何CPU周期）以等待操作完成。

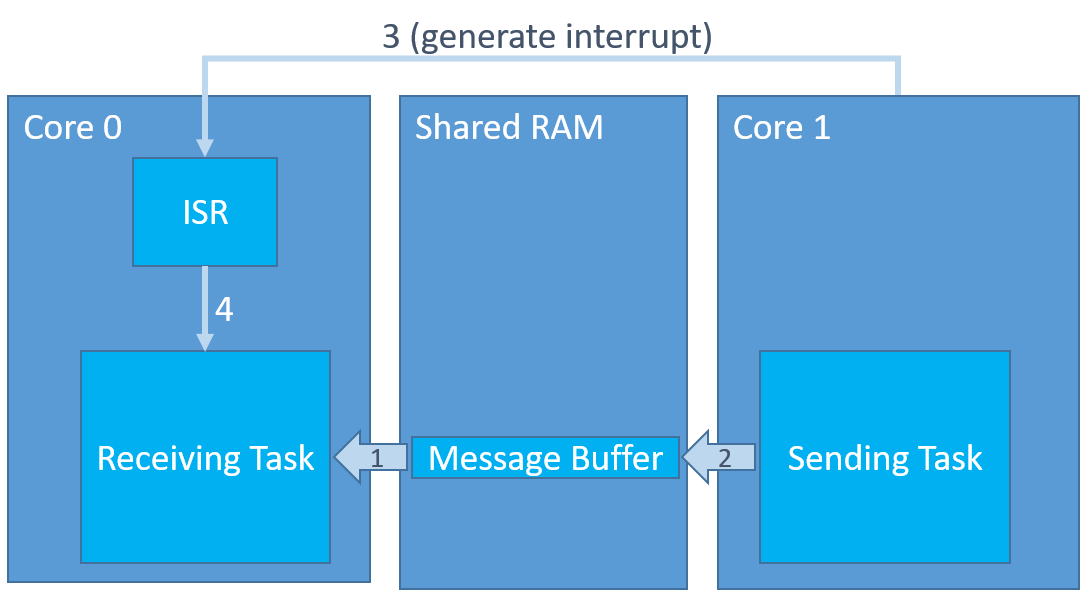
|  |
| --- |
| **xMessageBufferSend（）**  **{**  **/ \*如果指定了超时并且没有足够的时间**  **消息缓冲区中的空间以发送数据，然后**  **进入阻止状态以等待更多空间。\* /**  **如果（超时！= 0）**  **{**  **while（缓冲区中的空间不足&&**  **没有超时等待）**  **{**  **进入阻塞状态以等待缓冲区中的空间**  **}**  **}**  **if（缓冲区中有足够的空间）**  **{**  **将数据写入缓冲区**  **sbSEND\_COMPLETED（）**  **}**  **}**    **用于将数据发送到流缓冲区的简化伪码** |

|  |
| --- |
| **xMessageBufferReceive（）**  **{**  **/ \*如果指定了超时并且缓冲区没有**  **包含任何可以读取的数据，然后输入**  **阻塞状态，以等待缓冲区包含数据。\* /**  **如果（超时！= 0）**  **{**  **while（缓冲区中没有数据&&**  **没有超时等待）**  **{**  **进入阻止状态以等待数据**  **}**  **}**  **if（缓冲区中有数据）**  **{**  **从缓冲区读取数据**  **sbRECEIVE\_COMPLETED（）**  **}**  **}**    **用于从流缓冲区读取数据的简化伪码** |

如果任务在xMessageBufferReceive（）中进入阻塞状态以等待缓冲区包含数据，则将数据发送到缓冲区必须取消阻塞该任务，以便它可以完成其操作。当xMessageBufferSend（）调用sbSEND\_COMPLETED（）（这是一个预处理器宏）时，任务将不受阻碍。

默认的sbSEND\_COMPLETED实现假定发送任务（或中断）和接收任务在FreeRTOS内核的相同实例的控制下并在同一MCU内核上运行。在此AMP示例中，发送任务和接收任务在FreeRTOS内核的两个不同实例的控制下，并且在不同的MCU内核上运行，因此默认的sbSEND\_COMPLETED实现将无法工作（每个FreeRTOS内核实例仅了解任务）在其控制下）。因此，AMP方案要求覆盖sbSEND\_COMPLETED宏（以及可能的sbRECEIVE\_COMPLETED宏，请参见下文），只需在项目的[FreeRTOSConfig.h中](https://www.freertos.org/a00110.html)提供自己的实现即可完成此操作。 文件。重新实现的sbSEND\_COMPLETED（）宏可以简单地触发另一个MCU内核中的中断。然后，中断的处理程序（由一个内核触发但在另一个内核中执行的ISR）必须执行否则将由sbSEND\_COMPLETE的默认实现完成的工作–即，如果任务正在等待从消息中接收数据，则取消阻止任务现在包含数据的缓冲区。ISR通过将消息缓冲区的句柄作为参数传递给xMessageBufferSendCompletedFromISR（）函数来解除对任务的阻塞。此序列由图2中的数字箭头显示，其中发送和接收任务位于不同的MCU内核上：

1. 接收任务尝试从空的消息缓冲区中读取数据，并进入阻止状态以等待数据到达。
2. 发送任务将数据写入消息缓冲区。
3. sbSEND\_COMPLETED（）在正在执行接收任务的内核中触发一个中断。
4. 中断服务例程调用xMessageBufferSendCompletedFromISR（）来解除阻止接收任务，该任务现在可以从缓冲区读取，因为缓冲区不再为空。

[](https://www.freertos.org/wp-content/uploads/2019/05/multicore_amp_single_message_buffer.png)  
**图2：带编号的箭头对应于上面的编号列表，该列表描述了通过消息缓冲区传输一个数据项。 点击放大。**

当只有一个消息缓冲区时，将消息缓冲区的句柄传递到xMessageBufferSendCompletedFromISR（）很容易，但是要考虑有两个或更多消息缓冲区的情况–然后ISR必须首先确定哪个消息缓冲区包含数据。如果消息缓冲区的数量很少，则有几种方法可以实现。例如：

* 如果硬件允许，则每个消息缓冲区可以使用不同的中断线，从而使中断服务程序和消息缓冲区之间保持一对一的映射。
* 中断服务例程可以简单地查询每个消息缓冲区以查看其是否包含数据。
* 可以通过传递元数据（消息是什么，消息的预期接收者是什么，等等）以及实际数据的单个消息缓冲区来代替多个消息缓冲区。

但是，如果存在大量或未知的消息缓冲区，则这些技术效率不高–在这种情况下，可伸缩的解决方案是引入单独的控制消息缓冲区。如下面的代码所示，sbSEND\_COMPLETED（）使用控制消息缓冲区将包含数据的消息缓冲区的句柄传递到中断服务例程中。

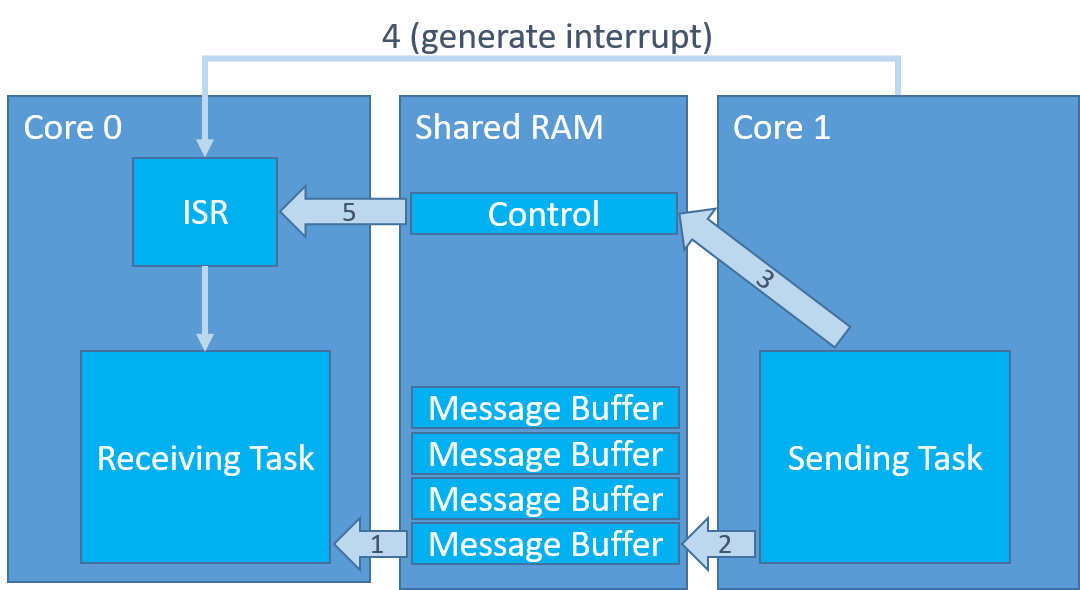
|  |
| --- |
| **/ \*添加到FreeRTOSConfig.h中以覆盖默认实现。\* /**  **#define sbSEND\_COMPLETED（pxStreamBuffer）vGenerateCoreToCoreInterrupt（pxStreamBuffer）**  **/ \*在C文件中实现。\* /**  **无效vGenerateCoreToCoreInterrupt（MessageBufferHandle\_t xUpdatedBuffer）**  **{**  **size\_t写入的字节数。**  **/ \*由FreeRTOSConfig.h中sbSEND\_COMPLETED（）的实现调用。**  **如果由于将数据写入任何消息缓冲区而调用此函数**  **除了控制消息缓冲区，然后写消息的句柄**  **包含数据到控制消息缓冲区的缓冲区，然后引发**  **在另一个内核中中断。如果由于数据被调用而调用此函数**  **写入控制消息缓冲区然后什么也不做。\* /**  **if（xUpdatedBuffer！= xControlMessageBuffer）**  **{**  **BytesWritten =** [**xMessageBufferSend**](https://www.freertos.org/xMessageBufferSend.html)**（xControlMessageBuffer，**  **＆xUpdatedBuffer，**  **sizeof（xUpdatedBuffer），**  **0）;**  **/ \*如果无法写入字节，则控制消息缓冲区**  **太小！\* /**  **configASSERT（BytesWritten == sizeof（xUpdatedBuffer）;**  **/ \*在另一个内核中生成中断（伪代码）。\* /**  **GenerateInterrupt（）;**  **}**  **}**    **使用控制消息缓冲区时，sbSEND\_COMPLETED（）的实现。** |

然后，ISR读取控制消息缓冲区以获取该句柄，然后将该句柄作为参数传递给xMessageBufferSendCompletedFromISR（）。请参见下面的代码清单。

|  |
| --- |
| **void InterruptServiceRoutine（void）**  **{**  **MessageBufferHandle\_t xUpdatedMessageBuffer;**  **BaseType\_t xHigherPriorityTaskWoken = pdFALSE;**  **/ \*接收消息缓冲区的句柄，该句柄包含来自**  **控制消息缓冲区。返回之前，请确保清空缓冲区。\* /**  **while（**[**xMessageBufferReceiveFromISR**](https://www.freertos.org/xMessageBufferReceiveFromISR.html)**（xControlMessageBuffer，**  **＆xUpdatedMessageBuffer，**  **sizeof（xUpdatedMessageBuffer），**  **＆xHigherPriorityTaskWoken）**  **== sizeof（xUpdatedMessageBuffer））**  **{**  **/ \*调用向所有任务发送通知的API函数**  **在xUpdatedMessageBuffer消息缓冲区上阻止，等待数据发送到**  **到达。\* /**  **xMessageBufferSendCompletedFromISR（xUpdatedMessageBuffer，**  **＆xHigherPriorityTaskWoken）;**  **}**  **/ \*** [**正常的FreeRTOS“从中断产生”语义**](https://www.freertos.org/FAQ_API.html#IQRAPI)**，其中**  **xHigherPriorityTaskWoken初始化为pdFALSE，然后将其设置为**  **如果中断解除了优先级高于的任务的阻止，则为pdTRUE**  **当前正在执行的任务。\* /**  **portYIELD\_FROM\_ISR（xHigherPriorityTaskWoken）;**  **}**    **使用控制消息缓冲区时，ISR的实现。** |

图3显示了使用控制消息缓冲区时的顺序。同样，图中的编号箭头与编号的项目有关：

1. 接收任务尝试从空的消息缓冲区中读取数据，并进入阻止状态以等待数据到达。
2. 发送任务将数据写入消息缓冲区。
3. sbSEND\_COMPLETED（）将现在包含数据的消息缓冲区的句柄发送到控制消息缓冲区。
4. sbSEND\_COMPLETED（）在正在执行接收任务的内核中触发一个中断。
5. 中断服务例程从控制消息缓冲区中读取包含数据的消息缓冲区的句柄，然后将该句柄传递给xMessageBufferSendCompletedFromISR（）API函数以取消阻止接收任务，该任务现在可以从缓冲区读取，因为缓冲区不再存在空的。

[](https://www.freertos.org/wp-content/uploads/2019/05/multicore_amp_multiple_message_buffer.png)  
**图3：带编号的箭头对应于上面的带编号的列表，该列表描述了通过使用控制消息缓冲区使ISR知道哪个消息缓冲区包含数据的多个消息缓冲区之一传送一个数据项。 点击放大。**

到目前为止，我们仅考虑了发送任务必须取消阻止接收任务的情况。如果用于核心到核心通信的消息缓冲区可能已满，从而导致发送任务阻塞，那么也有必要考虑接收任务如何解除阻塞发送任务。这可以通过重写sbRECEIVE\_COMPLETED（）的默认实现来完成，该方法与针对sbSEND\_COMPLETED（）所描述的完全相同。

在所有情况下，最好的防御性编程做法是确保任务永远不会在消息队列中无限期地阻塞，以防万一错过了中断，并且始终完全耗尽消息队列，而不是假设每个中断有一条消息。