AN14150

MCX Nx4x的核间通信

第1版 — 2024年1月20日 应用笔记

文档信息

信息	内容
关键词	Nirvana MCX Nx4x、多核通信、Mailbox
摘要	本应用笔记介绍了双核芯片如何使用Mailbox接口进行通信。

1 介绍

MCX Nx4x系列微控制器将两个Arm Cortex M33内核、一个CoolFlux BSP32、一个PowerQuad DSP协处理器、一个NPU，以及多个运行频率达150MHz的高速连接功能相结合。为了支持各种应用，MCX N系列配备了高级串行外设、定时器、高精度模拟和最先进的安全功能。所有的MCX Nx4x产品都包含双区闪存，支持在内部闪存进行边读边写的操作。MCX Nx4x系列还支持大型外部串行存储器配置。

MCX Nx4x是一个双核微控制器系列MCU。CPU0是主Cortex-M33（r0p4-00rel0版）处理器，支持TrustZone-M、浮点单元（FPU）和内存保护单元（MPU）。

MCX Nx4x芯片还包含第二个Cortex-M33实体CPU1，CPU1作为一个副CM33，旨在从主处理器上分担部分工作，以支持特殊的专用应用。此实体的配置不包括MPU、FPU、DSP、ETM、Trustzone-M、安全属性单元（SAU）及协处理器接口。两个内核均支持SYSTICK。

Cortex-M33使用两个32位总线接口（代码总线和系统总线），改变了Harvard存储器架构。总线接口按地址范围激活，且可以在一个特定的总线端口上同时进行指令获取和操作数数据引用。（传统的哈佛架构严格地将指令获取和操作数数据引用分离，将它们分配到特定的总线端口，不考虑访问地址。）代码总线通常用于指令获取和对PC相关数据的访问，而系统总线通常用于对片上和片外存储器的操作数数据的引用以及外设访问。此总线结构完全支持指令获取和数据访问的并发执行，但Cortex-M33的实现可以在每个总线上都生成这两种类型的引用。

2 双核的基本机制

CPU0是先启动的主核，作为主内核。CPU1作为从内核，在芯片启动时保持在复位状态，从而最大限度地降低功耗。因此，要在CPU1中运行或调试应用程序，必须在CPU0上执行一些代码来初始化CPU1。

在双核运行模式下，CPU0和CPU1需要相互通信。MCX Nx 4x提供了一个CPU间的 Mailbox模块，通过使用两种机制来实现这种通信：

- **中断机制**

  第一种机制是对另一个CPU触发中断并向其发送一条简单的消息。要对另一个CPU生成中断，可以使用Cortex_M33（CPU0）中断集（IRQ0SET）或Cortex_M33（CPU1）中断集（IRQ1SET）来设置任意一个中断标志（最多32个）。另一个CPU读取Cortex_M33（CPU0）中断集（IRQ0SET）和Cortex_M33（CPU1）中断集（IRQ1SET），以确定它必须执行的操作。如果不止一个IRQn字段为1，则接收中断的CPU会读取所有这些为1的IRQn字段。接收到中断请求的CPU在完成请求的服务或操作后，在Cortex_M33 (CPU0) Interrupt Clear (IRQ0CLR)或Cortex_M33 (CPU1) Interrupt Clear (IRQ1CLR)的一个或多个字段中写入1来将其清除。

  要查看操作是否完成，发送请求的CPU可以检查IRQn，或者接收原始中断的CPU可以使用同一机制来反向通知发送请求的CPU。用户可根据其具体应用确定Cortex_M33 (CPU0) Interrupt (IRQ0)和Cortex_M33 (CPU1) Interrupt (IRQ1)中每个位的具体含义，并为每个请求位提供附加信息。例如，两个CPU都可以使用预定义的存储区位置来保存有关中断请求的详细信息。

- **互斥机制**

  第二种机制是基于单比特资源分配请求的。MUTEX[EX]位定义了一个互斥请求，读取该位可获得资源的状态。如果此资源是可用的，则可以为读取该寄存器的进程保留该资源。用户可定义该资源的控制，并在获得这个资源的访问权限后，向MUTEX[EX]写入1，以向其他处理器发出信号，表明共享资源现在可用。有关更多信息，请参阅《MCX Nx 4x参考手册》中的MUTEX[EX]说明。

总之，CPU间的 Mailbox提供了以下功能：

- 提供处理器间的通信，允许多个CPU以简便的方式共享资源并相互通信。
- 使每个CPU能够生成多达32个用户定义的中断。
- 使每个CPU能够使用MUTEX寄存器位来获取共享资源。

3 CPU间的Mailbox模块

CPU间的 Mailbox是一个AHB从外设，配备一个单时钟域和复位信号。它有一个互斥寄存器，每个CPU有三个32位的寄存器（IRQ0/1、IRQ0/1SET、IRQ0/1CLR）。

图1所示为CPU间的 Mailbox的框图：

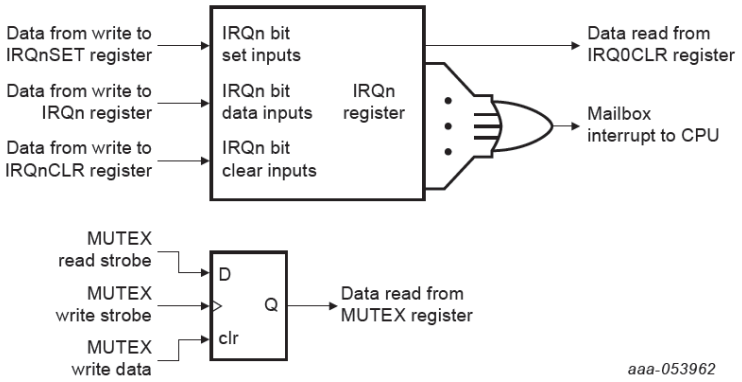

图1. CPU间的Mailbox框图

对于以下两个示例，可以认为CPU1已经启动。默认情况下，CPU1的镜像文件从FLASH存储区加载并执行，但也可以通过在CPU0的工程设置中将预处理器符号CORE1_IMAGE_COPY_TO_RAM定义为1，从RAM中启动CPU1镜像。下面的两个示例均使用MCUXpresso SDK 2.13.1。

此外，无论使用哪种机制，MAILBOX的初始配置都可以按照以下方式完成：

- 通过向SYSCON AHBCLKCTRLSET寄存器中的位31写入1，来启用MAILBOX模块的时钟。
- 通过向PRESETCTRLCLR0寄存器中写入位31（MAILBOX），清除SYSCON寄存器中的外设复位。
- 在NVIC中启用中断，以便当内核在正常模式下接收到中断时能够进行处理。

3.1 中断机制

本节介绍了如何利用中断机制配置和使用MAILBOX进行核间通信。

3.1.1 示例

一旦使用上述指令正确配置了MAILBOX，CPU0和CPU1就可以通过向另一个内核发送消息来进行通信，这个消息能自动生成一个中断。该消息包含在32位寄存器IRQn中，并由用户按需确定。因此，该寄存器的每个位都可以进行不同的配置。向该寄存器的多个位写入1会生成多个中断。

- **CPU0向CPU1发送消息**

1. CPU0通过将消息写入接收内核（CPU1）的IRQ1SET寄存器，来将消息写入副mailbox寄存器。

注：*写入IRQSET寄存器会将1写入IRQ的对应位中。*

1. 在CPU1中生成一个可以在mailbox中断处理程序中处理的中断。
2. CPU1读取其IRQ1寄存器中的消息。
3. CPU1通过将值写入IRQ1CLR，来清除IRQ1寄存器中的相应位。

- **CPU1向CPU0发送消息**

1. CPU1通过向接收内核（CPU0）的IRQ0SET寄存器写入消息，来将消息写入主mailbox寄存器。

注：*写入IRQSET寄存器会将1写入IRQ的对应位中。*

1. 在CPU0中生成一个可以在mailbox中断处理程序中处理的中断。
2. CPU0读取其IRQ0寄存器中的消息。
3. CPU0通过将值写入IRQ0CLR，来清除IRQ0寄存器中的相应位。

图2展示了上述示例。首先，CPU0使用IRQ1SET中的32位向CPU1发送消息（值），然后CPU1将消息（值+1）返回到CPU0。

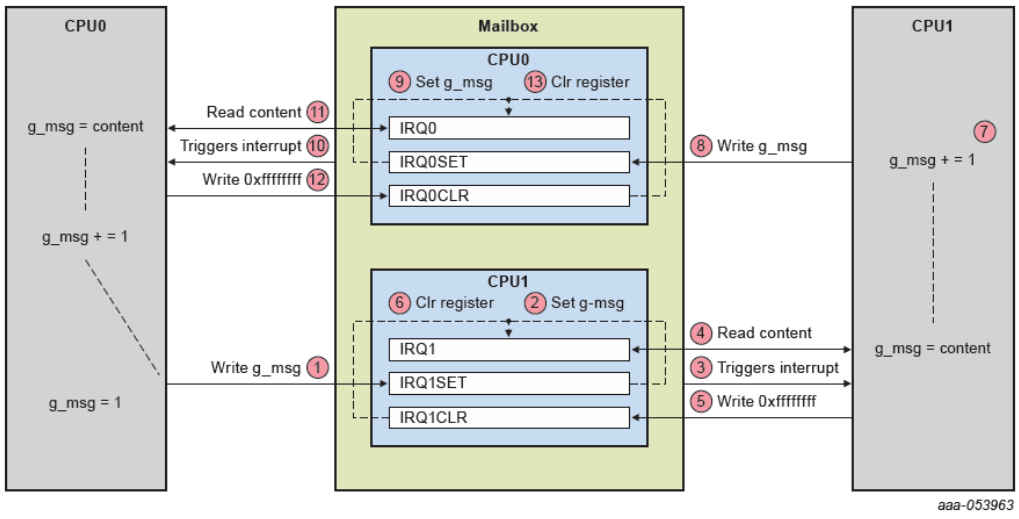

图2. Mailbox中断示例的流程

3.1.2 软件的实现

面向MCX Nx4x的MCUXpresso SDK包含一个使用了CPU间Mailbox中断机制的示例，该示例位于`<SDK_LOCATION>\boards\<board_name>\driver_examples\mailbox\interrupt\`。此SDK示例展示了上述示例场景，其中主内核向副内核Mailbox写入一个值，这会在副内核侧触发一个Mailbox中断。副内核从Mailbox中读取该值，给值加1后将其写入主内核的Mailbox寄存器，这会导致在主内核侧生成一个Mailbox中断。

```
#define PRIMARY_CORE_MAILBOX_CPU_ID   kMAILBOX_CM33_Core0
#define SECONDARY_CORE_MAILBOX_CPU_ID kMAILBOX_CM33_Core1

#define START_EVENT 1234
```

图3. 为应用和开始事件定义的Mailboxes宏

```
55241   #define MAILBOX_BASE                            (0x400B2000u)
55242   /** Peripheral MAILBOX base pointer */
55243   #define MAILBOX                                 ((MAILBOX_Type *)MAILBOX_BASE)
```

图4. Mailbox的基地址

```
114 int main(void)
115 {
116     /* Init board hardware.*/
117     /* attach main clock divide to FLEXCOMM0 (debug console) */
118     CLOCK_AttachClk(BOARD_DEBUG_UART_CLK_ATTACH);
119
120     BOARD_InitBootPins();
121     BOARD_InitBootClocks();
122     BOARD_InitDebugConsole();
123
124     PRINTF("Mailbox interrupt example\r\n");
125
126     /* Init Mailbox */
127     MAILBOX_Init(MAILBOX);
128
129     /* Enable mailbox interrupt */
130     NVIC_EnableIRQ(MAILBOX_IRQn);
131
132 #ifdef CORE1_IMAGE_COPY_TO_RAM
133     /* Calculate size of the image */
134     uint32_t core1_image_size;
135     core1_image_size = get_core1_image_size();
136     PRINTF("Copy CORE1 image to address: 0x%x, size: %d\r\n", CORE1_BOOT_ADDRESS, core1_image_size);
137
138     /* Copy application from FLASH to RAM */
139     memcpy((void *)CORE1_BOOT_ADDRESS, (void *)CORE1_IMAGE_START, core1_image_size);
140 #endif
141
142     /* Start the secondary core */
143     start_secondary_core(CORE1_BOOT_ADDRESS);
144
145     /* Wait for start and initialization of secondary core */
146     while (!g_secondary_core_started)
147         ;
148
149     PRINTF("Write to the secondary core mailbox register: %d\r\n", g_msg);
150     /* Write g_msg to the secondary core mailbox register - it causes interrupt on the secondary core */
151     MAILBOX_SetValue(MAILBOX, SECONDARY_CORE_MAILBOX_CPU_ID, g_msg);
152
153     while (1)
154     {
155         __WFI();
156     }
157 }
158
```

图5. CPU0的主函数

```
88 /* When the secondary core writes to the primary core mailbox register it causes call of this irq handler,
89    in which the received value is read, incremented and sent again to the secondary core */
90 void MAILBOX_IRQHandler()
91 {
92     if (!g_secondary_core_started)
93     {
94         if (START_EVENT == MAILBOX_GetValue(MAILBOX, PRIMARY_CORE_MAILBOX_CPU_ID))
95         {
96             g_secondary_core_started = true;
97         }
98         MAILBOX_ClearValueBits(MAILBOX, PRIMARY_CORE_MAILBOX_CPU_ID, 0xffffffff);
99     }
100    else
101    {
102        g_msg = MAILBOX_GetValue(MAILBOX, PRIMARY_CORE_MAILBOX_CPU_ID);
103        MAILBOX_ClearValueBits(MAILBOX, PRIMARY_CORE_MAILBOX_CPU_ID, 0xffffffff);
104        PRINTF("Read value from the primary core mailbox register: %d\r\n", g_msg);
105        g_msg++;
106        PRINTF("Write to the secondary core mailbox register: %d\r\n", g_msg);
107        MAILBOX_SetValue(MAILBOX, SECONDARY_CORE_MAILBOX_CPU_ID, g_msg);
108    }
109 }
```

图6. CPU0的Mailbox中断处理程序

```
int main(void)
{
    /* Init board hardware.*/
    /* set BOD VBAT level to 1.65V */
    BOARD_InitBootPins();

    /* Initialize Mailbox */
    MAILBOX_Init(MAILBOX);

    /* Enable mailbox interrupt */
    NVIC_EnableIRQ(MAILBOX_IRQn);

    /* Let the other side know the application is up and running */
    MAILBOX_SetValue(MAILBOX, PRIMARY_CORE_MAILBOX_CPU_ID, (uint32_t)START_EVENT);

    while (1)
    {
        __WFI();
    }
}
```

图7. CPU1的主函数

```
void MAILBOX_IRQHandler()
{
    g_msg = MAILBOX_GetValue(MAILBOX, SECONDARY_CORE_MAILBOX_CPU_ID);
    MAILBOX_ClearValueBits(MAILBOX, SECONDARY_CORE_MAILBOX_CPU_ID, 0xffffffff);
    g_msg++;
    MAILBOX_SetValueBits(MAILBOX, PRIMARY_CORE_MAILBOX_CPU_ID, g_msg);
}
```

图8. CPU1的Mailbox中断处理程序

图9所示为在终端上运行工程的结果。

```
Mailbox interrupt example
Write to the secondary core mailbox register: 1
Read value from the primary core mailbox register: 2
Write to the secondary core mailbox register: 3
Read value from the primary core mailbox register: 4
Write to the secondary core mailbox register: 5
Read value from the primary core mailbox register: 6
Write to the secondary core mailbox register: 7
Read value from the primary core mailbox register: 8
Write to the secondary core mailbox register: 9
Read value from the primary core mailbox register: 10
Write to the secondary core mailbox register: 11
Read value from the primary core mailbox register: 12
Write to the secondary core mailbox register: 13
Read value from the primary core mailbox register: 14
Write to the secondary core mailbox register: 15
```

图9. 串行终端上的CPU间Mailbox中断示例的输出

3.2 互斥机制

本节介绍了如何利用互斥机制配置和使用MAILBOX进行核间通信。

3.2.1 示例

当按照上述指令（**Inter-CPU Mailbox**）正确配置了MAILBOX后，CPU0和CPU1也可以通过使用互斥机制进行通信，以连续共享和修改变量。CPU0首先使用上一节所述的机制，将共享变量的地址发送到CPU1。

一旦共享了该内存的位置地址，两个内核就都可以尝试在while循环中获取互斥量。当互斥量可用时，一个内核会获得互斥量并更新共享变量，然后设置互斥量以释放它，并允许从另一个内核访问此共享变量。

3.2.2 软件的实现

面向MCX Nx4x的MCUXpresso SDK包含一个使用CPU 间的Mailbox中断机制的示例，该示例位于<SDK_LOCATION>\boards\<board_name>\driver_examples\mailbox\mutex\。此示例展示了上述示例场景，其中主内核将共享变量地址写入到副内核的Mailbox，在副内核侧触发一个Mailbox中断，以处理共享变量。然后在while循环中，两个内核都尝试获取互斥量。当一个内核获得了互斥量后，它通过将共享变量的值加1来更新此共享变量，并释放互斥量。

图10所示为Mailbox的互斥示例流程。有关此示例的中断机制的详细信息，请参考前面的示例。

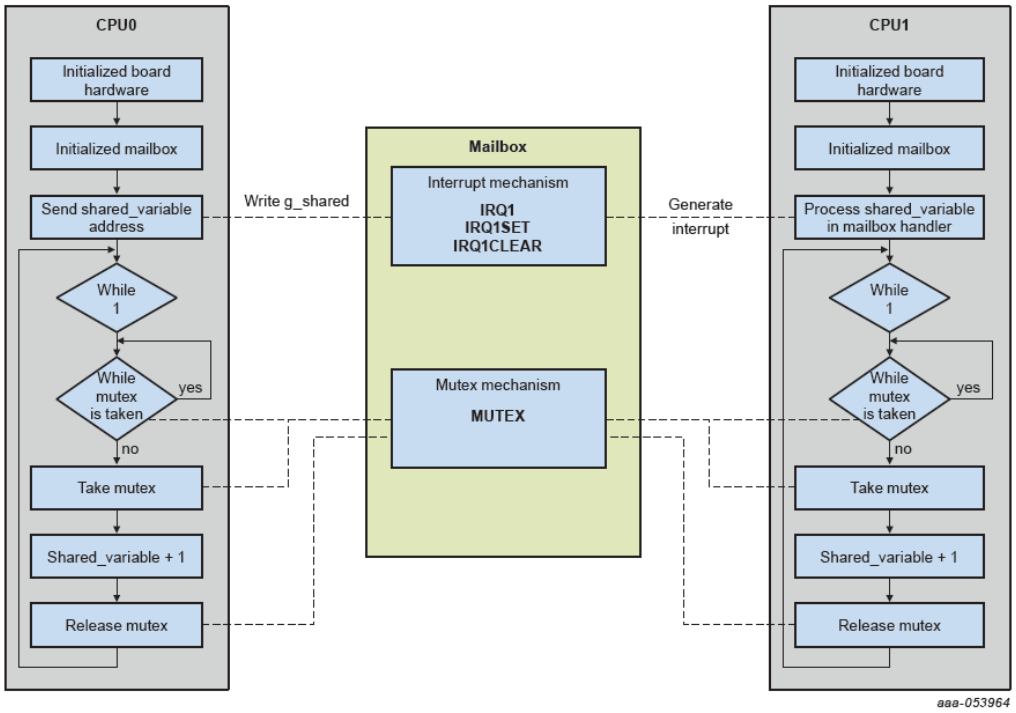

图10. Mailbox互斥示例的流程

```c
int main(void)
{
    /* Init board hardware.*/
    /* attach main clock divide to FLEXCOMM0 (debug console) */
    CLOCK_AttachClk(BOARD_DEBUG_UART_CLK_ATTACH);

    BOARD_InitBootPins();
    BOARD_InitBootClocks();
    BOARD_InitDebugConsole();

    PRINTF("Mailbox mutex example\r\n");

    /* Init Mailbox */
    MAILBOX_Init(MAILBOX);

    /* Enable mailbox interrupt */
    NVIC_EnableIRQ(MAILBOX_IRQn);

#ifdef CORE1_IMAGE_COPY_TO_RAM
    /* Calculate size of the image */
    uint32_t core1_image_size;
    core1_image_size = get_core1_image_size();
    PRINTF("Copy CORE1 image to address: 0x%x, size: %d\r\n", CORE1_BOOT_ADDRESS, core1_image_size);

    /* Copy application from FLASH to RAM */
    memcpy((void *)CORE1_BOOT_ADDRESS, (void *)CORE1_IMAGE_START, core1_image_size);
#endif

    /* Start the secondary core */
    start_secondary_core(CORE1_BOOT_ADDRESS);

    /* Wait for start and initialization of secondary core */
    while (!g_secondary_core_started)
        ;

    /* Send address of shared variable to the secondary core */
    MAILBOX_SetValue(MAILBOX, SECONDARY_CORE_MAILBOX_CPU_ID, (uint32_t)&g_shared);

    while (1)
    {
        /* Get Mailbox mutex */
        while (MAILBOX_GetMutex(MAILBOX) == 0)
            ;

        /* The core0 has mutex, can change shared variable g_shared */
        g_shared++;

        PRINTF("Core0 has mailbox mutex, update shared variable to: %d\r\n", g_shared);

        /* Set mutex to allow access other core to shared variable */
        MAILBOX_SetMutex(MAILBOX);

        /* Add several nop instructions to allow the opposite core to get the mutex */
        __asm("nop");
        __asm("nop");
        __asm("nop");
        __asm("nop");
        __asm("nop");
        __asm("nop");
        __asm("nop");
        __asm("nop");
        __asm("nop");
        __asm("nop");
    }
}
```

图11. CPU0的主函数

```c
void MAILBOX_IRQHandler()
{
    if (START_EVENT == MAILBOX_GetValue(MAILBOX, PRIMARY_CORE_MAILBOX_CPU_ID))
    {
        g_secondary_core_started = true;
    }
    MAILBOX_ClearValueBits(MAILBOX, PRIMARY_CORE_MAILBOX_CPU_ID, 0xffffffff);
}
```

图12. CPU0的Mailbox中断处理程序

```c
int main(void)
{
    /* Init board hardware */
    /* set BOD VBAT level to 1.65V */
    BOARD_InitBootPins();
    BOARD_InitDebugConsole();

    /* Initialize Mailbox */
    MAILBOX_Init(MAILBOX);

    /* Enable mailbox interrupt */
    NVIC_EnableIRQ(MAILBOX_IRQn);

    /* Let the other side know the application is up and running */
    MAILBOX_SetValue(MAILBOX, PRIMARY_CORE_MAILBOX_CPU_ID, (uint32_t)START_EVENT);

    while (1)
    {
        /* Get Mailbox mutex */
        while (MAILBOX_GetMutex(MAILBOX) == 0)
            ;

        /* The core1 has mutex, can change shared variable g_shared */
        if (g_shared != NULL)
        {
            (*g_shared)++;
            PRINTF("Core1 has mailbox mutex, update shared variable to: %d\r\n", *g_shared);
        }

        /* Set mutex to allow access other core to shared variable */
        MAILBOX_SetMutex(MAILBOX);

        /* Add several nop instructions to allow the opposite core to get the mutex */
        __asm("nop");
        __asm("nop");
        __asm("nop");
        __asm("nop");
        __asm("nop");
        __asm("nop");
        __asm("nop");
        __asm("nop");
        __asm("nop");
        __asm("nop");
    }
}

```

图13. CPU1的主函数

```c
/* Pointer to shared variable by both cores, before changing of this variable the
   cores must first take Mailbox mutex, after changing the shared variable must
   retrun mutex */
volatile uint32_t *g_shared = NULL;

/*******************************************************************************
 * Code
 ******************************************************************************/
void MAILBOX_IRQHandler()
{
    g_shared = (uint32_t *)MAILBOX_GetValue(MAILBOX, SECONDARY_CORE_MAILBOX_CPU_ID);
    MAILBOX_ClearValueBits(MAILBOX, SECONDARY_CORE_MAILBOX_CPU_ID, 0xffffffff);
    __DSB();
}
```

图14. CPU1的Mailbox中断处理程序

```
Core1 has mailbox mutex, update shared variable to: 1
Core0 has mailbox mutex, update shared variable to: 2
Core1 has mailbox mutex, update shared variable to: 3
Core0 has mailbox mutex, update shared variable to: 4
Core1 has mailbox mutex, update shared variable to: 5
Core0 has mailbox mutex, update shared variable to: 6
Core1 has mailbox mutex, update shared variable to: 7
Core0 has mailbox mutex, update shared variable to: 8
Core1 has mailbox mutex, update shared variable to: 9
Core0 has mailbox mutex, update shared variable to: 10
Core1 has mailbox mutex, update shared variable to: 11
Core0 has mailbox mutex, update shared variable to: 12
Core1 has mailbox mutex, update shared variable to: 13
Core0 has mailbox mutex, update shared variable to: 14
Core1 has mailbox mutex, update shared variable to: 15
Core0 has mailbox mutex, update shared variable to: 16
Core1 has mailbox mutex, update shared variable to: 17
Core0 has mailbox mutex, update shared variable to: 18
Core1 has mailbox mutex, update shared variable to: 19
Core0 has mailbox mutex, update shared variable to: 20
```

图15. 串行终端上的CPU间Mailbox互斥示例的输出

4 结论

本应用笔记简要介绍了MCX Nx4x芯片的双核机制，和一些基于MCX Nx4x SDK中MAILBOX驱动示例的双核通信应用程序，并展示了如何根据具体用例来使用中断和互斥机制。

5 关于本文中源代码的说明

本文中所示的示例代码具有以下版权和BSD-3-Clause许可：

2024年恩智浦版权所有。在满足以下条件的情况下，允许以源代码和二进制文件的形式重新分发和使用本源代码（无论是否经过修改）：

1. 重新分发源代码必须保留上述版权声明、这些条件和以下免责声明。
2. 以二进制文件形式重新分发时，必须在文档和/或随分发提供的其他材料中必须复制上述版权声明、这些条件和以下免责声明。
3. 未经事先书面许可，不得使用版权所有者的姓名或参与者的姓名为本软件的衍生产品进行背书或推广。

本软件由版权所有者和参与者“按原样”提供，不承担任何明示或暗示的担保责任，包括但不限于对适销性和特定用途适用性的暗示保证。在任何情况下，无论因何种原因或根据何种法律条例，版权所有者或参与者均不对因使用本软件而导致的任何直接、间接、偶然、特殊、惩戒性或后果性损害（包括但不限于采购替代商品或服务；使用损失、数据损失或利润损失或业务中断）承担责任，无论是因合同、严格责任还是侵权行为（包括疏忽或其他原因）造成的，即使事先被告知有此类损害的可能性也不例外。

6 修订历史

表1. 修订历史

文档ID	发布日期	说明
AN14150 v.1.0	2024年1月20日	初版发布

Legal information

Definitions

Draft — A draft status on a document indicates that the content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included in a draft version of a document and shall have no liability for the consequences of use of such information.

Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Terms and conditions of commercial sale — NXP Semiconductors products are sold subject to the general terms and conditions of commercial sale, as published at https://www.nxp.com.cn/profile/terms, unless otherwise agreed in a valid written individual agreement. In case an individual agreement is concluded only the terms and conditions of the respective agreement shall apply. NXP Semiconductors hereby expressly objects to applying the customer's general terms and conditions with regard to the purchase of NXP Semiconductors products by customer.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Suitability for use in non-automotive qualified products — Unless this document expressly states that this specific NXP Semiconductors product is automotive qualified, the product is not suitable for automotive use. It is neither qualified nor tested in accordance with automotive testing or application requirements. NXP Semiconductors accepts no liability for inclusion and/or use of non-automotive qualified products in automotive equipment or applications.

In the event that customer uses the product for design-in and use in automotive applications to automotive specifications and standards, customer (a) shall use the product without NXP Semiconductors' warranty of the product for such automotive applications, use and specifications, and (b) whenever customer uses the product for automotive applications beyond NXP Semiconductors' specifications such use shall be solely at customer's own risk, and (c) customer fully indemnifies NXP Semiconductors for any liability, damages or failed product claims resulting from customer design and use of the product for automotive applications beyond NXP Semiconductors' standard warranty and NXP Semiconductors' product specifications.

Translations — A non-English (translated) version of a document, including the legal information in that document, is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

Security — Customer understands that all NXP products may be subject to unidentified vulnerabilities or may support established security standards or specifications with known limitations. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer's applications and products. Customer's responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer's applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately.

Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP.

NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

NXP B.V. — NXP B.V. is not an operating company and it does not distribute or sell products.

Trademarks

Notice: All referenced brands, product names, service names, and trademarks are the property of their respective owners.

NXP — wordmark and logo are trademarks of NXP B.V.

目录

For more information, please visit: https://www.nxp.com.cn

Date of release: 20 January 2024
Document identifier: AN14150