AN14150

MCX Nx4x Inter-Core Communication

Rev. 1 — 20 January 2024 Application note

Document information

Information	Content
Keywords	Nirvana MCX Nx4x, multi core communication, Mailbox
Abstract	This application note introduces how dual core devices can communicate using the Mailbox interface.

1 Introduction

The MCX Nx4x series microcontrollers combine two Arm Cortex M33 cores with a CoolFlux BSP32, a PowerQuad DSP Co-processor, an NPU, and multiple high-speed connectivity options running at 150 MHz. To support a wide variety of applications, the MCX N-series includes advanced serial peripherals, timers, high-precision analog, and state-of-the-art security features. All MCX Nx4x products include dual-bank flash that supports read while write operation from internal flash. The MCX Nx4x series also supports large external serial memory configurations.

The MCX Nx4x series is a dual-core microcontroller family. CPU0 is the primary Cortex-M33 (ver r0p4-00rel0) processor that supports TrustZone-M, Floating Point Unit (FPU), and Memory Protection Unit (MPU).

The MCX Nx4x device also includes a second instance of Cortex-M33, CPU1 that is the secondary CM33 intended to offload work from the main processor to support special dedicated applications. The configuration of this instance does not include MPU, FPU, DSP, ETM, Trustzone-M, Secure Attribution Unit (SAU) or co-processor interface. SYSTICK is supported on both cores.

Cortex-M33 implements a modified Harvard memory architecture using two 32-bit bus interfaces: the Code and System buses. The bus interfaces are activated by address range and can include both instruction fetches and operand data references on a given bus port. (A traditional Harvard architecture strictly separates instruction fetches and operand data references onto specific bus ports regardless of access address.) The Code bus is typically used for instruction fetching and data accesses of PC-relative data, while the system bus is typically used for operand data references to the on-chip and off-chip memories and peripheral accesses. The bus structure fully supports concurrent instruction fetch and data access, but the Cortex-M33 implementations can generate both types of references on each bus.

2 Dual-core basic mechanism

CPU0 is the main core that boots first and acts as the Master core. CPU1 acts as the Slave core and is held in reset when the device boots up to minimize power consumption. Therefore, to run or debug applications in the CPU1, some code must be executed on CPU0 to initialize CPU1.

In dual-core running mode, CPU0 and CPU1 need to communicate with each other. The MCX Nx 4x provides an Inter-CPU Mailbox module (MAILBOX) to facilitate this communication by using two mechanisms:

- **Interrupts mechanism**
  The first mechanism triggers an interrupt, along with a simple message, to another CPU. To generate an interrupt to another CPU, you can set any one of the interrupt flags (up to 32) by using Cortex_M33 (CPU0) Interrupt Set (IRQ0SET) or Cortex_M33 (CPU1) Interrupt Set (IRQ1SET). The other CPU reads Cortex_M33 (CPU0) Interrupt Set (IRQ0SET) and Cortex_M33 (CPU1) Interrupt Set (IRQ1SET) to determine the action it must perform. If more than one IRQn field is 1, the CPU receiving the interrupts reads all of them. After completing the requested service or operation, the CPU that receives the interrupt requests, writes 1 to one or more fields in Cortex_M33 (CPU0) Interrupt Clear (IRQ0CLR) or Cortex_M33 (CPU1) Interrupt Clear (IRQ1CLR) to clear them.
  To check whether the operation is complete, the CPU sending the request can check IRQn, or the CPU receiving the original interrupt can use the same mechanism on the reverse to notify the requesting CPU. The user determines the particular meaning of each bit in Cortex_M33 (CPU0) Interrupt (IRQ0) and Cortex_M33 (CPU1) Interrupt (IRQ1) for their specific application, and provides additional information to each request bit. For example, both the CPUs can use predefined memory locations to hold detailed information about the interrupt requests.
- **Mutex mechanism**
  The second mechanism is based on a single-bit resource allocation request. The MUTEX[EX] bit defines a mutual exclusion request, and reading that bit provides the status of the resource. If the resource is available, it can be reserved for the process that reads the register. The user defines the resource control, and after

All information provided in this document is subject to legal disclaimers.

© 2024 NXP B.V. All rights reserved.

getting access to the resource, can write 1 to MUTEX[EX] to signal to the other processor that the shared resource is now available. For more information, see the MUTEX[EX] description in the MCX Nx 4x Reference Manual.

In summary, the Inter-CPU Mailbox provides the below features:

- Provides interprocessor communication, allowing multiple CPUs to share resources and communicate with each other in a simple manner.
- Enables each CPU to generate up to 32 user-defined interrupts.
- Enables each CPU to claim a shared resource using the MUTEX register bit.

3 Inter-CPU Mailbox

This Inter-CPU Mailbox is an AHB slave peripheral with a single clock domain and reset. The Inter-CPU Mailbox has one mutex register, and three 32-bit registers per CPU (IRQ0/1, IRQ0/1SET, iRQ0/1CLR).

Figure 1 shows the block diagram of the Inter-CPU Mailbox:

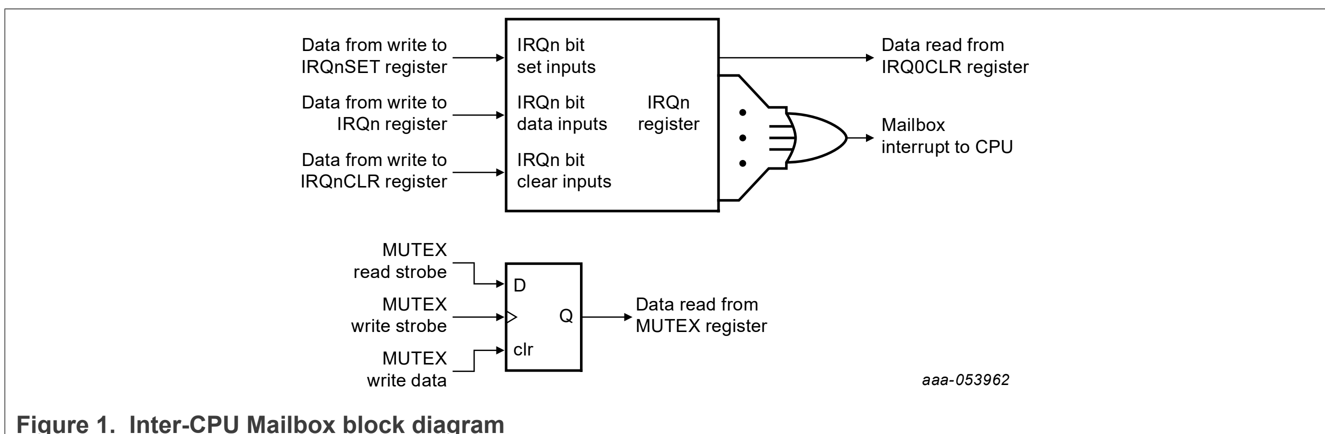

Figure 1. Inter-CPU Mailbox block diagram

For the following two examples, one can consider that the CPU1 is already booted. By default, the CPU1 image is loaded and executed from the FLASH memory, however there is the possibility to boot the CPU1 image from RAM by defining the preprocessor symbol CORE1_IMAGE_COPY_TO_RAM to 1 in the CPU0 project settings. The two examples below use the MCUXpresso SDK 2.13.1.

Also, the initial configuration of the MAILBOX can be accomplished as follow, whatever the mechanism used:

- Enable the clock to the MAILBOX module by writing a 1 to bit 31 in the SYSCON AHBCLKCTRLSET register.
- Clear the peripheral reset in the SYSCON register by writing bit 31 (MAILBOX) to the PRESETCTRLCLR0 register.
- Enable the interrupt in the NVIC to handle it when the core receives an interrupt in normal mode.

3.1 Interrupt mechanism

This section provides information about how to configure and use the MAILBOX for inter-core communication using the interrupt mechanism.

3.1.1 Example

Once the MAILBOX is properly configured using the above instruction, CPU0 and CPU1 can communicate by sending messages that generate automatically an interrupt to the other core. Messages are contained in the 32-bits register IRQn and determined by the user based on requirements. Therefore, each bit of this register can be configured differently. Writing 1 to multiple bits of this register generates more than one interrupt.

All information provided in this document is subject to legal disclaimers.

© 2024 NXP B.V. All rights reserved.

- **CPU0 sends message to CPU1**

1. CPU0 writes a message to the secondary mailbox register by writing the message into the IRQ1SET register of the receiver core (CPU1).

Note: *Writing into the IRQSET register writes 1 to the corresponding bit of IRQ.*

1. An interrupt is generated in CPU1 that can be processed in the mailbox interrupt handler.
2. CPU1 reads the message in its IRQ1 register.
3. CPU1 clears the wanted bits in the IRQ1 register by writing the value into IRQ1CLR.

- **CPU1 sends message to CPU0**

1. CPU1 writes a message to the primary mailbox register by writing the message into the IRQ0SET register of the receiver core (CPU0).

Note: *Writing into the IRQSET register writes 1 to the corresponding bit of IRQ.*

1. An interrupt is generated in CPU0 that can be processed in the mailbox interrupt handler.
2. CPU0 reads the message in the IRQ0 register.
3. CPU0 clears the wanted bits in the IRQ0 register by writing the value into IRQ0CLR.

Figure 2 illustrates the example above. First, CPU0 sends a message (value) to CPU1 using 32 bits in IRQ1SET, then CPU1 sends back the message (value + 1) to CPU0.

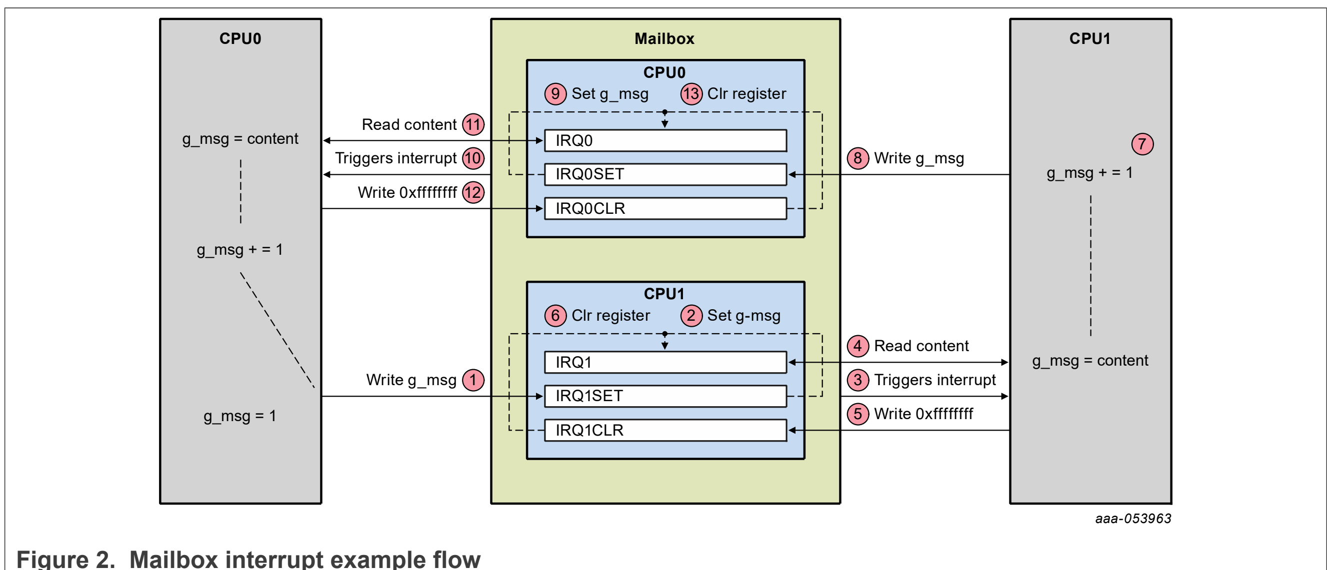

Figure 2. Mailbox interrupt example flow

3.1.2 Software implementation

MCUXpresso SDK for MCX Nx4x contains an example for using the interrupt mechanism of the Inter-CPU Mailbox at `<SDK_LOCATION>\boards\<board_name>\driver_examples\mailbox\interrupt\`. This SDK example shows the scenario explained above, where the primary core writes a value to the secondary core mailbox, it causes a mailbox interrupt on the secondary core side. The secondary core reads the value from the mailbox. It increments and writes it to the mailbox register for the primary core, which causes mailbox interrupt on the primary core side.

```
#define PRIMARY_CORE_MAILBOX_CPU_ID   kMAILBOX_CM33_Core0
#define SECONDARY_CORE_MAILBOX_CPU_ID kMAILBOX_CM33_Core1

#define START_EVENT 1234
```

Figure 3. Mailboxes macros defined for the application and start event

All information provided in this document is subject to legal disclaimers.

© 2024 NXP B.V. All rights reserved.

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

Figure 4. Mailbox base address

```
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 
```

Figure 5. Main function of CPU0

```
 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 }
```

Figure 6. Mailbox interrupt handler CPU0

All information provided in this document is subject to legal disclaimers.

© 2024 NXP B.V. All rights reserved.

```
45 int main(void)
46 {
47     /* Init board hardware.*/
48     /* set BOD VBAT level to 1.65V */
49     BOARD_InitBootPins();
50
51     /* Initialize Mailbox */
52     MAILBOX_Init(MAILBOX);
53
54     /* Enable mailbox interrupt */
55     NVIC_EnableIRQ(MAILBOX_IRQn);
56
57     /* Let the other side know the application is up and running */
58     MAILBOX_SetValue(MAILBOX, PRIMARY_CORE_MAILBOX_CPU_ID, (uint32_t)START_EVENT);
59
60     while (1)
61     {
62         __WFI();
63     }
64 }
```

Figure 7. Main function of CPU1

```
34 void MAILBOX_IRQHandler()
35 {
36     g_msg = MAILBOX_GetValue(MAILBOX, SECONDARY_CORE_MAILBOX_CPU_ID);
37     MAILBOX_ClearValueBits(MAILBOX, SECONDARY_CORE_MAILBOX_CPU_ID, 0xffffffff);
38     g_msg++;
39     MAILBOX_SetValueBits(MAILBOX, PRIMARY_CORE_MAILBOX_CPU_ID, g_msg);
40 }
```

Figure 8. Mailbox interrupt handler CPU1

Figure 9 shows the result of running the project displayed in the terminal.

```
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
```

Figure 9. Inter-CPU mailbox interrupt example output on a serial terminal

3.2 Mutex mechanism

This section provides information about how to configure and use the MAILBOX for inter-core communication using the mutex mechanism.

All information provided in this document is subject to legal disclaimers.

© 2024 NXP B.V. All rights reserved.

3.2.1 Example

Once the MAILBOX is properly configured following the above instruction (Inter-CPU Mailbox), CPU0 and CPU1 can also communicate by using a mutex to share and modify a variable consecutively. CPU0 starts by sending the address of the shared variable to CPU1 by using the mechanism seen in the previous chapter.

Once that memory location address has been shared, both cores can try to get the mutex in a while loop. When the mutex is available, a core takes the mutex and updates the shared variable and then sets the mutex to release it and allow access from the other core to the shared variable.

3.2.2 Software implementation

MCUXpresso SDK for MCX Nx4x contains an example for using the interrupt mechanism of the Inter-CPU Mailbox at <SDK_LOCATION>\boards\<board_name>\driver_examples\mailbox\mutex\. This example shows the scenario explained above, where the primary core writes the shared variable address to the secondary core mailbox that causes a mailbox interrupt on the secondary core side to process the shared variable. Then in a while loop, both cores try to get the mutex. When a core gets the mutex, it updates the shared variable by incrementing its value by 1 and releases the mutex.

Figure x shows the flow of the mailbox mutex example. For details on the interrupt mechanism of this example, refer to the previous example.

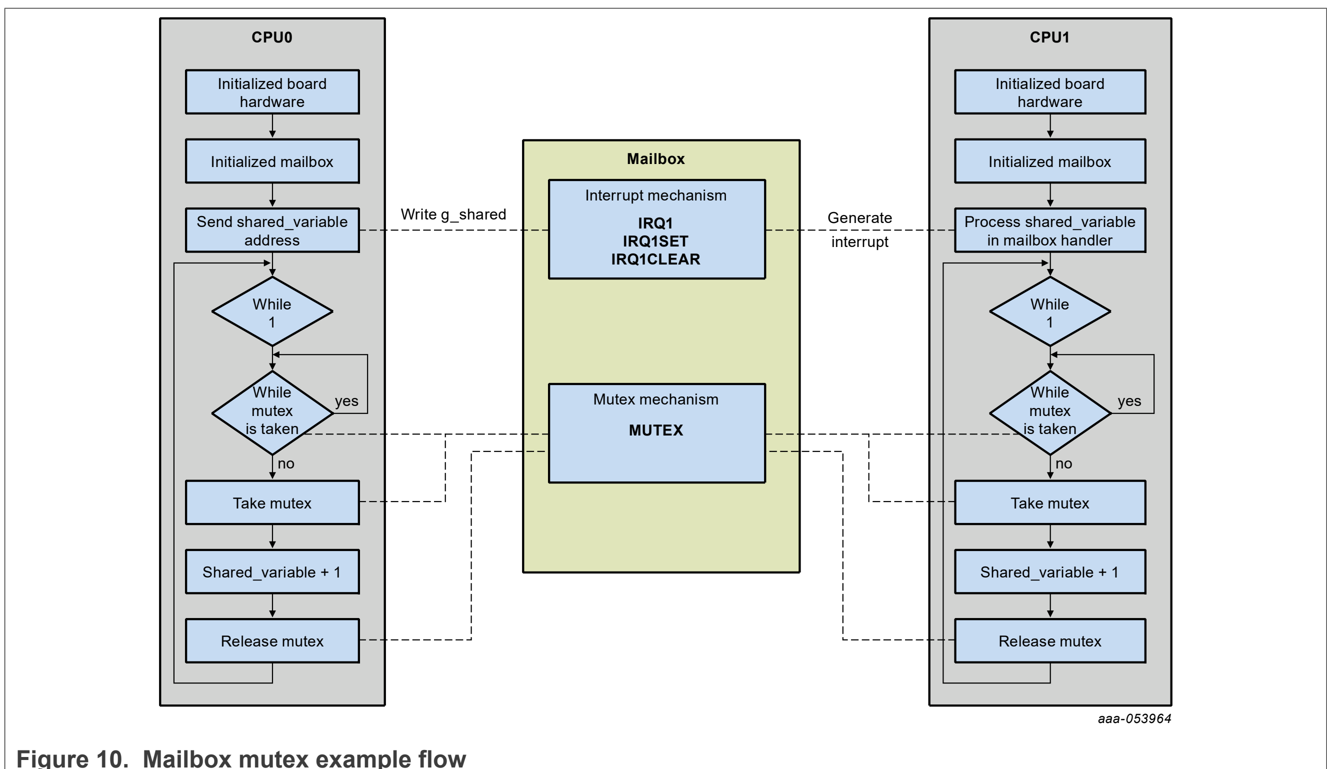

Figure 10. Mailbox mutex example flow

All information provided in this document is subject to legal disclaimers.

© 2024 NXP B.V. All rights reserved.

```
101 int main(void)
102 {
103     /* Init board hardware.*/
104     /* attach main clock divide to FLEXCOMM0 (debug console) */
105     CLOCK_AttachClk(BOARD_DEBUG_UART_CLK_ATTACH);
106 
107     BOARD_InitBootPins();
108     BOARD_InitBootClocks();
109     BOARD_InitDebugConsole();
110 
111     PRINTF("Mailbox mutex example\r\n");
112 
113     /* Init Mailbox */
114     MAILBOX_Init(MAILBOX);
115 
116     /* Enable mailbox interrupt */
117     NVIC_EnableIRQ(MAILBOX_IRQn);
118 
119 #ifdef CORE1_IMAGE_COPY_TO_RAM
120     /* Calculate size of the image */
121     uint32_t core1_image_size;
122     core1_image_size = get_core1_image_size();
123     PRINTF("Copy CORE1 image to address: 0x%x, size: %d\r\n", CORE1_BOOT_ADDRESS, core1_image_size);
124 
125     /* Copy application from FLASH to RAM */
126     memcpy((void *)CORE1_BOOT_ADDRESS, (void *)CORE1_IMAGE_START, core1_image_size);
127 #endif
128 
129     /* Start the secondary core */
130     start_secondary_core(CORE1_BOOT_ADDRESS);
131 
132     /* Wait for start and initialization of secondary core */
133     while (!g_secondary_core_started)
134         ;
135 
136     /* Send address of shared variable to the secondary core */
137     MAILBOX_SetValue(MAILBOX, SECONDARY_CORE_MAILBOX_CPU_ID, (uint32_t)&g_shared);
138 
139     while (1)
140     {
141         /* Get Mailbox mutex */
142         while (MAILBOX_GetMutex(MAILBOX) == 0)
143             ;
144 
145         /* The core0 has mutex, can change shared variable g_shared */
146         g_shared++;
147 
148         PRINTF("Core0 has mailbox mutex, update shared variable to: %d\r\n", g_shared);
149 
150         /* Set mutex to allow access other core to shared variable */
151         MAILBOX_SetMutex(MAILBOX);
152 
153         /* Add several nop instructions to allow the opposite core to get the mutex */
154         __asm("nop");
155         __asm("nop");
156         __asm("nop");
157         __asm("nop");
158         __asm("nop");
159         __asm("nop");
160         __asm("nop");
161         __asm("nop");
162         __asm("nop");
163         __asm("nop");
164     }
165 }
166 
```

Figure 11. Main function of CPU0

```
89 void MAILBOX_IRQHandler()
90 {
91     if (START_EVENT == MAILBOX_GetValue(MAILBOX, PRIMARY_CORE_MAILBOX_CPU_ID))
92     {
93         g_secondary_core_started = true;
94     }
95     MAILBOX_ClearValueBits(MAILBOX, PRIMARY_CORE_MAILBOX_CPU_ID, 0xffffffff);
96 }
```

Figure 12. Mailbox interrupt handler of CPU0

All information provided in this document is subject to legal disclaimers.

© 2024 NXP B.V. All rights reserved.

```
49 int main(void)
50 {
51     /* Init board hardware */
52     /* set BOD VBAT level to 1.65V */
53     BOARD_InitBootPins();
54     BOARD_InitDebugConsole();
55
56     /* Initialize Mailbox */
57     MAILBOX_Init(MAILBOX);
58
59     /* Enable mailbox interrupt */
60     NVIC_EnableIRQ(MAILBOX_IRQn);
61
62     /* Let the other side know the application is up and running */
63     MAILBOX_SetValue(MAILBOX, PRIMARY_CORE_MAILBOX_CPU_ID, (uint32_t)START_EVENT);
64
65     while (1)
66     {
67         /* Get Mailbox mutex */
68         while (MAILBOX_GetMutex(MAILBOX) == 0)
69             ;
70
71         /* The core1 has mutex, can change shared variable g_shared */
72         if (g_shared != NULL)
73         {
74             (*g_shared)++;
75             PRINTF("Core1 has mailbox mutex, update shared variable to: %d\r\n", *g_shared);
76         }
77
78         /* Set mutex to allow access other core to shared variable */
79         MAILBOX_SetMutex(MAILBOX);
80
81         /* Add several nop instructions to allow the opposite core to get the mutex */
82         __asm("nop");
83         __asm("nop");
84         __asm("nop");
85         __asm("nop");
86         __asm("nop");
87         __asm("nop");
88         __asm("nop");
89         __asm("nop");
90         __asm("nop");
91         __asm("nop");
92     }
93 }
94
```

Figure 13. Main function of CPU1

```
31 /* Pointer to shared variable by both cores, before changing of this variable the
32    cores must first take Mailbox mutex, after changing the shared variable must
33    retrun mutex */
34 volatile uint32_t *g_shared = NULL;
35
36 /*******************************************************************************
37  * Code
38  ******************************************************************************/
39 void MAILBOX_IRQHandler()
40 {
41     g_shared = (uint32_t *)MAILBOX_GetValue(MAILBOX, SECONDARY_CORE_MAILBOX_CPU_ID);
42     MAILBOX_ClearValueBits(MAILBOX, SECONDARY_CORE_MAILBOX_CPU_ID, 0xffffffff);
43     __DSB();
44 }
```

Figure 14. Mailbox interrupt handler of CPU1

© 2024 NXP B.V. All rights reserved.

```
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
```

Figure 15. Inter-CPU mailbox mutex example output on a serial terminal

4 Conclusion

This application note provides a quick look at the dual-core mechanisms in the MCX Nx4x device. It briefly introduces some applications related to dual-core communication based on the MAILBOX driver examples in the MCX Nx4x SDK and demonstrates how to use the interrupt and mutex mechanisms depending on the use case.

5 Note about the source code in the document

Example code shown in this document has the following copyright and BSD-3-Clause license:

Copyright 2024 NXP Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials must be provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

All information provided in this document is subject to legal disclaimers.

© 2024 NXP B.V. All rights reserved.

6 Revision history

Table 1. Revision history

Document ID	Release date	Description
AN14150 v.1.0	20 January 2024	Initial version

© 2024 NXP B.V. All rights reserved.

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/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.

© 2024 NXP B.V. All rights reserved.

AMBA, Arm, Arm7, Arm7TDMI, Arm9, Arm11, Artisan, big.LITTLE, Cordio, CoreLink, CoreSight, Cortex, DesignStart, DynamIQ, Jazelle, Keil, Mali, Mbed, Mbed Enabled, NEON, POP, RealView, SecurCore, Socrates, Thumb, TrustZone, ULINK, ULINK2, ULINK-ME, ULINK-PLUS, ULINKpro, μVision, Versatile — are trademarks and/or registered trademarks of Arm Limited (or its subsidiaries or affiliates) in the US and/or elsewhere. The related technology may be protected by any or all of patents, copyrights, designs and trade secrets. All rights reserved.

CoolFlux — is a trademark of NXP B.V.

MCX — is a trademark of NXP B.V.

Microsoft, Azure, and ThreadX — are trademarks of the Microsoft group of companies.

© 2024 NXP B.V. All rights reserved.

Contents

1	**Introduction**	**2**
2	**Dual-core basic mechanism**	**2**
3	**Inter-CPU Mailbox**	**3**
3.1	Interrupt mechanism	3
3.1.1	Example	3
3.1.2	Software implementation	4
3.2	Mutex mechanism	6
3.2.1	Example	7
3.2.2	Software implementation	7
4	**Conclusion**	**10**
5	**Note about the source code in the document**	**10**
6	**Revision history**	**11**
	Legal information	**12**

Please be aware that important notices concerning this document and the product(s) described herein, have been included in section 'Legal information'.

© 2024 NXP B.V. **All rights reserved.**

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

Date of release: 20 January 2024
Document identifier: AN14150