

Core Overview

Multiprocessor environments can use the mailbox core with Avalon® interface to send messages between processors.

The mailbox core contains mutexes to ensure that only one processor modifies the mailbox contents at a time. The mailbox core must be used in conjunction with a separate shared memory that is used for storing the actual messages.

The mailbox core is designed for use in Avalon-based processor systems, such as a Nios® II processor system. Altera provides device drivers for the Nios II processor. The mailbox core is SOPC Builder-ready and integrates easily into any SOPC Builder-generated system. This chapter contains the following sections:

- [“Functional Description”](#)
- [“Device Support” on page 27-2](#)
- [“Instantiating the Core in SOPC Builder” on page 27-2](#)
- [“Software Programming Model” on page 27-3](#)
- [“Mailbox API” on page 27-5](#)

Functional Description

The mailbox core has a simple Avalon Memory-Mapped (Avalon-MM) slave interface that provides access to four memory-mapped, 32-bit registers. [Table 27-1](#) shows the registers.

Table 27-1. Mutex Core Register Map

Offset	Register Name	R/W	Bit Description			
			31	16	15	1
0	mutex0	RW	OWNER	VALUE		
1	reset0	RW	Reserved			RESET
2	mutex1	RW	OWNER	VALUE		
3	reset1	RW	Reserved			RESET

The mailbox component contains two mutexes: One to ensure unique write access to shared memory and one to ensure unique read access from shared memory. The mailbox core is used in conjunction with a separate memory in the system that is shared among multiple processors.

Mailbox functionality using the mutexes and memory is implemented entirely in the software. Refer to [“Software Programming Model” on page 27-3](#) for details about how to use the mailbox core in software.



For a detailed description of the mutex hardware operation, refer to the [Mutex Core](#) chapter in volume 5 of the *Quartus II Handbook*.

Device Support

The mailbox core supports all Altera® device families.

Instantiating the Core in SOPC Builder

You can instantiate and configure the mailbox core in an SOPC Builder system using the following process:

1. Decide which processors share the mailbox.
2. On the SOPC Builder **System Contents** tab, instantiate a memory component to serve as the mailbox buffer. Any RAM can be used as the mailbox buffer. The mailbox buffer can share space in an existing memory, such as program memory; it does not require a dedicated memory.
3. On the SOPC Builder **System Contents** tab, instantiate the mailbox component. The mailbox MegaWizard™ Interface presents the following options:
 - **Memory module**—Specifies which memory to use for the mailbox buffer. If the **Memory module** list does not contain the desired shared memory, the memory is not connected in the system correctly. Refer to Step 4 on page 27-2.
 - **CPUs available with this memory**—Shows all the processors that can share the mailbox. This field is always read-only. Use it to verify that the processor connections are correct. If a processor that needs to share the mailbox is missing from the list, refer to Step 4 on page 27-2.
 - **Shared mailbox memory offset**—Specifies an offset into the memory. The mailbox message buffer starts at this offset.
 - **Mailbox size (bytes)**—Specifies the number of bytes to use for the mailbox message buffer. The Nios II driver software provided by Altera uses eight bytes of overhead to implement the mailbox functionality. For a mailbox capable of passing only one message at a time, **Mailbox size (bytes)** must be at least 12 bytes.
 - **Maximum available bytes**—Specifies the number of bytes in the selected memory available for use as the mailbox message buffer. This field is always read-only.
4. If not already connected, make component connections on the SOPC Builder **System Contents** tab.
 - a. Connect each processor's data bus master port to the mailbox slave port.
 - b. Connect each processor's data bus master port to the shared mailbox memory.

Software Programming Model

The following sections describe the software programming model for the mailbox core. For Nios II processor users, Altera provides routines to access the mailbox core hardware. These functions are specific to the mailbox core and directly manipulate low-level hardware.

The mailbox software programming model has the following characteristics and assumes there are multiple processors accessing a single mailbox core and a shared memory:

- Each mailbox message is one 32-bit word.
- There is a predefined address range in shared memory dedicated to storing messages. The size of this address range imposes a maximum limit on the number of messages pending.
- The mailbox software implements a message FIFO between processors. Only one processor can write to the mailbox at a time, and only one processor can read from the mailbox at a time, ensuring message integrity.
- The software on both the sending and receiving processors must agree on a protocol for interpreting mailbox messages. Typically, processors treat the message as a pointer to a structure in shared memory.
- The sending processor can post messages in succession, up to the limit imposed by the size of the message address range.
- When messages exist in the mailbox, the receiving processor can read messages. The receiving processor can block until a message appears, or it can poll the mailbox for new messages.
- Reading the message removes the message from the mailbox.

Software Files

Altera provides the following software files accompanying the mailbox core hardware:

- **altera_avalon_mailbox_regs.h**—Defines the core's register map, providing symbolic constants to access the low-level hardware.
- **altera_avalon_mailbox.h**—Defines data structures and functions to access the mailbox core hardware.
- **altera_avalon_mailbox.c**—Contains the implementations of the functions to access the mailbox core.

Programming with the Mailbox Core

This section describes the software constructs for manipulating the mailbox core hardware.

The file **altera_avalon_mailbox.h** declares a structure `alt_mailbox_dev` that represents an instance of a mailbox device. It also declares functions for accessing the mailbox hardware structure, listed in [Table 27-2](#). For a complete description of each function, refer to [“Mailbox API” on page 27-5](#).

Table 27-2. Mailbox API Functions

Function Name	Description
<code>altera_avalon_mailbox_close()</code>	Closes the handle to a mailbox.
<code>altera_avalon_mailbox_get()</code>	Returns a message if one is present, but does not block waiting for a message.
<code>altera_avalon_mailbox_open()</code>	Claims a handle to a mailbox, enabling all the other functions to access the mailbox core.
<code>altera_avalon_mailbox_pend()</code>	Blocks waiting for a message to be in the mailbox.
<code>altera_avalon_mailbox_post()</code>	Posts a message to the mailbox.

Example 27-1 demonstrates writing to and reading from a mailbox. For this example, assume that the hardware system has two processors communicating via mailboxes. The system includes two mailbox cores, which provide two-way communication between the processors.

Example 27-1. Writing to and Reading from a Mailbox

```
#include <stdio.h>
#include "altera_avalon_mailbox.h"

int main()
{
    alt_u32 message = 0;
    alt_mailbox_dev* send_dev, rcv_dev;
    /* Open the two mailboxes between this processor and another */
    send_dev = altera_avalon_mailbox_open("/dev/mailbox_0");
    rcv_dev = altera_avalon_mailbox_open("/dev/mailbox_1");

    while(1)
    {
        /* Send a message to the other processor */
        altera_avalon_mailbox_post(send_dev, message);

        /* Wait for the other processor to send a message back */
        message = altera_avalon_mailbox_pend(rcv_dev);
    }
    return 0;
}
```

Mailbox API

This section describes the application programming interface (API) for the mailbox core.

altera_avalon_mailbox_close()

Prototype: `void altera_avalon_mailbox_close (alt_mailbox_dev* dev);`
Thread-safe: Yes.
Available from ISR: No.
Include: `<altera_avalon_mailbox.h>`
Parameters: `dev`—the mailbox to close.
Returns: Null.
Description: `altera_avalon_mailbox_close()` closes the mailbox.

altera_avalon_mailbox_get()

Prototype: `alt_u32 altera_avalon_mailbox_get (alt_mailbox_dev* dev, int* err);`
Thread-safe: Yes.
Available from ISR: No.
Include: `<altera_avalon_mailbox.h>`
Parameters: `dev`—the mailbox handle.
`err`—pointer to an error code that is returned.
Returns: Returns a message if one is available in the mailbox, otherwise returns 0. The value pointed to by `err` is 0 if the message was read correctly, or `EWOULDBLOCK` if there is no message to read.
Description: `altera_avalon_mailbox_get()` returns a message if one is present, but does not block waiting for a message.

altera_avalon_mailbox_open()

Prototype: `alt_mailbox_dev* altera_avalon_mailbox_open (const char* name);`
Thread-safe: Yes.
Available from ISR: No.
Include: `<altera_avalon_mailbox.h>`
Parameters: `name`—the name of the mailbox device to open.
Returns: Returns a handle to the mailbox, or NULL if this mailbox does not exist.
Description: `altera_avalon_mailbox_open()` opens a mailbox.

altera_avalon_mailbox_pend()

Prototype: `alt_u32 altera_avalon_mailbox_pend (alt_mailbox_dev* dev);`
Thread-safe: Yes.
Available from ISR: No.
Include: `<altera_avalon_mailbox.h>`
Parameters: `dev`—the mailbox device to read a message from.
Returns: Returns the message.
Description: `altera_avalon_mailbox_pend()` is a blocking routine that waits for a message to appear in the mailbox and then reads it.

altera_avalon_mailbox_post()

Prototype: `int altera_avalon_mailbox_post (alt_mailbox_dev* dev, alt_u32 msg);`
Thread-safe: Yes.
Available from ISR: No.
Include: `<altera_avalon_mailbox.h>`
Parameters: `dev`—the mailbox device to post a message to.
`msg`—the value to post.
Returns: Returns 0 on success, or `EWOULDBLOCK` if the mailbox is full.
Description: `altera_avalon_mailbox_post()` posts a message to the mailbox.

Document Revision History

Table 27-3 shows the revision history for this chapter.

Table 27-3. Document Revision History

Date and Document Version	Changes Made	Summary of Changes
March 2009 v9.0.0	No change from previous release.	—
November 2008 v8.1.0	Changed to 8-1/2 x 11 page size. No change to content.	—
May 2008 v8.0.0	No change from previous release.	—



For previous versions of the *Quartus II Handbook*, refer to the [Quartus II Handbook Archive](#).