

2016.12.19

UG-01085



Subscribe



Send Feedback

## Core Overview

Multiprocessor environments can use the mutex core with Avalon<sup>®</sup> interface to coordinate accesses to a shared resource. The mutex core provides a protocol to ensure mutually exclusive ownership of a shared resource.

The mutex core provides a hardware-based atomic test-and-set operation, allowing software in a multiprocessor environment to determine which processor owns the mutex. The mutex core can be used in conjunction with shared memory to implement additional interprocessor coordination features, such as mailboxes and software mutexes.

The mutex core is designed for use in Avalon-based processor systems, such as a Nios<sup>®</sup> II processor system. Altera provides device drivers for the Nios II processor to enable use of the hardware mutex.

## Functional Description

The mutex core has a simple Avalon Memory-Mapped (Avalon-MM) slave interface that provides access to two memory-mapped, 32-bit registers.

**Table 29-1: Mutex Core Register Map**

Offset	Register Name	R/W	Bit Description		
			31 16	15 1	0
0	mutex	RW	OWNER	VALUE	
1	reset	RW	Reserved		RESET

The mutex core has the following basic behavior. This description assumes there are multiple processors accessing a single mutex core, and each processor has a unique identifier (ID).

© 2016 Intel Corporation. All rights reserved. Intel, the Intel logo, Altera, Arria, Cyclone, Enpirion, MAX, NIOS, Quartus and Stratix words and logos are trademarks of Intel Corporation in the US and/or other countries. Other marks and brands may be claimed as the property of others. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

ISO  
9001:2008  
Registered

**ALTERA**  
now part of Intel

- When the `VALUE` field is `0x0000`, the mutex is unlocked and available. Otherwise, the mutex is locked and unavailable.
- The mutex register is always readable. Avalon-MM master peripherals, such as a processor, can read the mutex register to determine its current state.
- The mutex register is writable only under specific conditions. A write operation changes the mutex register only if one or both of the following conditions are true:
  - The `VALUE` field of the mutex register is zero.
  - The `OWNER` field of the mutex register matches the `OWNER` field in the data to be written.
- A processor attempts to acquire the mutex by writing its ID to the `OWNER` field, and writing a non-zero value to the `VALUE` field. The processor then checks if the acquisition succeeded by verifying the `OWNER` field.
- After system reset, the `RESET` bit in the `reset` register is high. Writing a one to this bit clears it.

## Configuration

The MegaWizard™ Interface provides the following options:

- **Initial Value**—the initial contents of the `VALUE` field after reset. If the **Initial Value** setting is non-zero, you must also specify **Initial Owner**.
- **Initial Owner**—the initial contents of the `OWNER` field after reset. When **Initial Owner** is specified, this owner must release the mutex before it can be acquired by another owner.

## Software Programming Model

The following sections describe the software programming model for the mutex core. For Nios II processor users, Altera provides routines to access the mutex core hardware. These functions are specific to the mutex core and directly manipulate low-level hardware. The mutex core cannot be accessed via the HAL API or the ANSI C standard library. In Nios II processor systems, a processor locks the mutex by writing the value of its `cpuid` control register to the `OWNER` field of the mutex register.

## Software Files

Altera provides the following software files accompanying the mutex core:

- `altera_avalon_mutex_regs.h`—Defines the core's register map, providing symbolic constants to access the low-level hardware.
- `altera_avalon_mutex.h`—Defines data structures and functions to access the mutex core hardware.
- `altera_avalon_mutex.c`—Contains the implementations of the functions to access the mutex core

## Hardware Access Routines

This section describes the low-level software constructs for manipulating the mutex core. The file `altera_avalon_mutex.h` declares a structure `alt_mutex_dev` that represents an instance of a mutex device. It also declares routines for accessing the mutex hardware structure, listed in the table below.

Table 29-2: Hardware Access Routines

Function Name	Description
<code>altera_avalon_mutex_open()</code>	Claims a handle to a mutex, enabling all the other functions to access the mutex core.
<code>altera_avalon_mutex_trylock()</code>	Tries to lock the mutex. Returns immediately if it fails to lock the mutex.
<code>altera_avalon_mutex_lock()</code>	Locks the mutex. Will not return until it has successfully claimed the mutex.
<code>altera_avalon_mutex_unlock()</code>	Unlocks the mutex.
<code>altera_avalon_mutex_is_mine()</code>	Determines if this CPU owns the mutex.
<code>altera_avalon_mutex_first_lock()</code>	Tests whether the mutex has been released since reset.

These routines coordinate access to the software mutex structure using a hardware mutex core. For a complete description of each function, see section the **Mutex API** section.

The code shown in below demonstrates opening a mutex device handle and locking a mutex.

```
#include <altera_avalon_mutex.h>

/* get the mutex device handle */
alt_mutex_dev* mutex = altera_avalon_mutex_open( "/dev/mutex" );

/* acquire the mutex, setting the value to one */
altera_avalon_mutex_lock( mutex, 1 );

/*
 * Access a shared resource here.
 */

/* release the lock */
altera_avalon_mutex_unlock( mutex );
```

## Mutex API

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

### `altera_avalon_mutex_is_mine()`

<b>Prototype:</b>	<code>int altera_avalon_mutex_is_mine(alt_mutex_dev* dev)</code>
<b>Thread-safe:</b>	Yes.
<b>Available from ISR:</b>	No.
<b>Include:</b>	<code>&lt;altera_avalon_mutex.h&gt;</code>
<b>Parameters:</b>	<code>dev</code> —the mutex device to test.
<b>Returns:</b>	Returns non zero if the mutex is owned by this CPU.

<b>Description:</b>	<code>altera_avalon_mutex_is_mine()</code> determines if this CPU owns the mutex.
---------------------	---

## altera\_avalon\_mutex\_first\_lock()

<b>Prototype:</b>	<code>int altera_avalon_mutex_first_lock(alt_mutex_dev* dev)</code>
<b>Thread-safe:</b>	Yes.
<b>Available from ISR:</b>	No.
<b>Include:</b>	<code>&lt;altera_avalon_mutex.h&gt;</code>
<b>Parameters:</b>	<code>dev</code> —the mutex device to test.
<b>Returns:</b>	Returns 1 if this mutex has not been released since reset, otherwise returns 0.
<b>Description:</b>	<code>altera_avalon_mutex_first_lock()</code> determines whether this mutex has been released since reset.

## altera\_avalon\_mutex\_lock()

<b>Prototype:</b>	<code>void altera_avalon_mutex_lock(alt_mutex_dev* dev, alt_u32 value)</code>
<b>Thread-safe:</b>	Yes.
<b>Available from ISR:</b>	No.
<b>Include:</b>	<code>&lt;altera_avalon_mutex.h&gt;</code>
<b>Parameters:</b>	<code>dev</code> —the mutex device to acquire. <code>value</code> —the new value to write to the mutex.
<b>Returns:</b>	—
<b>Description:</b>	<code>altera_avalon_mutex_lock()</code> is a blocking routine that acquires a hardware mutex, and at the same time, loads the mutex with the <code>value</code> parameter.

## altera\_avalon\_mutex\_open()

<b>Prototype:</b>	<code>alt_mutex_dev* alt_hardware_mutex_open(const char* name)</code>
<b>Thread-safe:</b>	Yes.
<b>Available from ISR:</b>	No.
<b>Include:</b>	<code>&lt;altera_avalon_mutex.h&gt;</code>
<b>Parameters:</b>	<code>name</code> —the name of the mutex device to open.

<b>Returns:</b>	A pointer to the mutex device structure associated with the supplied name, or NULL if no corresponding mutex device structure was found.
<b>Description:</b>	<code>altera_avalon_mutex_open()</code> retrieves a pointer to a hardware mutex device structure.

## altera\_avalon\_mutex\_trylock()

<b>Prototype:</b>	<code>int altera_avalon_mutex_trylock(alt_mutex_dev* dev, alt_u32 value)</code>
<b>Thread-safe:</b>	Yes.
<b>Available from ISR:</b>	No.
<b>Include:</b>	<code>&lt;altera_avalon_mutex.h&gt;</code>
<b>Parameters:</b>	<code>dev</code> —the mutex device to lock. <code>value</code> —the new value to write to the mutex.
<b>Returns:</b>	0 = The mutex was successfully locked. Others = The mutex was not locked.
<b>Description:</b>	<code>altera_avalon_mutex_trylock()</code> tries once to lock the hardware mutex, and returns immediately.

## altera\_avalon\_mutex\_unlock()

<b>Prototype:</b>	<code>void altera_avalon_mutex_unlock(alt_mutex_dev* dev)</code>
<b>Thread-safe:</b>	Yes.
<b>Available from ISR:</b>	No.
<b>Include:</b>	<code>&lt;altera_avalon_mutex.h&gt;</code>
<b>Parameters:</b>	<code>dev</code> —the mutex device to unlock.
<b>Returns:</b>	Null.
<b>Description:</b>	<code>altera_avalon_mutex_unlock()</code> releases a hardware mutex device. Upon release, the value stored in the mutex is set to zero. If the caller does not hold the mutex, the behavior of this function is undefined.

## Document Revision History

**Table 29-3: Mutex Core Revision History**

Date	Version	Changes
July 2014	2014.07.24	Removed mention of SOPC Builder, updated to Qsys
December 2010	v10.1.0	Removed the “Device Support”, “Instantiating the Core in SOPC Builder”, and “Referenced Documents” sections.
July 2010	v10.0.0	No change from previous release.
November 2009	v9.1.0	No change from previous release.
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.