

ARMv8-A Synchronization primitives

Version 1.0

Revision Information

The following revisions have been made to this User Guide.

Date	Issue	Confidentiality	Change
03 March 2017	0100	Non-Confidential	First release

Proprietary Notice

Words and logos marked with ® or ™ are registered trademarks or trademarks of ARM® in the EU and other countries, except as otherwise stated below in this proprietary notice. Other brands and names mentioned herein may be the trademarks of their respective owners.

Neither the whole nor any part of the information contained in, or the product described in, this document may be adapted or reproduced in any material form except with the prior written permission of the copyright holder.

The product described in this document is subject to continuous developments and improvements. All particulars of the product and its use contained in this document are given by ARM in good faith. However, all warranties implied or expressed, including but not limited to implied warranties of merchantability, or fitness for purpose, are excluded.

This document is intended only to assist the reader in the use of the product. ARM shall not be liable for any loss or damage arising from the use of any information in this document, or any error or omission in such information, or any incorrect use of the product.

Where the term ARM is used it means “ARM or any of its subsidiaries as appropriate”.

Confidentiality Status

This document is Confidential. The right to use, copy and disclose this document may be subject to license restrictions in accordance with the terms of the agreement entered into by ARM and the party that ARM delivered this document to.

Unrestricted Access is an ARM internal classification.

Product Status

The information in this document is final, that is for a developed product.

Web Address

<http://www.arm.com>

Contents

1	Synchronization primitives	4
2	A single lock implementation	5
3	Exclusive access instructions	6
4	Simple exclusive lock	7
5	Exclusive Monitors	8
5.1	Local Monitor	8
5.2	Global Monitor	8
6	Power efficient locks	9
6.1	WFE and WFI	9

I Synchronization primitives

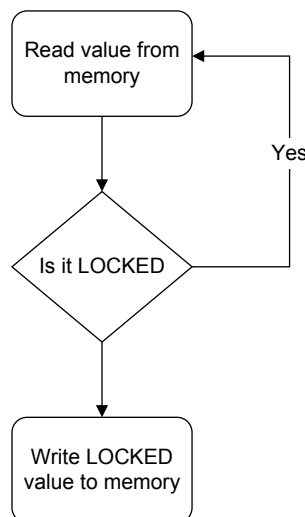
In systems with more than one thread of execution certain resources exist that must not be accessed or modified at the same time. These resources could be peripheral devices or memory buffers and data structures that are accessed by multiple threads or cores.

This type of concurrency control is a general computer science problem. There are many different algorithms that can be used to achieve the necessary mutual exclusion. Many of these algorithms can be implemented entirely in software although more efficient implementations are available if special hardware features are supported. Evaluating the various algorithms is outside the scope of this article which will, instead, focus on the synchronization primitives provided by the ARM architecture.

2 A single lock implementation

It is possible to protect a shared resource by implementing a lock function which allows a thread request ownership of the resource. Software can then enter into a contract to only access the shared resource after claiming the lock. The simple lock function can be extended to support multiple locks, one for each shared resource. An unlock function is necessary to allow software to release the lock.

A simple lock implementation could be implemented using a variable in memory that can contain one of two values, LOCKED and, UNLOCKED. The process would be to read the value from memory, if the lock is UNLOCKED then the value can be updated to LOCKED and written back.



This gives three separate steps:

- A memory read and compare.
- Modify a value in a register.
- A memory write.

In a system with multiple cores or threads this method is vulnerable to another thread modifying the value in memory in between the first read and the write-back of the new value.

This problem can be solved in software using something like a ticket lock or can be solved using extra hardware features. One solution could be an instruction that makes the read-compare-modify-write operation atomic. Earlier versions of the ARM architecture had a similar functionality available using the SWP instruction.

ARMv8-A has a different solution, using a special type of load and store instruction to detect if the value in memory has changed since it was last read.

3 Exclusive access instructions

The A64 instruction set has instructions for generating an exclusive access to a memory location:

- Load Exclusive (LDXR)
- Store Exclusive (STXR)

Each of these instructions has variants to allow different access sizes (doubleword, word, halfword and, byte).

In AArch32 state, A32 and T32 have instructions with a similar behavior:

- Load Exclusive (LDREX)
- Store Exclusive (STREX)

Whenever an address is read using a Load Exclusive instruction, it is marked as being for an exclusive access. If an address marked as exclusive is written to using a Store Exclusive instruction, it clears the exclusive status. An attempt to write to an address not marked as exclusive using a Store Exclusive instruction will not succeed. This enables software to detect if the contents of that address have been changed since the last time it was read.

The exclusive state of an address is maintained by a piece of hardware known as an Exclusive Monitor.

The syntax of the exclusive load is similar to a regular load:

```
LDXR <Wt/Xt>, [Xn]
```

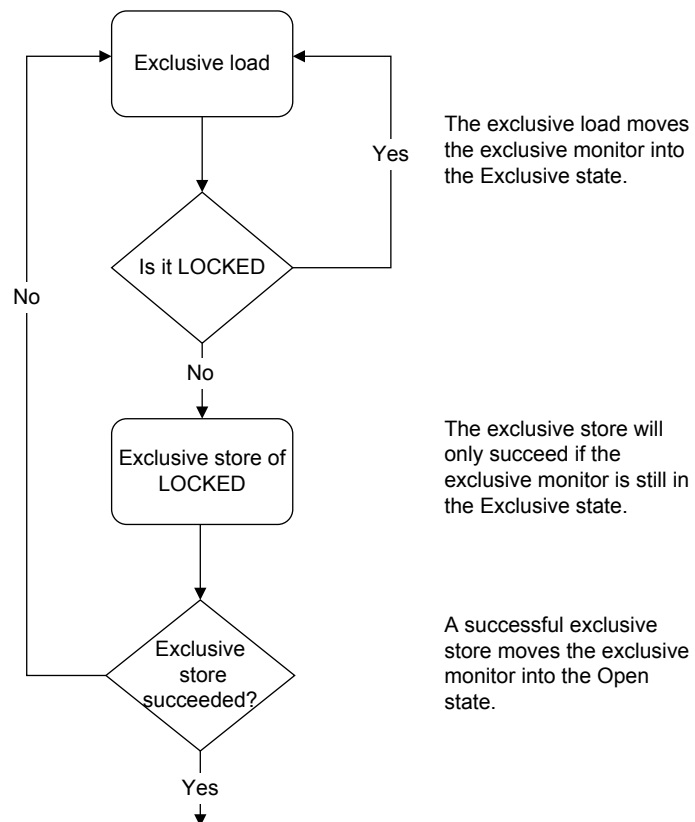
The exclusive store instructions return an extra parameter that indicates if the store could complete successfully based on the exclusive monitor status.

```
STXR Ws, <Wt/Xt>, [Xn]
```

If the store succeeds memory is updated and Ws returns zero. If the exclusive store cannot proceed Ws is nonzero and memory is not updated.

4 Simple exclusive lock

The simple lock example can now be expanded using the exclusive loads and stores.



The update of the lock value is not guaranteed to be atomic but any changes between the initial read and the update can now be detected. If the exclusive store fails, software can then attempt to gain the lock again.

5 Exclusive Monitors

An exclusive monitor is a simple state machine, with the possible states open and exclusive. The ARM architecture defines two different state machines: a Local Monitor and a Global Monitor.

- Exclusive accesses are checked against one or both of these monitors, based on the shareability attribute of the address being accessed.
- Exclusive accesses to a Non-shareable address are only checked against the local monitor belonging to that PR.
- Exclusive accesses to a shareable address are checked against the state of the local monitor and a global monitor. The global monitor is shared with other masters in the system which allows a single mutex address to be shared between multiple masters. The accesses are checked against both monitors and if the check fails on any monitor the exclusive store fails.

5.1 Local Monitor

Each core has a local monitor that is associated with it. The local monitor can be constructed to hold the exclusive state for a particular address or be constructed in such a way that it does not hold the address. In this case, any exclusive store is treated as if it matches the address of the previous exclusive load. The local monitor is implemented as part of the processor.

5.2 Global Monitor

Global monitors are shared between multiple cores. Like the local monitor, they are only required to monitor a single address although it is much more common for them to be able to maintain state for multiple addresses. If the monitor can mark multiple addresses each address has its own state machine.

It is an architectural requirement that the following memory types are able to function with a global monitor:

- Inner Shareable, Inner Write-Back, Outer Write-Back Normal memory with Read allocation hints and Write allocation hints and not transient.
- Outer shareable, Inner Write-Back, Outer Write-Back Normal memory with Read allocation hints and Write allocation hints and not transient.

In many implementations, the global monitor for these memory types is implemented as part of the cache coherency logic.

Other memory types are not guaranteed to be covered by a global monitor although it is also possible to implement a global monitor as part of the system-on-chip level memory system. If this type of monitor is present, it is often built into the RAM controller of memory interconnect.

If exclusive accesses to shared memory are not covered by a global monitor the exclusive stores will not function correctly. It is IMPLEMENTATION DEFINED if an abort is generated or the store simply fails.

6 Power efficient locks

The example exclusive lock enters a busy loop if it cannot immediately gain the lock. This can be inefficient and consumes unnecessary power. Many implementations pause before retrying to claim the lock if the initial attempt fails. This can be done in several ways. For locks where there will be a relatively long wait before the lock is released the lock code can yield back to the operating system scheduler. This allows other threads to be scheduled while until the lock is released.

In situations where the lock is expected to be released quickly the ARM architecture has a mechanism that allows execution to be temporarily suspended. This mechanism effectively puts the processor to sleep until a wake-up event is received.

6.1 WFE and WFI

The architecture has two mechanisms for putting the processor into a shallow sleep.

WFE is often used inside lock implementations as it is an efficient way of temporarily suspending the core while it waits for a lock to be released. In legacy code, it is common to see a Send Event (SEV) instruction in unlock code which serves to wake up any core which is waiting for that lock. In the ARMv8-A architecture clearing the global monitor automatically sends a wake-up event to all connected cores.