Non-Confidential Proprietary Notice

This document is protected by copyright and other related rights and the practice or implementation of the information contained in this document may be protected by one or more patents or pending patent applications. No part of this document may be reproduced in any form by any means without the express prior written permission of ARM Limited (“ARM”). No license, express or implied, by estoppel or otherwise to any intellectual property rights is granted by this document unless specifically stated.

Your access to the information in this document is conditional upon your acceptance that you will not use or permit others to use the information for the purposes of determining whether implementations infringe any patents.

THIS DOCUMENT IS PROVIDED “AS IS”. ARM PROVIDES NO REPRESENTATIONS AND NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY, SATISFACTORY QUALITY, NON-INFRINGEMENT OR FITNESS FOR A PARTICULAR PURPOSE WITH RESPECT TO THE DOCUMENT. For the avoidance of doubt, ARM makes no representation with respect to, and has undertaken no analysis to identify or understand the scope and content of, third party patents, copyrights, trade secrets, or other rights.

This document may include technical inaccuracies or typographical errors.

This document may be translated into other languages for convenience, and you agree that if there is any conflict between the English version of this document and any translation, the terms of the English version shall prevail.

TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL ARM BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF ARM HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

This document consists solely of commercial items. You shall be responsible for ensuring that any use, duplication or disclosure of this document complies fully with any relevant export laws and regulations to assure that this document or any portion thereof is not exported, directly or indirectly, in violation of such export laws. Use of the word “partner” in reference to ARM’s customers is not intended to create or refer to any partnership relationship with any other company. ARM may make changes to this document at any time and without notice.

If any of the provisions contained in these terms conflict with any of the provisions of any signed written agreement specifically covering this document with ARM, then the signed written agreement prevails over and supersedes the conflicting provisions of these terms.

Words and logos marked with ® or ™ are registered trademarks or trademarks of ARM Limited or its affiliates in the EU and/or elsewhere. All rights reserved. Other brands and names mentioned in this document may be the trademarks of their respective owners. You must follow the ARM trademark usage guidelines http://www.arm.com/about/trademarks/guidelines/index.php.

Copyright © ARM Limited or its affiliates. All rights reserved.
110 Fulbourn Road, Cambridge, England CB1 9NJ.

In this document, where the term ARM is used to refer to the company it means “ARM or any of its subsidiaries as appropriate”.
Confidentiality Status

This document is Non-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.

Product Status

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

Web Address

http://www.arm.com
# Table of Contents

1. **Aggregator** .......................................................................................................................... 8  
   1.1 Description ....................................................................................................................... 8  
   1.2 Ports .................................................................................................................................. 8  
   1.3 Parameters ....................................................................................................................... 8  

2. **Deaggregator** ....................................................................................................................... 9  
   2.1 Description ....................................................................................................................... 9  
   2.2 Ports .................................................................................................................................. 9  
   2.3 Parameters ....................................................................................................................... 9  

3. **MxMem** .................................................................................................................................. 10  
   3.1 Description ....................................................................................................................... 10  
   3.2 Variants .......................................................................................................................... 10  
   3.3 Ports .................................................................................................................................. 10  
   3.4 Port Access Interface ....................................................................................................... 11  
   3.4.1 Read/Write .................................................................................................................... 11  
   3.5 Parameters ....................................................................................................................... 12  
   3.6 Debug Support ................................................................................................................ 13  
   3.7 Examples ........................................................................................................................ 13  
   3.8 FAQ .................................................................................................................................. 13  

4. **MxBus** .................................................................................................................................. 14  
   4.1 Description ....................................................................................................................... 14  
   4.2 Variants .......................................................................................................................... 14  
   4.3 Ports .................................................................................................................................. 14  
   4.4 Port Access Interface ....................................................................................................... 14
# MxTimer

## Description

## Variants

## Ports

## Parameters

## Debug Support

## Example

# MemoryMappedCounterModule

## Description

## Ports

## Parameters

# MxSigDriver

## Description

## Ports

## Parameters

# intVector

# MxStub

## Description

## Ports

## Parameters

## Memory Spaces

## Debug Support

## Example

# Semihost and Semihost2
12.1 Description ............................................................................................................. 37
12.2 Ports.......................................................................................................................... 37
12.3 Parameters ............................................................................................................... 38

13 Miscellaneous Components ......................................................................................... 39
13.1 FOUT ....................................................................................................................... 39
13.2 BDEC ....................................................................................................................... 39
  13.2.1 Parameters ......................................................................................................... 39
13.3 CDIV ......................................................................................................................... 40
  13.3.1 Ports .................................................................................................................... 40
  13.3.2 Parameters ......................................................................................................... 40
1 Aggregator

1.1 Description
The Aggregator component takes multiple input signals and aggregates them into a single output signal. The aggregator has a maximum of 32 input ports; each input port takes in up to 32 bits and drives the appropriate bits through the output port. The output port drives 128 bits to all the connected slave signals, which are responsible for extracting the appropriate number of bits.

After instantiation on the SoC Designer Canvas:

1. Configure the number of input ports by disabling and hiding unused ports.
2. Configure the width of the input ports using the 'inputWidth' parameter.

Note: Ensure that the number of input ports times the input port width (inputWidth) is not greater than 128.

1.2 Ports

<table>
<thead>
<tr>
<th>Port</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in[00-31]</td>
<td>Signal slave ports.</td>
</tr>
<tr>
<td>output</td>
<td>128 bit signal master.</td>
</tr>
</tbody>
</table>

Aggregator ports

1.3 Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inputWidth</td>
<td>Specifies the width of the input ports up to 32 bits. The default width is 4 bits.</td>
</tr>
</tbody>
</table>

Aggregator parameters
2 Deaggregator

2.1 Description
The Deaggregator component takes in one input signal (up to 128 bits), splits it into multiple output signals, and drives the appropriate output ports. The Deaggregator has a maximum of 32 output ports.

After instantiation on the SoC Designer Canvas, configure the number of output ports by disabling and hiding unused ports.

**Note:** Ensure that the number of output ports times the output port width (outputWidth) is equal to the width of the input port (inputWidth).

2.2 Ports

<table>
<thead>
<tr>
<th>Port</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>input</td>
<td>Signal Slave port.</td>
</tr>
<tr>
<td>out[00-31]</td>
<td>Signal master ports.</td>
</tr>
</tbody>
</table>

Deaggregator ports

2.3 Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inputWidth</td>
<td>Specifies the width (up to 128 bits) of the input port. 128 bits is the default).</td>
</tr>
<tr>
<td>outputWidth</td>
<td>Specifies the width (up to 32 bits) of the output ports. 32 bits is the default).</td>
</tr>
</tbody>
</table>

Deaggregator parameters
3  MxMem

3.1 Description
Multi-ported memory component with parameter-selectable byte or half-word (16-bit) addresses. Support for data accesses up to 64-bit is provided.

The memory size can be adjusted to up to 4GByte. If the size is greater than 64KByte then the memory is organized as 64KByte segments which are allocated in the host memory only when used for the first time in order to limit resource usage.

The memory component supports wait states (Parameter “delay”). Calls to read() or write() have to be repeated as long as CASI_STATUS_WAIT is returned. Depending on the success of the operation the component returns CASI_STATUS_OK or CASI_STATUS_ERROR. It is important to keep in mind that ctrl[0] will be overwritten with the return value after every call to read() or write(). So it has to be reinitialized every time if CASI_STATUS_WAIT has been returned.

The variants with more than one port can only handle one read and one write in total per cycle, otherwise the results are unpredictable.

3.2 Variants
MxMem[1-4] component variants provide 1, 2, 3, or 4 transaction slave ports.

3.3 Ports

<table>
<thead>
<tr>
<th>Port</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>clk-in</td>
<td>This component must be connected to the clock.</td>
</tr>
<tr>
<td>Port[0-3]</td>
<td>Each port is of equal priority. Concurrent write accesses are not detected and can result in unpredictable behavior.</td>
</tr>
</tbody>
</table>

MxMem ports
### 3.4 Port Access Interface

#### 3.4.1 Read/Write

<table>
<thead>
<tr>
<th>Port</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>port[0-3]</td>
<td>Address: <code>uint64_t</code> memory address</td>
</tr>
<tr>
<td></td>
<td>Data: <code>uint32_t</code> array memory data</td>
</tr>
<tr>
<td></td>
<td>Ctrl: <code>uint32_t</code> array data access qualifier. The master should set this to one of the following:</td>
</tr>
<tr>
<td></td>
<td>=1 for 8-bit access</td>
</tr>
<tr>
<td></td>
<td>=2 for 16-bit access</td>
</tr>
<tr>
<td></td>
<td>=3 for 32-bit access</td>
</tr>
<tr>
<td></td>
<td>=4 for 64-bit access</td>
</tr>
</tbody>
</table>

The memory component supports read-accesses from any byte address, regardless of the data size. Write accesses for 8-32 bit data can be made to any byte address. When the data is 64-bit, then only 32-bit aligned addresses are valid, both in debug and non-debug mode.

The return value of read() and write() is of type `eslapi::CASIStatus` enum.
### 3.5 Parameters

The component can be configured to adjust the memory map and size (up to 4 GByte), and the latency of the memory. Furthermore it can be configured to support byte as well as half word (16-bit) addresses.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| 32bit aligned [1-4]              | **true** = Accesses are aligned to 32bit addresses for the specified port. The lower 2 bits of the byte address are ignored, the parameter “byte addressable” must be true.  
**false** = Accesses can use byte addresses without any particular alignment |
| base port [1-4]                  | Base address for memory map for each port                                                        |
| big endian                       | **true** = byte order is big endian                                                               
**false** = byte order is little endian                                                      |
| byte addressable [1-4]           | Access mode for each port:  
**true** = addresses refer to byte addresses  
**false** = addresses refer to halfword addresses                                            |
| delay                            | Number of read/write delay cycles (max=10)                                                      |
| enable debug messages            | **0** = no debug messages  
**1** = report read/write accesses  
**2** = report readDbg/writeDbg accesses                                                      |
| init value                       | Value assigned during initialization                                                            |
| read only                        | If true, the memory is not accepting writes, except via the debug interfaces                   |
| size                             | Size of memory (max. 4 GByte)                                                                   |
| use input file                   | When this parameter is set to true the MxMem model can be                                       |
initialized with a binary file containing a memory image. The binary file consists of values only. The addresses are assumed to start at the base addresses (port 1) and are incremented with each value.

\[ \text{true} = \text{prompt for input file on hard-reset} \]

\[ \text{false} = \text{no input file} \]

### MxMem parameters

#### 3.6 Debug Support

This component provides an CADI interface for memory display. The contents of the memory can be viewed via the debug menu of the SoC Designer Simulator.

#### 3.7 Examples

See the MxBus example in the next chapter.

#### 3.8 FAQ

**Q:** How can I use this memory component if the addresses of my component refer to word (32-bit) addresses?

**A:** This memory component only supports byte or half-word accesses. Therefore, the 32-bit word address should be multiplied by 4 to obtain a byte address. This is best done by left-shifting the address by 2 bit.
4 MxBus

4.1 Description
This is a bus component implementing basic arbitration.

4.2 Variants
MxBus[1-4] component variants provide 1, 2, 3 or 4 transaction slave ports.

4.3 Ports

<table>
<thead>
<tr>
<th>Port</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>clk-in</td>
<td>Clock</td>
</tr>
<tr>
<td>bmaster</td>
<td>This port needs to be connected to transaction slave ports of memory components or memory mapped peripherals</td>
</tr>
<tr>
<td>bslave[0-3]</td>
<td>This port needs to be connected to transaction master ports of any component that can access memory and supports arbitration</td>
</tr>
</tbody>
</table>

MxBus ports

4.4 Port Access Interface

4.4.1 Read/Write
The read and write functions simply forward the transaction to the connected slave component, which is mapped to the requested address. If no component is mapped to that address then CASI_STATUS_NOMEMORY is returned.

<table>
<thead>
<tr>
<th>Port</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bslave[0-3]</td>
<td>Address, data and ctrl from bslave[0-3] are forwarded onto the port connected to the bmaster port</td>
</tr>
</tbody>
</table>

MxBus port interfaces
### 4.4.2 Arbitration

The MxBus components support simple arbitration using the requestAccess() and checkForGrant() functions of the transaction interface. To request the bus, a component must call requestAccess. This function calls internally checkForGrant(), so access to the bus can be obtained without an obligatory delay cycle. If the bus is not granted immediately the component can query the result of the request in the following cycle by calling checkForGrant(). If the latter returns CASI_GRANT_OK then it has permission to access the bus for read/write. The bus snoops the slave’s response. If it is CASI_STATUS_WAIT the bus stays granted for this master until CASI_STATUS_OK or CASI_STATUS_ERROR is returned.

The priority of the ports is descending, i.e. port 0 has highest priority, port 3 has the lowest. The bus also supports concurrent read and write accesses. However grant is given to single port at a time based on priority.

### 4.5 Parameters

The component can be configured to adjust the covered address space and the minimum block size for each connected component. Also, the arbitration can be disabled if necessary.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| address space     | This parameter defines the address space covered by the bus decoder. Possible values are:  
8 bit, 16 bit, 24 bit, 32 bit, 36 bit, 64 bit |
| base address      | This address is reported as the start address in getMemoryRegions, required for cascaded bus structures. |
| block size        | Minimum size of a memory block in the memory map. Please be aware that a small block size can result in a very large decoder array if the address space is large. |
| enable arbitration| This parameter allows to turn the arbitration on and off                      |
| enable debug messages | 0 = no debug messages   
1 = report read/write accesses   
2 = report readDbg/writeDbg accesses |
### MxBus parameters

**Warning:** The decode array allocated by this component is equivalent to the address space size divided by the block size. For example, if address space is 32bit and the block size is 64k, then an array with 65536 elements will be allocated. If the block size were reduced to one in this case then the host will most likely fail to allocate the necessary memory ($2^{32}$ entries).

### 4.6 Debug Support

This component does not provide any debug interface.

### 4.7 Examples

In order to connect multiple memories to the bus component it is necessary to configure the memories so that they occupy non-overlapping memory regions in the memory map.

In the above example, the system consists of a single core, connected to the MxBus component, which is connected to two memory components on the other side. In order to configure this system for the memory map mention above, the following parameters need to be set:

- **BUS :: block_size** 0x400
- **RAM :: base** 0
- **RAM :: size** 0x400
- **ROM :: base** 0x400
- **ROM :: size** 0x3C00
The block_size parameter cannot be larger than the smallest memory block in the system, which is 0x400. However it can be set to smaller values.

## 4.8 FAQ

**Q:** Why do I get the following message in SoC Designer when running my simulation?

```
WARNING :: test.mxbus1[0]::BusMaster - unable to access unmapped address 0xA0000000
```

**A:** The bus component cannot find a memory component for the requested address. This means that you have not connected a memory component to the busmaster port of the bus component, reporting a memory region starting at 0xA0000000. You can for example connect an MxMem component to the busmaster port and set its base parameter to "0xA0000000" and the size parameter to "0x10000", occupying 64k from address 0xA0000000.

**Q:** I’m trying to connect my model together with some memory models to the bus. However when my model is connected to the bus component, SoC Designer shows the following error message on startup. Why?

```
ERROR : untitled.mxbus1[0]-BusMaster::connect - Port <MyMemSlavePort> doesn't report any memory regions. Cannot connect to shared bus.
```

**A:** Your component cannot be connected to the bus model because your model does not implement the memory map functions "getNumRegions" and "getAddressRegions". These two functions define the address range that your component is mapped to. If they are not implemented then it is assumed that they are mapped to the FULL address range, therefore the bus model complains, because 0-0x20000 is already occupied by the first two MxMem components.

Ideally you should implement the memory map of your component as a parameter of your component. The following is a simple implementation of these functions:

```cpp
int MyMemPort::getNumRegions() { return 1; }

void MyMemPort::getAddressRegions(MxU64* _start, MxU64* _size, string* _name) {
    if (_start && _size && _name) {
        _start[0] = owner->base; // this is set via a parameter
        _size[0] = owner->size; // this is set via a parameter
        _name[0] = "region0";
    }
    else
```

```cpp
```
{  
  owner->message(CASI_MSG_ERROR,
      "%s: Uninitialized data structures passed in getAddressRegions call",
      owner->getInstanceID().c_str());
}
5 MxIO

5.1 Description
This component drives out signals based on memory-mapped registers.

5.2 Variants
MxIO[4,8] component variants provide 4 or 8 signal master ports.

5.3 Ports

<table>
<thead>
<tr>
<th>Port</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mem</td>
<td>This port needs to be connected to a transaction master port of a core or bus component</td>
</tr>
<tr>
<td>out[0-7]</td>
<td>This port can be connected to any signal slave port. The signal is driven when a value is written to the respective address of the mem port</td>
</tr>
</tbody>
</table>

MxIO ports
Port Access Interface

5.3.1 Read/Write

The read and write functions of the mem port.

<table>
<thead>
<tr>
<th>Port</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mem</td>
<td>Address: A 32-bit byte address in the range from “base” to “base” + 3 or 7. If the address is “base” + 32 then all signals can be driven at once. Data: Accepted values are 0 and 1. If the address is “base” + 32 then values can be between 0 and 15(255) to drive all 4(8) signals. Ctrl: not used.</td>
</tr>
</tbody>
</table>

MxIO port interfaces

5.4 Parameters

The component can be configured to adjust the memory map.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>base</td>
<td>Base address for memory map</td>
</tr>
<tr>
<td>enable_debug_messages</td>
<td>0 = no debug messages</td>
</tr>
<tr>
<td></td>
<td>1 = report read/write accesses</td>
</tr>
</tbody>
</table>

MxIO parameters

5.5 Debug Support

This component does not provide any debug interface.
5.6 Examples

In the example shown below the DLX can cause an interrupt or reset by writing to addresses 0x4000 or 0x4001 respectively.

In order to configure this system for the memory map above, the following parameters need to be set in the components:

- **BUS :: block_size** 0x400
- **RAM :: base** 0
- **RAM :: size** 0x400
- **ROM :: base** 0x400
- **ROM :: size** 0x3C00
- **IO :: base** 0x4000
6 MxCache

6.1 Description
This is a configurable model of a direct mapped cache with profiling support. It can seamlessly be inserted in front of any MxMem or MxBus component. Cache size, cache line size, endianness and "Write back" or "Write through" strategy are configurable. Four non-cacheable areas are available. Misaligned accesses are supported.

The cache is organized in cache lines of a configurable size. The granularity of memory accesses is always a cache line. Please note that in this implementation this can result in a cache line read followed by a cache line write if one single byte is written in “write through” mode. Nevertheless, this cache line is available in the cache for further accesses.

The interface of the cache “cache”-port is the same as the one of MxMem. Calls to read() or write() have to be repeated as long as CASI_STATUS_WAIT is returned. Depending on the success of the operation the component returns CASI_STATUS_OK or CASI_STATUS_ERROR.

The component has the capability to be connected to an MxBus component. This means that it implements the protocol to request access before calling read() or write().

Up to four non-cacheable areas can be defined by their base address and their size in bytes. If a misaligned access affects two cache lines it is not allowed that one of them is in a cacheable area while the other is not.

Since profiling affects the simulation performance it can be disabled by a parameter.

6.2 Variants
MxCache, MxCache2 (dual-port)

6.3 Ports

<table>
<thead>
<tr>
<th>Port</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mem_bus</td>
<td>This port must be connected to a memory or a bus component.</td>
</tr>
<tr>
<td>cache</td>
<td>This port must be connected to the component that uses the memory.</td>
</tr>
</tbody>
</table>

MxCache ports
### 6.4 Port Access Interface

#### 6.4.1 Read/Write

<table>
<thead>
<tr>
<th>Port</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cache</td>
<td>Address: 32-bit address</td>
</tr>
<tr>
<td></td>
<td>Data: 8-, 16-, 32-, or 64-bit value</td>
</tr>
<tr>
<td></td>
<td>Ctrl: =1 for byte (8-bit) access, 2 for halfword (16-bit) access, 3 for</td>
</tr>
<tr>
<td></td>
<td>word (32-bit) access and 4 for word (64-bit) access</td>
</tr>
</tbody>
</table>

**MxCache port interfaces**

The return value of `read()` and `write()` is `eslapi::CASIStatus` enum.

### 6.5 Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>big endian</td>
<td>True = byte order is big endian</td>
</tr>
<tr>
<td></td>
<td>False = byte order is little endian</td>
</tr>
<tr>
<td>cacheLineBytes</td>
<td>Size of a cache line in bytes. Must be a power of 2 and between 8 and 64.</td>
</tr>
<tr>
<td></td>
<td>Cache size must also a multiple of the size of a cache line.</td>
</tr>
<tr>
<td>cacheSizeInBytes</td>
<td>Size of the cache. Must be a power of two (otherwise it will be</td>
</tr>
<tr>
<td></td>
<td>rounded upwards automatically)</td>
</tr>
<tr>
<td>enableCache</td>
<td>True = cache is enabled</td>
</tr>
<tr>
<td></td>
<td>False = cache is disabled, all accesses are uncached</td>
</tr>
<tr>
<td>enableDebugMessages</td>
<td>True = Debug messages enabled</td>
</tr>
<tr>
<td></td>
<td>False = Debug messages disabled</td>
</tr>
<tr>
<td>enableProfiling</td>
<td>True = Profiling enabled</td>
</tr>
<tr>
<td></td>
<td>False = Profiling disabled</td>
</tr>
</tbody>
</table>
noncacheable area base [0-3]  Base address of a non-cacheable areas
noncacheable area size [0-3]  Size of non cacheable area in bytes (0 to disable)
region base  Base address of the region covered by this cache component
region size  Size of the address region covered by this cache component
valAddrBits  Number of valid address bits (other bits will be masked out)
writeBack  True = write strategy is “write back”
            False = write strategy is “write through”

**MxCache parameters**

### 6.6 Debug Support

This component provides a CADI interface for memory display. The contents of the cached memory can be viewed via the debug menu of the SoC Designer Simulator ("View Memory for ...”). A set of pseudo registers is used for profiling data. The contents of the profiling registers can be viewed via the debug menu of the SoC Designer Simulator ("View Registers for ...”). If profiling is enabled the following registers are maintained:

<table>
<thead>
<tr>
<th>Register</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>read-accesses cached</td>
<td>number of read accesses in cacheable address ranges (if cache is enabled)</td>
</tr>
<tr>
<td>write-accesses cached</td>
<td>number of write accesses in cacheable address ranges (if cache is enabled)</td>
</tr>
<tr>
<td>read-accesses uncached</td>
<td>number of read accesses in non-cacheable address ranges. If cache is disabled: all accesses</td>
</tr>
<tr>
<td>write-accesses uncached</td>
<td>number of write accesses in non-cacheable address ranges. If cache is disabled: all accesses</td>
</tr>
</tbody>
</table>

**Register**

**Description**

---

ARM DUI 1042A  Copyright© 2016 ARM Limited. All Rights Reserved  24

Non-Confidential
misaligned-accesses  number of accesses without natural alignment

cross-cacheline-accesses  number of accesses affecting 2 cache lines

read-waits  number of wait cycles caused by memory read accesses

write-waits  number of wait cycles caused by memory write accesses

cache-hits  number of accesses to data available in the cache. If write back is enabled write hits are included.

cache-read-misses  number of read accesses to data not available in the cache

cache-dirty-writes  number of writes of dirty cache lines to the memory

MxCache registers

6.7 Examples
This is an example for a simple system using the MxCache component.
6.8 FAQ

Q: What is the behavior of the cache component if a single byte is written in “write through” mode?

A: Since the granularity of memory accesses is a cache line, a complete line has to be read from memory, modified with the written byte and rewritten back to memory. Nevertheless, this cache line is available in the cache for further accesses.

Q: How to use the endianness parameter?

A: The endianness should be the same as the one of the connected memory.
7 MxTimer

7.1 Description
This component is a memory mapped timer component that can be used to trigger signals on signal ports controlled by memory mapped registers. When enabled, each timer idles until the expiration of a non-zero timer delay. At zero delay the timer is free to count up to its trigger value. At the trigger, an internal pulse count register is initialized with the pulse width. The timer signal port is asserted (active high/low) until the pulse count decrements to zero. At zero, the timer signal port is deasserted, the timer is reset to zero and the sequence repeats. If the timer is disabled while active, the signal port is deasserted and the pulse count is reset to zero.

All memory mapped registers are 32 bits. Therefore, the lower 2 bits of the byte address is ignored, and only 32 bit access is allowed.

<table>
<thead>
<tr>
<th>Address</th>
<th>Register</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>base + 4*p</td>
<td>timer[p]</td>
<td>timer channel cycle count</td>
</tr>
<tr>
<td>base + 4<em>n + 4</em>p</td>
<td>trigger[p]</td>
<td>timer channel trigger point (in cycles)</td>
</tr>
<tr>
<td>base + 8*n</td>
<td>enable</td>
<td>timer channel enable/disable (bit state 1 or 0)</td>
</tr>
<tr>
<td>base + 12<em>n + 4</em>p</td>
<td>delay[p]</td>
<td>timer channel cycle delay (before count up)</td>
</tr>
<tr>
<td>base + 16<em>n + 4</em>p</td>
<td>pulsewidth[p]</td>
<td>timer channel pulse width (in cycles)</td>
</tr>
<tr>
<td>base + 20*n</td>
<td>signal active</td>
<td>timer channel active state high/low (bit state 1 or 0)</td>
</tr>
</tbody>
</table>

MxTimer memory mapped registers

base - Register base address
n - Number of timer channels (output ports)
p - Individual timer channel 0..(n-1)

7.2 Variants
The MxTimer component comes in variants with 1, 4, and 8 signal master ports.
7.3 Ports

<table>
<thead>
<tr>
<th>Port</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mem</td>
<td>This is the port that must be connected to an MxBus compliant master, such as MxBus.</td>
</tr>
<tr>
<td>out[1-8]</td>
<td>These are the ports on which the signals are driven. They must be connected to signal slave ports.</td>
</tr>
</tbody>
</table>

MxTimer ports

7.4 Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>base</td>
<td>Base address for the memory mapping. The size reported to the transaction master port is always 1024 bytes.</td>
</tr>
<tr>
<td>pulse_width[0..(n-1)]</td>
<td>Specifies the number of cycles before a signal is deasserted. Also available in the register map (see above).</td>
</tr>
</tbody>
</table>

MxTimer parameters

7.5 Debug Support

The CADI interface of this component exposes all memory mapped registers, additionally the state of each timer channel can be seen.

7.6 Example

The following C Language example shows the initialization of MxTimer channel 2. Note that any macro definition for MxTimer register access should be compatible with the mapping defined in section 6.1.

```c
// Example C macros for MxTimer register access
#define MAX_PORTS 4 // MxTimer4 component
#define MEM_BASE_ADDR 0x1000 // register base address

#define TIMER_BLK (uint32 *) (MEM_BASE_ADDR)
#define TRIGGER_BLK (uint32 *) (MEM_BASE_ADDR + (4 * MAX_PORTS))
#define TRIG_ENA_BLE (uint32 *) (MEM_BASE_ADDR + (8 * MAX_PORTS))
```


```c
#define TIMER_DELAY_BLK (uint32 *)(MEM_BASE_ADDR + (12 * MAX_PORTS))
#define PULSE_WIDTH_BLK (uint32 *)(MEM_BASE_ADDR + (16 * MAX_PORTS))
#define SIG_ACTIVE (uint32 *)(MEM_BASE_ADDR + (20 * MAX_PORTS))

// Example MxTimer setup using C

void main()
{
    uint32 port = 2;          // timer channel 2
    uint32 *timer = TIMER_BLK;
    uint32 *enable = TRIG_ENABLE;
    uint32 *trigger = TRIGGER_BLK;
    uint32 *pulsewidth = PULSE_WIDTH_BLK;
    uint32 *delay = TIMER_DELAY_BLK
    uint32 *signal = SIG_ACTIVE;

    timer[port] = 0;          // initial timer value
    trigger[port] = 50;       // trigger at cycle 50
    pulsewidth[port] = 5;     // pulse width is 5 cycles
    delay[port] = 5;          // timer idles for 5 cycles before starting count
    signal &= ~(1 << port);  // signal (pulse) is active low
    enable |= (1 << port);    // enable timer channel 2
}
```
8 MemoryMappedCounterModule

8.1 Description
This is an external counter component that is usually used as the clock source for the CPU’s internal timer. It is required for models containing multiple clusters of processors with Generic Timers. It must also be used to run a single processor system where the Generic Timer runs at a different rate to the input clock to the processor.

8.2 Ports

<table>
<thead>
<tr>
<th>Port</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>clk-in</td>
<td>This component must be connected to the clock.</td>
</tr>
<tr>
<td>reset</td>
<td>Reset port of this component.</td>
</tr>
<tr>
<td>cntvalueb</td>
<td>Counter value output port.</td>
</tr>
</tbody>
</table>

MemoryMappedCounterModule ports

8.3 Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable Debug Messages</td>
<td>ON/OFF switch for debug messages.</td>
</tr>
</tbody>
</table>

MemoryMappedCounterModule parameters
9 MxSigDriver

9.1 Description
This component can be used to generate custom signal waveform. An example use case may be to generate a clock enable signal.

9.2 Ports

<table>
<thead>
<tr>
<th>Port</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rst</td>
<td>When driven with a non-zero value, triggers a reset on MxSigDriver.</td>
</tr>
<tr>
<td>sm</td>
<td>Generated waveform will be driven out on this port.</td>
</tr>
</tbody>
</table>

MxSigDriver ports

9.3 Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>initial value</td>
<td>The initial signal value at reset.</td>
</tr>
<tr>
<td>initial delay</td>
<td>Number of cycles after reset before the waveform is generated.</td>
</tr>
<tr>
<td>cycles asserted</td>
<td>Number of cycles the signal should be asserted</td>
</tr>
<tr>
<td>cycles de-asserted</td>
<td>Number of cycles the signal should be deasserted.</td>
</tr>
</tbody>
</table>

MxSigDriver parameters
10 intVector

The intVector component provides an interface for the interrupt ports of a processor Cycle Model. It takes simple 1-bit interrupt signals and provides the IRQ vector expected at the processor Cycle Models interrupt port.

The intVector provides a simple 1-bit interface for the interrupt drivers:

\[
\text{driveSignal}(\text{uint32}_t \text{ irqVal}, \text{uint32}_t* \text{ extVal})
\]

- irqVal: 1-bit interrupt signal value
- extVal: not used

The intVector forwards the interrupt onto the processor Cycle Model which expects the following in the driveSignal() method on its interrupt port:

\[
\text{driveSignal}(\text{uint32}_t \text{ irqSrc}, \text{uint32}_t* \text{ irqVal})
\]

- irqSrc: denotes the interrupt source
- irqVal: holds the interrupt signal value (0 or 1)

The intVector provides 32 interrupt ports. “offset” parameter is provided in case the processor accepts more than 32 interrupts sources. The offset value will be added to the port number when the intVector component forwards the interrupt onto the processor.
11 MxStub

11.1 Description

This component is an MxBus compliant stub model that can be programmed using a script. The symbol of the MxStub component in SoC Designer is shown below.

During reset the user can select the script file (mxscr file) to be loaded. The disassembly window can be used to debug the script as it is being processed, while the register window can display the global variables of the script.

For information on writing scripts, see the MxScript Reference Manual (DUI 1011A).

11.2 Ports

<table>
<thead>
<tr>
<th>Port</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mx_out</td>
<td>This is a port that must be connected to an MxBus compliant slave. It is used to drive read or write transactions on the connected bus.</td>
</tr>
<tr>
<td>mx_in</td>
<td>This transaction slave port can be connected to an MxBus compliant master. It is used to accept read or write transactions on the connected bus for access to internal slave memory.</td>
</tr>
</tbody>
</table>

*Note:* On stub components, accessing transaction slave ports using MxScript is not supported. Use a memory component if scripting is required.

| p_in[0..3] | These signal slave ports can be used to feed input signals into the script driven component. The script component can then wait on an event on these ports. |
These signal master ports are used to drive signals into the system.

### 11.3 Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base Address</td>
<td>Base address of internal slave memory.</td>
</tr>
<tr>
<td>Big Endian</td>
<td>Endianness of internal slave memory.</td>
</tr>
<tr>
<td>CPP Include Path</td>
<td>Include path for the C preprocessor. Can be used to include header files from other directories.</td>
</tr>
<tr>
<td>Data Width</td>
<td>Width of the Mx data. Allowed values are 32 and 64-bits. Default is 32.</td>
</tr>
<tr>
<td>Enable Debug Messages</td>
<td>Enables debug messages in the script component. Default is false.</td>
</tr>
<tr>
<td>Memory Initialization Value</td>
<td>Memory initialization value, typically a 32-bit word. Default is zero. 32 and 24-bit initializers get treated as a 32-bit word. A 16-bit initializer is replicated in the upper 16-bits of a 32-bit word. An 8-bit initializer is replicated throughout a 32-bit word at 8-bit intervals. Initial values greater than 32-bits get truncated to 32-bits.</td>
</tr>
<tr>
<td>Region Size</td>
<td>Size of the transaction slave internal memory. Default is 1024 bytes.</td>
</tr>
<tr>
<td>Transaction Port Trace</td>
<td>Echoes transaction slave bus cycle information to the output window.</td>
</tr>
</tbody>
</table>

MxStub parameters
11.4 Memory Spaces

MxStub has two memory spaces: MxScript and External. It is also unclear which MxStub memory related parameters impact which of these two memory spaces. The MxStub memory related parameters include "Base Address", "Big Endian", "Memory Initialization Value", and "Region Size".

11.5 Debug Support

The component offers CADI registers and disassembly support. The disassembly window shows the current script being processed, the register window shows information like line number and current file, but also all global script variables. Access to internal slave memory is accessible through an CADI memory window.

11.6 Example

The following shows parts of the script used in the example "MxStubExample.mxp":

```c
#include "MxMacros.h" // includes macros needed for driving transactions

#define MEM_BASE_ADDR_ADDR 0x1000
#define INIT_DATA 0x1401
#define MEM_VALUES 4

int i, addr, data;

// Initialize Memory: write words 0x1401..1404 into addresses 0x1000..100c
i = 0;
while (i < MEM_VALUES)
{
   // write words (macro MX_WORD_WRITE defined in MxMacros.h)
   MX_WORD_WRITE("mx_out", MEM_BASE_ADDR_ADDR + 4 * i, INIT_DATA + i);
   i++;
}

wait(10); // wait for 10 cycles

while (i < MEM_VALUES*4)
{
   // write bytes
   MX_BYTE_WRITE("mx_out", MEM_BASE_ADDR_ADDR + 4 * i, i);
   i++;
}

message(MX_MSG_INFO, "Memory initialized", 0); // emit message to output window
wait(10);

// endless loop generates pulses of different length
```
int length = 200;
while(1)
{
    // emit a pulse in port "p_out1"
    int p_out1 = getPortID("p_out1");
    setPortData(p_out1, "DATA", 0x1);
    drivePort(p_out1);
    wait(length);
    setPortData(p_out1, "DATA", 0x0);
    drivePort(p_out1);

    // wait for interrupt request on port "p_in0"
    int p_in0 = getPortID("p_in0");
    waitEvent(p_in0);
    wait(2000);

    // increase length
    length *= 3;
    length /= 2;
    if (length > 1000)
        length = 200;
};
12 Semihost and Semihost2

12.1 Description
The Semihost and Semihost2 components are used for performing semihosting functions for processor models. Use the:

- Semihost2 component with the following Cycle Models:
  - Cortex-A53
  - Cortex-A57
  - Cortex-A72
- Semihost component with all other components.

12.2 Ports

<table>
<thead>
<tr>
<th>Port</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>semihost</td>
<td>This port is used to connect a core model which requires external semihosting.</td>
</tr>
</tbody>
</table>
### 12.3 Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>heap base address</td>
<td>Base address of the semihosting functions’ heap space</td>
</tr>
<tr>
<td>heap limit</td>
<td>Maximum heap address</td>
</tr>
<tr>
<td>stack base address</td>
<td>Base address of the semihosting functions’ stack space</td>
</tr>
<tr>
<td>stack limit</td>
<td>Minimum stack address</td>
</tr>
<tr>
<td>use system time</td>
<td>Relevant to semihost functions which requires calculating time. If true, for all time calculations system time functions are used (standard C functions). If false, time specific quantities are calculated using the simulator elapsed cycles.</td>
</tr>
<tr>
<td>command line</td>
<td>Command line arguments for program being executed.</td>
</tr>
<tr>
<td>svc debug</td>
<td>If true, emit debug messages.</td>
</tr>
</tbody>
</table>

**Semihost and Semihost2 parameters**
13 Miscellaneous Components

13.1 FOUT

This component can be used to connect a signal master port to multiple signal slave ports. Every time the master calls the driveSignal function, this function call will be passed on to every slave.

The FOUT component has no parameters.

13.2 BDEC

This component can be used to connect a transaction master port to multiple transaction slave ports. Every time the master calls the one of the interface function, this function call will be passed on to the corresponding slave.

13.2.1 Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>minBlockSize</td>
<td>Minimum block size of the address space covered by BDEC.</td>
</tr>
</tbody>
</table>

BDEC parameters
13.3 CDIV

This component can be used to drive connected components at a lower clock rate than the other components. It has an adjustable clock-ratio and offset.

<table>
<thead>
<tr>
<th>Port</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACLKEN</td>
<td>AXI clock enable port. This signal slave port is used to enable or disable the cdiv clock output. A value of ‘1’ enables the clock output and a value of ‘0’ disables the output.</td>
</tr>
<tr>
<td>freq_in</td>
<td>The value driven on this port is used as the clock ratio between the CDIV clock input and the clock output (“clock-ratio” parameter serves the same purpose). This can be used to dynamically alter the clock frequency.</td>
</tr>
<tr>
<td>clk_in</td>
<td>This port is used internally. Leave unconnected.</td>
</tr>
<tr>
<td>clk_out</td>
<td>Clock master. This port should be connected to the clock input port of all components that need to be clocked at the slower ratio.</td>
</tr>
</tbody>
</table>

13.3.2 Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>clock-delay-update</td>
<td>This flag determines when update() is called in this CDIV. If this flag is false then update() is called at the same cycle of communicate(); otherwise update() is called one cycle before the next communicate() is called. Default is false.</td>
</tr>
<tr>
<td>clock-enable</td>
<td>This parameter can enable/disable the clocking of all connected components. Default is 1.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>clock-offset</td>
<td>This parameter specifies how many cycles to count down before starting to drive the clock signal. Default is 0.</td>
</tr>
<tr>
<td>clock-ratio</td>
<td>The ratio between the clock slave and the clock master interface is specified in this parameter. Default is 1.</td>
</tr>
<tr>
<td>clock-ratio-denominator</td>
<td>This parameter divides the clock-ratio, allowing for non-integer clock ratios when different from 1. Default is 1.</td>
</tr>
<tr>
<td>ACLKEN-clock-delay</td>
<td>This parameter should be set to a value greater than or equal to '2'. The clock output gets delayed by these many clock cycles after every ACLKEN port value of '1'</td>
</tr>
<tr>
<td>clock-use-ACLKEN</td>
<td>When set to true, ACLKEN port is used for generating the clock out from CDIV component</td>
</tr>
</tbody>
</table>

**CDIV parameters**

It is possible to implement a phase shift between components by e.g. using two clock dividers, both with a clock ration of “2”, but one with a clock offset of 1.

The clock-ratio-denominator provides non-integer clock rate change. For instance, the clock change from 300MHz to 200MHz can be specified by clock-ratio=3 and clock-ratio-denominator=2. The clock-ratio-denominator should be no larger than clock-ratio.