Application Note 222

Cortex-M1 TCM Initialization Considerations for System Designers

Document number: ARM DAI 0222A
Issued: 4th June, 2009
Copyright ARM Limited 2009
Application Note 0222
Cortex-M1 TCM Initialization Considerations for System Designers

Copyright © 2009 ARM Limited. All rights reserved.

Release information

<table>
<thead>
<tr>
<th>Date</th>
<th>Issue</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>June 2009</td>
<td>A</td>
<td>First release</td>
</tr>
</tbody>
</table>

Proprietary notice

Words and logos marked with © and ™ are registered trademarks owned by ARM Limited, 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 Limited 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.

Confidentiality status

This document is Open Access. This document has no restriction on distribution.

Feedback on this Application Note

If you have any comments on this Application Note, please send email to errata@arm.com giving:

- the document title
- the document number
- the page number(s) to which your comments refer
- an explanation of your comments.

General suggestions for additions and improvements are also welcome.

ARM web address

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

1. Introduction .................................................................................................................. 1-1

2. Initialisation Considerations ......................................................................................... 2-1
  2.1 Initialisation in Hardware ......................................................................................... 2-1
  2.2 Initialisation in Software ......................................................................................... 2-1

3. References ..................................................................................................................... 3-1
1. Introduction

The ARM Cortex-M1 processor was developed for use with FPGAs (Field Programmable Gate Arrays). The processor provides interfaces which enable the system designer to attach Tightly Coupled Memories (TCMs). These memories allow very fast access to time critical code or data.

Cortex-M1 has two separate TCM interfaces, one for the instruction side (ITCM) and one for the data side (DTCM). A TCM memory is normally built using FPGA internal memory. TCMs offer a significant performance improvement compared with execution from the AHB-Lite interface. Most FPGAs provide the possibility to pre-load the content of this internal memory at configuration time, others do not.

After reset, the processor automatically starts loading the first two words at addresses 0x0 and 0x4, where the Stack Pointer (SP) for the Main Stack and the address for the reset handler are located. These locations need to be initialized at boot time. This implies that it is not possible to boot from the ITCM unless it is pre-loaded at FPGA configuration time and therefore that the processor must boot from the AHB-Lite interface. Usually in such a situation, bootstrap loader software is used to initialize the memory by copying code and/or data from initialized memory via the AHB-Lite interface into the ITCM.

From revision r1 of the Cortex-M1, there is an option to alias the start address of the ITCM to two different memory locations, 0x0 and 0x10000000. This option can be used to support preloading of the ITCM even if the FPGA architecture itself does not allow the initialization of the memories at configuration time.

This Application Note describes the features of the processor which allow the user to successfully preload an ITCM using the alias capability of Cortex-M1.
2. Initialisation Considerations

2.1 Initialisation in Hardware

Some FPGA targets allow the initialization of internal memories at configuration time of the FPGA. The compiled program code is used at the synthesis and place and route (P&R) stages of the implementation flow to generate the final FPGA bitstream. This offers the maximum code compatibility because no boot loader is needed. On the other hand it is necessary to rebuild the bitstream if the program code changes.

Please refer to your FPGA vendor and tool vendor documentation to see how internal memories can be preloaded for your target.

However, initialization using the FPGA bitstream might not be possible or desirable in some applications. In these cases, software can be used to initialise the TCMs.

2.2 Initialisation in Software

The aliasing scheme of Cortex-M1 enables software to initialize the ITCM. This mechanism is particularly useful when it is not possible to initialize the internal FPGA memories at configuration time.

2.2.1 Alias Scheme on Cortex-M1

From revision r1 onwards, Cortex-M1 provides an aliasing scheme for the ITCM which allows the memory to be aliased within the memory map. This memory map is seen by the processor as well as a debugger when used in halt mode. The aliasing scheme can be used to preload the ITCM and use it afterwards e.g. for the vector table and/or exception handlers.

Note: The aliasing scheme is only applicable to the ITCM and is not available for the DTCM.

![Figure 1 Lower 512 MB of the Cortex-M1 memory map](image)

Figure 1 shows the first 512 MB of the Cortex-M1 memory map. It contains aliases for the ITCM and external memory i.e. accesses which are routed over the external AHB-Lite interface.

**Lower Alias**

The ITCM can have a maximum size of 1 MB and if the Lower Alias is activated then it starts from address 0x0. Every access over the implemented size for the ITCM is passed onto the external AHB-Lite interface.

**Upper Alias**

Similar to the Lower Alias, the Upper Alias places the ITCM at start address 0x10000000.
Using the Scheme

The aliases can be used to access the address space of an enabled alias from either the processor or the debugger to the ITCM. When either alias is enabled, any access to that address space by the processor or debugger is directed to the ITCM. When the corresponding alias is disabled, accesses to the corresponding region are passed directly to the AHB-Lite interface. Both aliases map to the same physical memory in the ITCM so, for example, address 0x10000008 in the upper alias and address 0x00000008 will access both address 0x000008 in the ITCM. However, when the upper alias is disabled, accesses to this region pass unmapped onto the AHB-Lite interface so an access to 0x10000008 would access 0x10000008 in external memory via the AHB-Lite interface.

Figure 2 shows the usage of the aliasing scheme. When an alias is enabled, all data accesses and instruction fetches to an address within the enabled region are directed to the ITCM.

![Figure 2 Usage of the aliasing scheme]

The Auxiliary Control Register (ACTRL) contains the enables for the aliasing scheme, ITCMLAEN (ITCM Lower Alias Enable) and ITCMUAEN (ITCM Upper Alias Enable). Figure 3 shows the ACTRL register which can be modified by software to allow dynamic control of both aliases.

Refer to the Cortex-M1 TRM [1] for a detailed description of this register.

![Figure 3 Auxiliary Control Register (ACTRL) of Cortex-M1]

The processor provides configuration pins CFGITCMEN[1:0] for enabling the aliases within the ACTRL register as required from reset:

- **CFGITCMEN[1]** sets the ITCMUAEN bit in the ACTRL
- **CFGITCMEN[0]** sets the ITCMLAEN bit in the ACTRL

**Note:** It is also possible to set both enable pins at the same time. This enables the ITCM on the Upper Alias and Lower Alias. Further it is possible to disable the ITCM completely by disabling both enables.

When the content of the ACTRL is changed in software, additional synchronisation instructions must be used after the control bits are updated to ensure that the change in aliasing occurs before subsequent instructions attempt to access the aliased regions. It is also recommended that exceptions are disabled on the system level, this includes interrupts. The following code shows the recommended way of modifying the content of the ACTLR register:

```c
// Example code snippet

ITCAEN = 1;
ITCMUAEN = 1;
ITCMUAEN &= 0;
ITCMUAEN &= 0;
```

```c
// Example code snippet

ITCAEN = 1;
ITCMUAEN = 1;
ITCMUAEN &= 0;
ITCMUAEN &= 0;
```
2.2.2 Considerations for the initialisation

There are several ways to use the alias scheme of Cortex-M1. However, the following steps are usually required in any initialisation sequence:

- Set CGITCMEN values on the processor accordingly (e.g. Upper Alias = 1, Lower Alias = 0)
- Reset of the core
- Running the boot loader
  - Start to copy the application image from external memory into ITCM, using the Upper Alias (e.g. by using the ITCM size as a boundary for the copy)
  - Disable interrupts
  - Set the remap bits in the ACTRL register accordingly (e.g. Upper Alias = 0, Lower Alias = 1)
  - Executing a DSB and ISB
  - Enable interrupts
  - Branch to the main application

Boot code as part of the application

There are several ways in which the boot loader can be built. One way is to include the boot loader as a part of the application. Figure 4 shows an example of a boot loader included in the application:

1. The application and the boot loader reside in non-volatile memory, as one image.
2. The aliasing pins on the processor are set to enable the upper alias only. The boot loader copies the whole image into the ITCM, including the boot loader itself.
3. The boot loader disables the upper alias and enables the lower alias over software.
Before such a scheme is implemented, the following considerations should be checked:

- Boot loader can be located in the same address space as the enabled ITCM aliases as it remains accessible after the lower ITCM alias has been enabled.
- Changes in the boot code results in re-generating the whole application and vice versa.
- Consumes more space in the ITCM, because the boot loader is loaded into the ITCM.
- Boot loader might be more complex

For performance reasons it may be desirable to run the initial boot loading process only once, on a hard reset. Therefore the boot loader needs to check if an initialisation is required, for example on a soft reset.

**Boot code as a separate application**

Figure 5 shows how a boot loader could be isolated from the application:

1. The application image and the boot loader reside separately in non-volatile memory.
   **Note:** The boot loader contains its own vector table and exception handlers.
2. The aliasing pins on the processor are set to enable the upper alias only. The boot loader copies only the application image into the ITCM, without copying the boot loader itself.
3. The boot loader disables the upper alias and enables the lower alias over software.

Similar to the previous example the programmer should check the following considerations:

- Boot loader must not be located in the same address space as the ITCM aliases since it must remain accessible after the lower ITCM alias is enabled in order to branch to the application.
- Boot loader needs to be generated only once.
- Consumes no space in the ITCM.
### 3. References

#### Table 1 Document References

<table>
<thead>
<tr>
<th>Ref</th>
<th>Author(s)</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ARM</td>
<td>Cortex-M1 Technical Reference Manual</td>
</tr>
<tr>
<td>2</td>
<td>ARM</td>
<td>ARMv6-M Architecture Reference Manual</td>
</tr>
</tbody>
</table>