TriCore
Using Data Access Overlay Functionality with TC1766
LEGAL DISCLAIMER

THE INFORMATION GIVEN IN THIS APPLICATION NOTE IS GIVEN AS A HINT FOR THE IMPLEMENTATION OF THE INFINEON TECHNOLOGIES COMPONENT ONLY AND SHALL NOT BE REGARDED AS ANY DESCRIPTION OR WARRANTY OF A CERTAIN FUNCTIONALITY, CONDITION OR QUALITY OF THE INFINEON TECHNOLOGIES COMPONENT. THE RECIPIENT OF THIS APPLICATION NOTE MUST VERIFY ANY FUNCTION DESCRIBED HEREIN IN THE REAL APPLICATION. INFINEON TECHNOLOGIES HEREBY DISCLAIMS ANY AND ALL WARRANTIES AND LIABILITIES OF ANY KIND (INCLUDING WITHOUT LIMITATION WARRANTIES OF NON-INFRINGEMENT OF INTELLECTUAL PROPERTY RIGHTS OF ANY THIRD PARTY) WITH RESPECT TO ANY AND ALL INFORMATION GIVEN IN THIS APPLICATION NOTE.

Information

For further information on technology, delivery terms and conditions and prices please contact your nearest Infineon Technologies Office (www.infineon.com).

Warnings

Due to technical requirements components may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies Office.

Infineon Technologies Components may be used in life-support devices or systems only with the express written approval of Infineon Technologies, if a failure of such components can reasonably be expected to cause the failure of that life-support device or system, or to affect the safety or effectiveness of that device or system. Life support devices or systems are intended to be implanted in the human body, or to support and/or maintain and sustain and/or protect human life. If they fail, it is reasonable to assume that the health of the user or other persons may be endangered.
**We Listen to Your Comments**

Any information within this document that you feel is wrong, unclear or missing at all? Your feedback will help us to continuously improve the quality of this document. Please send your proposal (including a reference to this document) to:

**mcdocu.comments@infineon.com**
# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1    Introduction</td>
<td>5</td>
</tr>
<tr>
<td>2    Data Access Overlay Functionality</td>
<td>6</td>
</tr>
<tr>
<td>2.1  Program Memory Unit (PMU) Overlay Control Registers</td>
<td>7</td>
</tr>
<tr>
<td>3    Code Example</td>
<td>7</td>
</tr>
<tr>
<td>3.1  Running the Code</td>
<td>7</td>
</tr>
</tbody>
</table>
1 Introduction

This document provides a method for overlaying code memory onto data memory with an example code. This method is primarily intended for the AUDO-NG Family with Overlay memory. An overview of this functionality is shown in Figure 1. An example is shown in Figure 2.

This application note assumes that the reader is already familiar with the TriCore architecture, the Tasking Compiler, the PLS debugger and the TC1766 TriBoard / StarterKit. The reader should also have read the section, Data Access Overlay functionality in the TC1766 User Manual.

Please note that the code delivered with this application note is intended for demonstration purpose only. Neither is its quality nor its robustness guaranteed.

Figure 1 Redirection of Data Read Accesses from Code Memory to PMU SRAM (Internal Data) Memory


\section{Data Access Overlay Functionality}

The data overlay functionality provides the capability to redirect data read accesses from internal code memory to data accesses from the PMU SRAM. This functionality makes it possible, for example, to modify program parameters (which are typically stored in the code memory) during run time of a program. This is commonly referred to as a calibration sequence. Instruction fetches are not affected by the PMU data read access redirection capability. Note that read and write data accesses from/to code memory are redirected.

During the calibration sequence (cycle), calibration memory (usually an internal SRAM) is temporarily utilized by the system. The sequence begins by the system copying its current parameter tables from non-volatile memory (usually Flash) into volatile memory (SRAM). Once copied, an overlay mechanism is instituted (configure required PMU Overlay Control registers). The system now runs as it normally would. The user program thinks it is reading its parameters in their normal address locations in nonvolatile memory. But in reality, they are now being accessed from the SRAM locations. Once the calibration cycle has been completed, the modified parameters that are in the volatile memory need to be stored back to the non-volatile memory. Then the calibration memory is removed from the system and the sequence is complete.
2.1 Program Memory Unit (PMU) Overlay Control Registers

In the TC1766, the 8 Kbyte PMU SRAM memory can be divided into a maximum of sixteen overlay memory blocks, with programmable base address and block sizes, which can be individually enabled for overlay functionality. The block size of each overlay memory block can be in the range of 2 bytes up to 512 bytes.

Users need to configure three PMU Overlay Control Registers to enable the overlay function.

- The overlay target address in the Overlay Target Address Register OTARx (x = 0-15), which determines the base address of the code memory data block x to be redirected (i.e. 0xA000C000, as shown in Figure 2)
- The base address of the overlay memory block in the PMU SRAM in the Redirected Address Base Register RABRx (i.e. 0xC0001000, as shown in Figure 2)
- A mask in the Overlay Mask Register OMASKx, defining the size of the block, the address bits to check for an address match, and which bits are used from the target address and which from the redirected address base (i.e. 0x0FFFFF00, 256bytes, as shown in Figure 2)

3 Code Example

The code example provided, demonstrates how to perform an overlay of 256 bytes of code memory. The memory size is arbitrary, but user needs to be aware that all load/store operations to code memory will be redirected when the overlay control registers are configured / enabled.

The TC1766 TriBoard / StarterKit was used to verify the operation of the code were correct. The software environment used was Tasking EDE (TriCore-PCP Vx-toolset) along with their CrossView Pro debugger or pls UDE / UAD2 debugger. Two Tasking EDE project option files are included to run the example code either from the internal SPRAM (0xD4000000) or from the internal program flash (0xA0000000).

The example code is generated using Digital Application virtual Engineer (DAvE) for TC1766. It is compiled using Tasking EDE v2.5r2. The code is located inside the program flash (0xA0000000). When running the code on the TC1766 TriBoard / StarterKit, the onboard LED will also be toggling periodically. With the overlay control registers configured / enabled, although the code is writing data (0x00 – 0xFF) to program flash address of 0xA000C000 – 0xA000C0FF, the actual data are written to SRAM address of 0xC0001000 – 0xC00010FF (refer to Figure 2).

In addition, the code will do a verification of the data written to the Overlay SRAM location. If the value written is not correct, the code will stop running and the onboard LED will not be toggling.

3.1 Running the Code

1. Unzip the Tasking Project attached with this document in the folder C:\Infineon\.
2. Start a debugger session (e.g. pls UDE) and load the executable “overlay.elf”.
3. Run the application.

By activating the memory window of the debugger, users can see that data is written to the SRAM memory (0xC0001000 – 0xC00010FF) although we are writing to the program flash as shown in Figure 3.
As seen in Figure 3, the three overlay control registers are configured / enabled as follows.

- **PMU_OTAR0 = 0x0000C000** (PMU Overlay Target Address Register 0, TBASE = 0x000C000, PFlash = 0xA000C000)
- **PMU_RABR0 = 0x80001000** (PMU Redirected Address Base Register 0, enable overlay, OBASE = 0xC0001000, from 0x00 to 0xFF)
- **PMU_OMASK0 = 0x0FFFFFF00** (PMU Overlay Mask Register 0, 256 bytes)

![Figure 3 Memory view and registers view of PMU SRAM using the pls debugger](image-url)