

# AP32085

## TriCore

A practical introduction to the  
TC1766 Memory Checker Module  
driver

Microcontrollers



Never stop thinking.

<b>Revision History:</b>		2005-03	<b>V 1.0</b>
Previous Version:		-	
Page	Subjects (major changes since last revision)		

**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](mailto:mcdocu.comments@infineon.com)**



**Edition 2005-03**

**Published by  
Infineon Technologies AG  
81726 München, Germany**

**© Infineon Technologies AG 2006.  
All Rights Reserved.**

#### **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](http://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 only be used in life-support devices or systems 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.

<b>Table of Contents</b>		<b>Page</b>
1	Introduction .....	4
2	Build a sample program .....	6
3	Add a MCHK driver to the sample program.....	11
4	Add an Alternate Boot Mode (ABM) header to the sample program .....	15
5	Goodies.....	16
5.1	CRC debugger macro .....	16
5.2	CRC command line tool .....	17
6	Example code .....	19
7	Glossary .....	20

## 1 Introduction

This Application Note explains the usage of the Memory Checker (MCHK) software driver. It assumes that the reader is already familiar with the Tricore architecture, the Tasking Compiler and the PLS debugger. The reader should also have read the chapter 4.1.11 Alternate Boot Mode and chapter 9 Direct Memory Access Controller (DMA) of the TC1766 User Manual (Table 1). The development environment used to build the sample program is listed in Table 2.

**Table 1 Documents**

Title	Version
TC1766, System Units, User Manual	2.1
TriBoard TC1766, Hardware Manual, User Manual	1.0

**Table 2 Tools**

Tools	Version	Remarks
DAvE	2.1r22	<a href="http://www.infineon.com/dave">http://www.infineon.com/dave</a>
DAvE TC1766 Plug-in	3.0	available on request
SAG	1.2	
Tasking Vx Compiler	2.1r2	<a href="http://www.tasking.com">http://www.tasking.com</a>
UDE Debugger	1.10.02	<a href="http://www.pls-mc.com/">http://www.pls-mc.com/</a>
GNU bash	2.05	<a href="http://www.cygwin.com">http://www.cygwin.com</a>
GNU binutils	2.13	<a href="http://www.cygwin.com">http://www.cygwin.com</a>
tricore-readelf	2.13	<a href="http://www.hightec-rt.com/tricore.html">http://www.hightec-rt.com/tricore.html</a>
TriBoard TC1766	TC1766.102	
Windows 2000		

The MCHK driver is a thin software layer on top of the MCHK hardware module. For an architectural overview see Figure 2-1 in the TC1766 User Manual. The MCHK module allows checking the data consistency of memories and was especially developed to check the flash memory. The Program Flash and the Data Flash provide error correction (ECC) of single-bit errors within a 64-bit read double-word, resulting in an extremely low failure rate. Double-bit errors causes a Flash interrupt (see TC1766 User Manual chapter 7.2.9) but triple or even more-bit errors, which are hardly found by the hardware, can be detected with MCHK module. For this purpose the MCHK

**Introduction**

module implements a fast polynomial checksum calculation. A software implementation of the algorithm is given by:

```
unsigned long crc_buffer(unsigned long crc,
                        const unsigned long *p,
                        unsigned long sz)
{
    const unsigned long poly = 0xEDB88320;
    unsigned long      tmp1, tmp2;

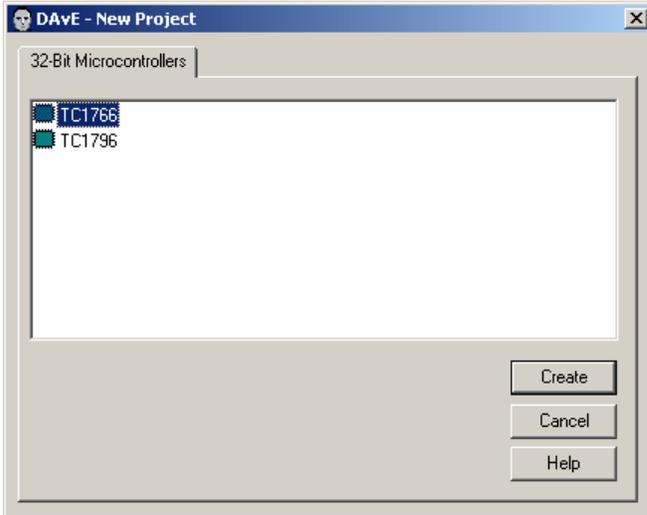
    while(sz--)
    {
        tmp1 = 0;
        tmp2 = crc & poly;
        for(int i = 0; i <=31; i++ )
            tmp1 ^= ((tmp2 >> i) & 1);

        crc = *p++ ^ ((crc << 1) | tmp1) ;
    }
    return crc;
}
```

Within three steps this Application Note will guide you through the usage of the MCHK driver. First a simple program will be created using DAVE. The program will blink the on-board LED of the TriBoard. Next an MCHK driver will be added to the program. The driver calculates a checksum over a flash memory range. The Signature Analysis Generator (SAG) tool, which is distributed with this Application Note, is used to calculate the checksum in advance directly from the elf file. Finally an Alternate Boot Mode (ABM) header will be added to the application using SAG.

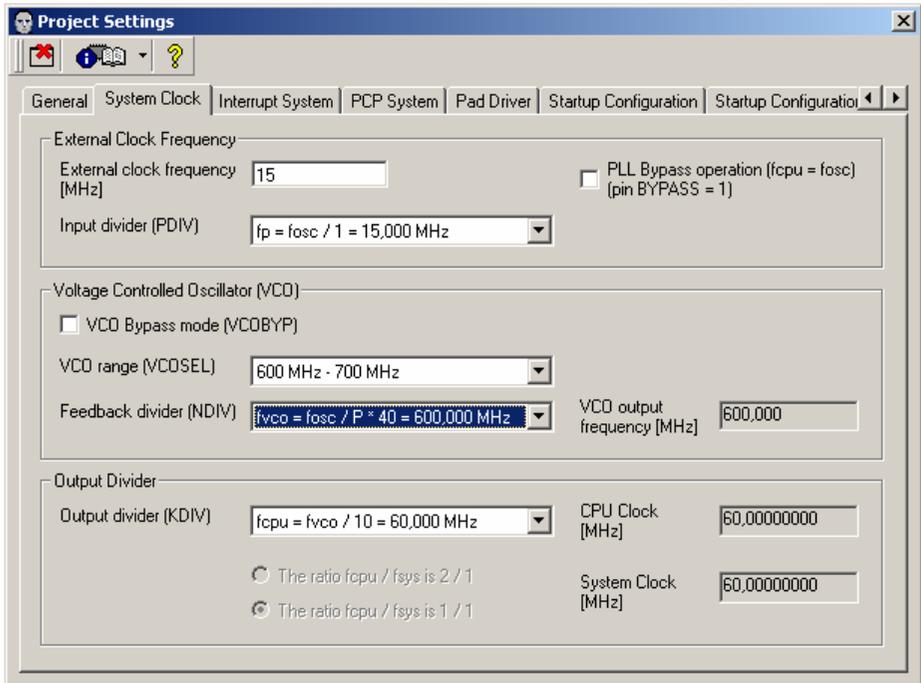
## 2 Build a sample program

Start DAVe and create a new TC1766 project (Figure 1).



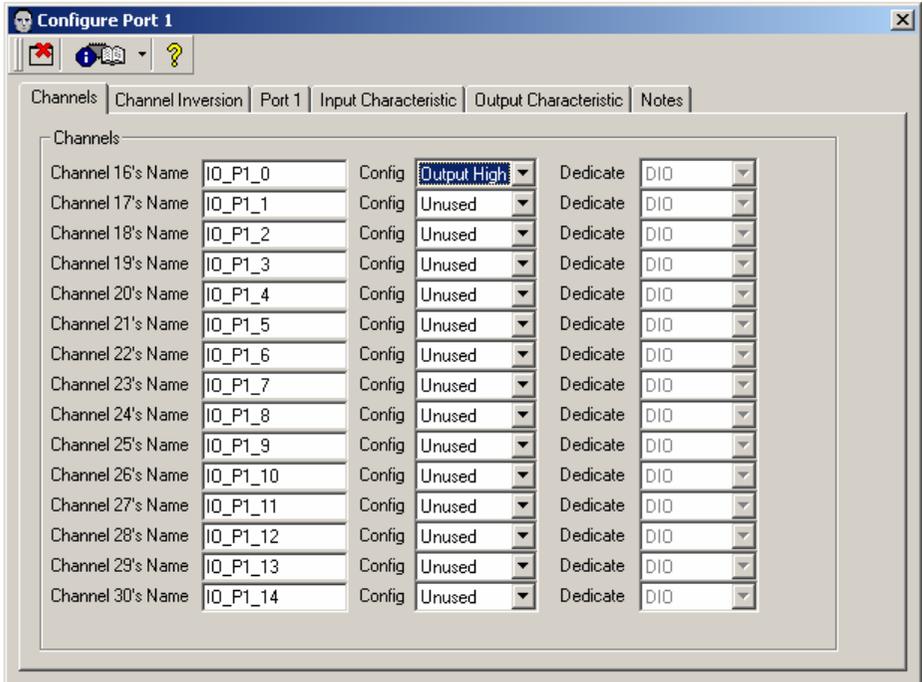
**Figure 1 DAVe New Project Dialog**

If the TC1766 is not in the list of 32-Bit Microcontrollers the Plug-In has to be installed. The TC1766 Plug-In is part of this Application Note. Please follow the instructions in the DAVe manual how to install Plug-Ins.



**Figure 2 DAVE Project Settings Dialog, Page System Clock**

Open the "Project Settings" from the File menu. Keep the default settings on the "General" page and goto the "System Clock" page (Figure 2). In general the TC1766 TriBoard is shipped with a 15 MHz external quartz. Set the "External clock frequency [MHz]" to 15, set the "Feedback divider (NDIV)" to 600 MHz and keep the default values for the rest of the dialog. This configuration will lead to a CPU and System Clock of 60 MHz.



**Figure 3 Configure Port 1 Dialog, Page Channels**

Finally pin 0 of port 1, which is connected to the LED of the board, needs to be configured. For detailed information read the TriBoard Manual. Click on the Port bubble in DAVE and on the "Ports" page press the "Configure Port 1" button. Configure the Channel 16, which is port 1 pin 0, to "Output High" (Figure 3).

Close the Dialog and go to the "Functions" page and check the Initialization Function. The configuration is done. Close the configuration dialogs and save the project as 'mchk.dav'. Generate the source files by selecting "Generate Code" from the File menu.

Dave generates the followings source and header files: The project initialization function 'MAIN.c', 'MAIN.h', the port driver 'LLD\_DIO.c', 'LLD\_DIO.h', C Startup code 'cstart.c', CAS data types definition 'LLD.h', 'CAS\_DTYP.h' and the register definition file for the TC1766 'TC1766Regs.h'.

In the next step the elf file will be build using the Tasking compiler and linker. Open Tasking IDE and create a new project. Open the Project options and go to Processor->"Processor definitions". Select "TC1766". Go to "Compiler"->Optimization

**Build a sample program**

and select "No Optimization" from the dropdown menu. Go to Processor->Startup and deselect the "Automatic copy and link cstart.asm to your project" checkbox. Go to Linker->"Script File"->"Special Areas" and set the values to the following addresses:

```
RESET start address           0xA0000000
Libraries start address       0xA0000400
Interrupt table start address 0xA0002000
Trap table start address      0xA0003F00
```

Go to Linker->"Script File"->"Internal Memory" and define the program flash

Name = pflash, Alloc = ON, Type = ROM, Size = 1504k

Close the project options. Open the project properties and add the files 'MAIN.c', 'LLD\_DIO.c' and 'cstart.c' to the project.

In 'LLD\_DIO.h' add two macros which were part of former plug-ins but did not make it to the TC1766 plug-in

```
// USER CODE BEGIN (IO_Header,3)
#define DIO_vTogglePin(PinName)    TOGGLE##PinName
#define TOGGLEIO_P1_0              P1_OMR.I = 0x00010001
// USER CODE END
```

In MAIN.c add a delay function

```
// USER CODE BEGIN (MAIN_General,9)
void delay(unsigned long n)
{
    n *=0x100000;
    while(n--)
        __nop();
} // USER CODE END
```

and a forever loop which toggles the TriBoard LED.

```
// USER CODE BEGIN (Main,9)
while(1)
{
    DIO_vTogglePin(IO_P1_0);
    delay(8);
}
// USER CODE END
```

Build the elf file. Ensure the HW Boot Configuration DIP-switch is set to ON-OFF-ON-ON-OFF-ON-OFF-OFF. The ON position of the switch is equal to a logical LOW, that means HWCFG[3..0] is 0010 (See the TriBoard Manual chap. 5.1.1). The board will start from internal flash at 0xA0000000.

Open the UDE debugger and create a new workspace with a TC1766 target. Install the UDE memtool, select "Flash programming" from the Tools menu and enable the flash

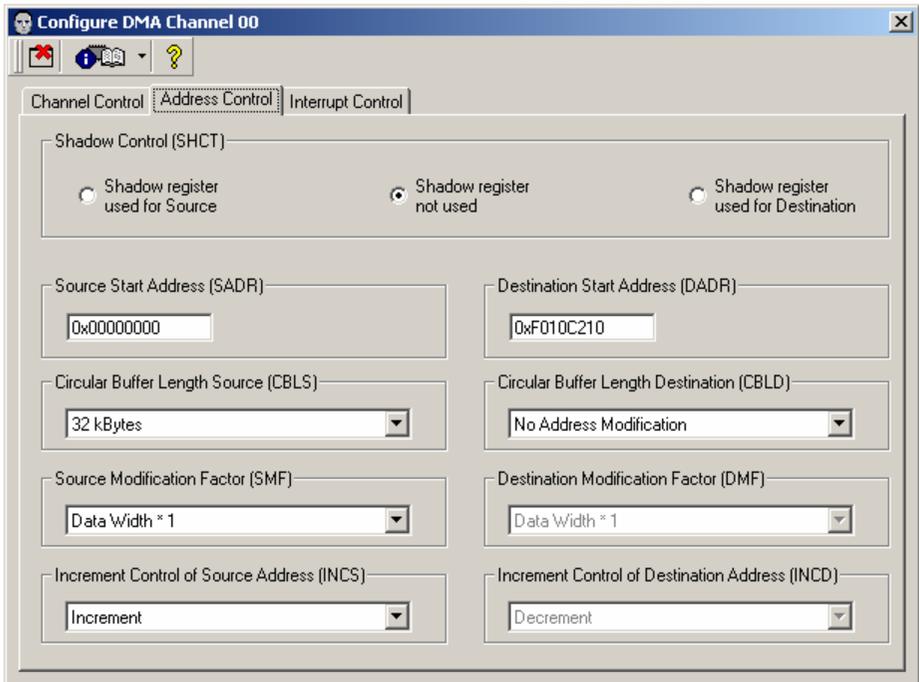
**Build a sample program**

by checking the box "Enable". Erase the Flash by pressing the "Erase..." button and exit the dialogs. Now you are ready to flash the program. Select "Load Program" from the File menu and open the 'mchk.elf' file. The UDE memory programming tool dialog pops up again. Press the "Program" button. Flashing will take up a minute. On success exit the dialogs. Reset the board and see the LED blinking.

**Add a MCHK driver to the sample program**

### 3 Add a MCHK driver to the sample program

In the following step a MCHK driver will be added to the sample program. The driver will continuously calculate a checksum over a flash memory range. Open DAVE with the 'mchk.dav' project again and select the DMA module. The MCHK module is part of the DMA<sup>1</sup>. Go to the "block 0" page. Pressing the "DMA Channel 00" button brings up the configuration dialog. Check "Use channel 00" and set the Reset Request Control (RROAT) to "Reset DMA Request Only After Transaction" and the Channel Data Width (CHDW) to "32 Bits (Word)".



**Figure 4 Configure DMA Channel 00, Page Address Control**

Go to the "Address Control" page of the "Configuration DMA Channel 00" dialog (Figure 4). Set the destination address (DADR) to the memory checker input register MCHK\_IR value 0xF010C210 (see Table 19-24 in the TC1766 User Manual). The circular buffer length source (CBLS) must exceed the memory range that will be checked. Set it to the maximum of 32kBytes which is the limit of a DMA transaction.

<sup>1</sup> The MEMCK bubble on the overview will be deleted in a later version.

### Add a MCHK driver to the sample program

Set the Increment of source Address (INCS) to "Increment". The source address (SADR) will be set by the driver.

Close the "Configuration DMA Channel 00" dialog and go to the "Memory 0" page of the "Direct Memory Access (DMA)" dialog. Check the "SCU incl. WDT, MEMCHK" and "Program Flash Space"

Go to the MEMCHK page and check the "DMA CH0" box.

At last open the "Functions" page and check the functions OCCDMA\_vlnit, OCCRC\_InitSync, OCCRC\_ResetSync and OCCRC\_CalculateSync. When generating the code two new files 'DMA.c' and 'DMA.h' will be generated.

Add these files to the Tasking project and modify the 'MAIN.c' file:

```
// USER CODE BEGIN (Main,9)
const IO_ChannelType chnl = 0;
unsigned int ret;

OCCRC_InitSync(NULL);

while(1)
{
    DIO_vTogglePin(IO_P1_0);

    OCCRC_ResetSync(0xFFFFFFFF, chnl);
    ret = OCCRC_CalculateSync((void*)0xA0000000, 1, chnl);

    if (ret == 0L)
        delay(8);
    else
        delay(1);
}
// USER CODE END
```

OCCRC\_InitSync(NULL) inits the driver, OCCRC\_ResetSync(0xFFFFFFFF, chn) saves the seed value of the channel in a global array. The seed value is the init value for the checksum calculation which will be passed to the result register MCHK\_RR before the calculation starts. The use of 0xFFFFFFFF is equal to the procedure used in the ABM checksum calculation, were the checker result register MCHK\_RR is preloaded with 0xFFFFFFFF (see TC1766 User Manual chap. 4.1.11.2).

At that time the parameter values passed by OCCRC\_CalculateSync are just placeholders. The correct values will be determined after a first build procedure. In this sample program a checksum will be calculated over the libc. After building the program, the correct values are given by the map file.

**Add a MCHK driver to the sample program**

```
* Section translation:
=====
```

```
...
Section                               | Size (MAU) | Space addr | Chip addr
-----
...
.text.libc                             | 0x000002ac | 0xa0000400 | 0x00000400
...
```

Section .text.libc starts at 0xA0000400 and has a size of 0x000002AC. 0x2AC is the size of 171 words. With the correct parameters the function call OCCRC\_CalculateSync is modified to:

```
ret = OCCRC_CalculateSync((void*)0xA0000400, 171, chnl);
```

Rebuild the elf file.

When the program is flashed now the LED will blink fast because OCCRC\_CalculateSync will not return zero so that delay(1) will be called.

To calculate the correct checksum in advance the Signature Analysis Tool (SAG) which is part of the Application Note will be used. Create a text file 'mchk.smd' and add the following line:

```
-eav 0 0xA0000400 - 0xA00006A8
```

The end address 0xA00006A8 is 4 bytes less than the sum of the start address and the size because the end address is included in the memory range.

Execute the SAG tool and read the result file 'ifxcrc.inf' from the command line

```
$ifxsag mchk.elf mchk.smd; cat ifxcrc.inf
```

```
0xd272b356
```

Put the result value in the compare condition, so that the code look like:

**Add a MCHK driver to the sample program**

```
// USER CODE BEGIN (Main,9)
const IO_ChannelType chnl = 0;
unsigned int ret;

OCCRC_InitSync(NULL);

while(1)
{
    DIO_vTogglePin(IO_P1_0);

    OCCRC_ResetSync(0xFFFFFFFF, chnl);
    ret = OCCRC_CalculateSync((void*)0xA0000400, 171, chnl);

    if (ret == 0xd272b35)
        delay(8);
    else
        delay(1);
}
```

Build, flash and run the program. The LED will blink slowly.

---

**Add an Alternate Boot Mode (ABM) header to the sample program**

## 4 Add an Alternate Boot Mode (ABM) header to the sample program

With the SAG Tool it is straightforward to add an ABM header to the elf file, so that a memory range is checked at start-up. Modify the SAG command file to:

```
-eavw 0 0xa0000400 - 0xa00006A8
```

Executing the following command

```
$mchk.elf mchk.smd -write_abm_header -flash_internal -o mchk_abm.elf
```

will add an ABM header to a copy of the elf file.

Use the e.g. the GNU readelf utility to verify the section changes.

```
$tricore-read -S mchk_abm.elf
```

...

Section Headers:

[Nr]	Name	Type	Addr	Off	Size	ES	Flg	Lk	Inf	Al
------	------	------	------	-----	------	----	-----	----	-----	----

...

[27]	.abmheader1	PROGBITS	a001ffe0	003a4c	000020	00	AX	0	0	1
------	-------------	----------	----------	--------	--------	----	----	---	---	---

[28]	.abmheader2	PROGBITS	a003ffe0	003a6c	000020	00	AX	0	0	1
------	-------------	----------	----------	--------	--------	----	----	---	---	---

Flash the program. Switch the Hardware configuration DIP 1 to OFF to select the Alternate Boot Mode. Reset the TriBoard and see again the LED blinking.

The SAG tool has many more options. To learn more about the complete functionality read the SAG manual.

## 5 Goodies

This Application Note comes with two goodies.

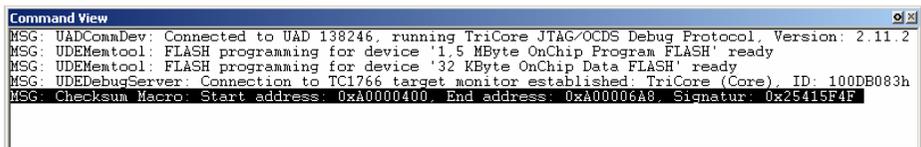
### 5.1 CRC debugger macro

Install the macro file 'ifxcrc.dsm' as an UDE debugger macro. The file contains the macro ifxcrc. If the "Run" button is pressed in the Macro dialog an input parameter pops up. The ifxcrc macro needs 3 parameters: the initial crc value, the start and the end address. Use Visual Basic notation for hexadecimal values.



**Figure 5 Input parameter dialog of the ifxcrc macro.**

The result is shown in a Message Box dialog and also written to the command view window.



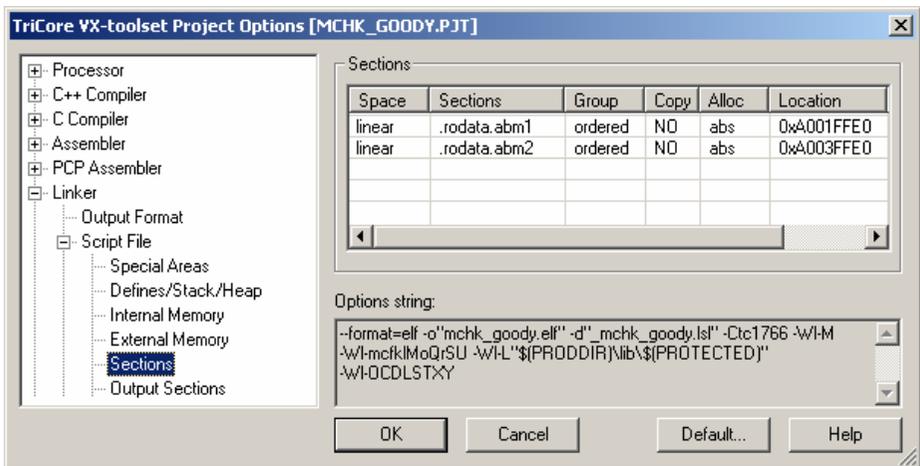
## 5.2 CRC command line tool

Use the 'ifxrc.exe' command line tool to generate an ABM header as a source file to be linked to your project. 'ifxrc.exe' comes as source code including a Makefile for GNU.

Execute

```
$ ifxrc -a -sA0000400 -eA00006A8 mchk.elf >abm1.c; cp abm1.c abm2.c
```

to create two new files 'abm1.c' and 'abm2.c'. Add both to a copy 'mchk\_goody.pjt' of the previous project file 'mchk.pjt' and setup two Sections .rodata.abm1, .rodata.abm2 (Figure 6).



**Figure 6 Linker Section Settings**

Rebuild the project. Verify the section header with the tricore-readelf utility

```
$ tricore-readelf -S mchk_goody.elf
```

```
...
Section Headers:
[Nr] Name           Type           Addr           Off           Size          ES Flg Lk  Inf Al
...
[22] .rodata.abm2 PROGBITS      a003ffe0      002c37      000020      00  AT  0   0  4
[23] .rodata.abm1 PROGBITS      a001ffe0      002c57      000020      00  AT  0   0  4
```

Flash the program and reset the TriBoard.

'ifxrc.exe' has many other options. Type --help to display all options and explore the source code.



## 6 Example code

The Application note comes with a code package ap3208510\_tc1766\_mchk.zip. The zip file contains the following files.

```

├──mchk
│   ├──abm1.c           // Generated by ifxcrc.exe
│   ├──abm2.c           // Generated by ifxcrc.exe
│   ├──CAS_DTYP.h       // Generated by DAVE mchk.dav Project
│   ├──cstart.c         // Generated by DAVE mchk.dav Project
│   ├──IFXCRC.INF       // SAG output file
│   ├──LLD.h            // Generated by DAVE mchk.dav Project
│   ├──LLD_DIO.c        // Generated by DAVE mchk.dav Project
│   ├──LLD_DIO.h        // Generated by DAVE mchk.dav Project
│   ├──LLD_DMA.c        // Generated by DAVE mchk.dav Project
│   ├──LLD_DMA.h        // Generated by DAVE mchk.dav Project
│   ├──MAIN.c           // Generated by DAVE mchk.dav Project
│   ├──MAIN.h           // Generated by DAVE mchk.dav Project
│   ├──mchk.dav         // DAVE Project file
│   ├──mchk.pjt         // Tasking EDE Project file
│   ├──mchk.psp         // Tasking EDE Project Workspace
│   ├──mchk.smd         // SAG Command file
│   ├──MCHK.WSP         // UDE Workspace
│   ├──mchk_goody.pjt   // Tasking EDE Project file
│   └──TC1766Regs.h     // Generated by DAVE mchk.dav Project
├──SAG
│   └──TriCore_SAG_Release_V1.2_2004-12-14.zip // SAG Tool package
├──Goodies
│   ├──ifxcrc
│   │   ├──ifxcrc.cpp   // ifxcrc source file
│   │   └──makefile     // Makefile for ifxcrc.exe
│   └──pls
│       └──ifxcrc.dsm    // UDE Debugger macro

```

## 7 Glossary

DAvE	Digital Application virtual Engineer. Code generator and Low level driver configurator Infineon Microcontroller. See <a href="http://www.infineon.com/dave">http://www.infineon.com/dave</a>
UDE	Universal Debug Engine from PLS
PLS	Company name: Programmierbare Logik & Systeme GmbH Manufacturer of the UDE Debugger used in this Application Note. See <a href="http://www.pls-mc.com/">http://www.pls-mc.com/</a>
Tasking	Company name: Tasking Manufacture of the Compiler used in this Application Note See <a href="http://www.tasking.com/products/tricore/">http://www.tasking.com/products/tricore/</a>
SAG	Signature Analysis Generator Tool for calculating the CRC value and attaching an ABM header to an elf file.

<http://www.infineon.com>