TC1766

TC1766 Starter Kit: "Cookery Book" for a hello world application using Altium's TASKING TriCore toolset
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 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.
Introduction:

This “Application Note” / “Appnote” is a Hands-On Training / Cookery Book / step-by-step book. It will help inexperienced users to get the TC1766 / TC176x / TC116x Family Starter Kit up and running.

With this step-by-step book you should be able to get your first useful program in less than 2 hours.

The purpose of this document is to gain know-how of the microcontroller and the tool-chain. Additionally, the "hello world example" can easily be expanded to suit your needs. You can connect either a part of - or your entire application to the TC1766 Starter Kit. You are also able to benchmark any of your algorithms to find out if the selected microcontroller fulfills all the required functions within the time frame needed.

Note:
The style used in this document focuses on working through this material as fast and easily as possible. That means there are full screenshots instead of dialog-window-screenshots; extensive use of colours and page breaks; and listed source-code is not formatted to ease copy & paste.

Have fun and enjoy TriCore!
TC1766 Block Diagram (Source: Product Marketing)
TC1766 Block Diagram (Source: Product Sheet)

Abbreviations
ICACHE: Instruction Cache
SPRAM: Scratch-Pad RAM
LDRAM: Local Data RAM
OVRAM: Overlay RAM
BROM: Boot ROM
PFLASH: Program Flash
DFLASH: Data Flash
PRAM: Parameter RAM in PCP
PCODE: Code RAM in PCP

1) Depending on the version or 8 KB of Overlay RAM will be guaranteed to be functional.
TC1766 Block Diagram (Source: User’s Manual)

Abbreviations:
ICACHE: Instruction Cache
SPRAM: Scratch- Pad RAM
LDRAM: Local Data RAM
OVRAM: Overlay RAM
BROM: Boot ROM
PFlash: Program Flash
DFlash: Data Flash
PRAM: Parameter Memory in PCP
CMEM: Code Memory in PCP
LMB: Local Memory Bus
SPB: System Peripheral Bus
Shaded: Only available in TC1766ED
Note:
Just by comparing the different sources of block diagrams, you should be able to get a complete picture of the TC1766 microcontroller and to answer some of your initial questions.
**“Cookery Book“**
For your first programming example for the TC1766 Starter Kit Board:

<table>
<thead>
<tr>
<th>Your program:</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Screenshot of terminal with program execution" /></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter/Step:</th>
<th>*** Recipes ***</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.) <strong>TC1766 Board</strong></td>
<td>Power Supply, Jumper Setting, Serial cable to the notebook, pls-Debugger</td>
</tr>
<tr>
<td>2.) <strong>DAvE – program generator</strong></td>
<td>DAvE installation (mothersystem) + DAvE Update-installation for TC1766 (DIP-file)</td>
</tr>
<tr>
<td>3.) <strong>Using DAvE</strong></td>
<td>Microcontroller initialization for your programming example</td>
</tr>
</tbody>
</table>
| 4.) **Using the TASKING Development Tools (C/C++/EC++ Compiler)** | Programming of your application with Altium’s TASKING TriCore tool chain (EDE) - v2.3r1
Locating programs into the 1,5 MByte OnChipProgramFlash (PFLASH), using OnChipSRAM) |
| 5.) **Using the pls Debugger** | Using the pls Debugger to download (program into Flash) and run your program |

**Feedback**

| 6.) **Feedback** |
1.) TC1766 Starter Kit Board:
Screenshot of the TC1766 Starter Kit homepage:

http://www.infineon.com/cms/en/product/channel.html?channel=db3a304312dc768d0112e71e62150b30
Connecting the TC1766 Starter Kit:

1. Connect a Power Supply:
The TC1766 Board requires an external power supply. A (un)regulated DC power supply from 5.5 to 60 Volts can be connected to the power connector. 500 mA are sufficient for the TC1766 Starter Kit.

2. Connect a RS-232 Serial Cable
   (1:1; 9-pin Sub-D plug – 9-pin Sub-D connector; the “Hello World” example uses this interface):

3. Connect the pls-Debugger (Flash-Programming und Debugging):

For further information, please refer to the TriBoard TC176X User’s Manual, V1.0, June 2005.
**Jumper Settings (Jumper JP501):**

Source: TriBoard TC176X User’s Manual, V1.0, June 2005

**Table 5-4 Jumper for On Board Wiggler**

*Note: The shadowed line indicates the default setting*

<table>
<thead>
<tr>
<th>Setting</th>
<th>On Board Wiggler</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 - 2</td>
<td>Enable On Board Wiggler</td>
</tr>
<tr>
<td>2 - 3</td>
<td>Disable On Board Wiggler</td>
</tr>
</tbody>
</table>

**Jumper JP501**

1-2 … Enable On-Board Wiggler *(use parallel-on-board-interface)*

2-3 … Disable On-Board Wiggler *(use pls-Debugger)*
Jumper JP501
1-2 ... Enable On-Board Wiggler (use parallel-on-board-interface)
2-3 ... Disable On-Board Wiggler (use pls-Debugger)
TC1766 Execution-Environment = OnChipFlash:

Jumper Settings (HW-Configuration DIP-Switch):

<table>
<thead>
<tr>
<th>/BRKIN</th>
<th>HWCFG[3...0]</th>
<th>Type of Boot</th>
<th>PC Start value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0000</td>
<td>Serial boot from ASC to PMI scratchpad, run loaded program</td>
<td>0xD4000000</td>
</tr>
<tr>
<td>1</td>
<td>0001</td>
<td>Serial boot from CAN to PMI scratchpad, run loaded program</td>
<td>0xD4000000</td>
</tr>
<tr>
<td>1</td>
<td>0010</td>
<td>Start from internal flash</td>
<td>0xA0000000</td>
</tr>
<tr>
<td>1</td>
<td>0011</td>
<td>Alternate Bootmode from internal flash</td>
<td>from Header or 0xD4000000</td>
</tr>
<tr>
<td>1</td>
<td>1000</td>
<td>Internal Start in EEC SRAM, if ED</td>
<td>0xAF200000</td>
</tr>
<tr>
<td>1</td>
<td>1111</td>
<td>Serial boot from ASC via CAN pins to PMI scratchpad, run loaded program</td>
<td>from Header or 0xD4000000</td>
</tr>
<tr>
<td>1</td>
<td>all others</td>
<td>reserved; don’t use this combination</td>
<td>-</td>
</tr>
<tr>
<td>0</td>
<td>0000</td>
<td>put chip in tristate (deep sleep)</td>
<td>-</td>
</tr>
<tr>
<td>0</td>
<td>all others</td>
<td>reserved; don’t use this combination</td>
<td>-</td>
</tr>
</tbody>
</table>

HW Configuration DIP-Switch:
1, 3, 4, 6 : ON
2, 5, 7, 8 : OFF
TC1766 Execution-Environment = OnChipFlash:
Accessories for the TC1766 Starter Kit: Extension Boards

“TriBoard+XC16x-Adapter-Board” to have access to all microcontroller pins. Stencils are available with the Board.

Ordering information:

Name: TriBoard+XC16x-Adapter-Platine.
The price is approximately € 32 per extension board (3 required).

Purpose: extension boards are used for easy measuring of the signals on the extension connectors to have access to all microcontroller pins and/or to connect either a part of – or the entire application to the TC1766 Starter Kit.

You can order them at:

TQ Components GmbH
Schulstraße 29a
D-82234 Weßling
Deutschland
T: +49-8153-9308-161
Mr. Rolf Müller
2.) DAvE – Installation for TC1766 microcontrollers:

Install DAvE (mothersystem):

Download the DAvE mothersystem setup.exe @ http://www.infineon.com/DAvE

<table>
<thead>
<tr>
<th>Title</th>
<th>Date</th>
<th>Version</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>DAvE - Mothersystem</td>
<td>05 Dec 2007</td>
<td>V2.1 r24</td>
<td>14.8 MB</td>
</tr>
<tr>
<td>DAvE - Mothersystem - latest version</td>
<td>04 Jul 2006</td>
<td>V2.1 r23</td>
<td>15.1 MB</td>
</tr>
</tbody>
</table>

and execute setup.exe to install DAvE.

Note:
Abort the installation of Acrobat Reader.
Install the TC1766 microcontroller support/update (TC1766 DIP file):

1.)
Download the DAve-update-file (.DIP) for the required microcontroller
@ http://www.infineon.com/DAve

<table>
<thead>
<tr>
<th>Title</th>
<th>Date</th>
<th>Version</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>TC179x Family DIP file for DAve (Microcontroller Configuration Tool)-latest version (TC179x_Series_v1.1.zip)</td>
<td>07 Jul 2008</td>
<td>v1.1</td>
<td>15 MB</td>
</tr>
<tr>
<td>TC176x DIP file for DAve (Microcontroller Configuration Tool)-latest version (TC176x_Series_v1.0.zip)</td>
<td>07 Jul 2008</td>
<td>v1.0</td>
<td>11.1 MB</td>
</tr>
<tr>
<td>TC176x family DIP file for DAve (Microcontroller Configuration Tool)-latest version (TC176x_Series.zip)</td>
<td>19 Jun 2006</td>
<td>v0.2</td>
<td>6.9 MB</td>
</tr>
</tbody>
</table>

Unzip the zip-file “TC176x_series_v1.0.zip” and save “TC176x_Series.dip”
@ e.g. D:\DAve\TC1766\TC176x_Series.dip.
2.)
Start DAvE  -  (click)

3.)
View
Setup Wizard
Default: • Installation
Forward>
Select: • I want to install products from the DAvE’s web site
Forward>
Select: D:\DAvE\TC1766
Forward>
Select: Available Products
click ✓ TC176x_Series
Forward>
Install
End

4.) DAvE is now ready to generate code for the TC1766 microcontroller.
3.) DAvE - Microcontroller Initialization after Power-On:

Start the program generator DAvE and select the TC1766 microcontroller:

File
New
32-Bit Microcontrollers
TC1766
Create
Choose the Project Settings as you can see in the screenshots:

**General: Compiler Settings:**
For the **Tasking Compiler** check/choose ☑ **Tasking** in the **Compiler Settings:**
System Clock: CPU Clock will be 80 MHz:

System Clock: **External Clock Frequency:** External clock frequency check/insert 15 [MHz]

Note:
We strongly suggest that you check first to see if your board is equipped with a 15 MHz Crystal (default).

Note:
The final result should be 80 MHz CPU Clock and 80 MHz System Clock
**Interrupt System: CPU Global Interrupt Enable:** tick ✓ Enable globally the interrupt system (IE)
PCP System: (do nothing)
Pad Driver: (do nothing)
Startup Configuration: Hardware Booting Scheme: Boot type (external pins CFG[3:0])
select Normal Internal Start. Jump to internal flash (CFG[3:0] = 0010)

Note:
Notes: If you wish, you can insert your comments here.

Exit and Save this dialog now by clicking the close button:
Configuration of the ASC0:

The configuration window/dialog can be opened by clicking the specific block/module.

Note:
ASC0 is used for the serial communication with a terminal program running on your host computer.
Module Clock: Module Disable Request: **untick**  
Disable the ASC0 module

Module Clock: Module Run Mode Clock Control: **choose** System clock/1 (=80,000 MHz)

Module Clock: Sleep Mode Enable Control: **tick**  
Disable the sleep mode
Pin Selection: Alternate Pin Selection: click Configure pins ASC0_RXD0 and ASC0_TXD0
Pin Selection: Alternate Pin Selection: Configure pins ASC0_RXD0 and ASC0_TXD0:
ASC0_RXD0: Use pin P3.0 as ASC0 receive input signal

Pin Selection: Alternate Pin Selection: Configure pins ASC0_RXD0 and ASC0_TXD0:
ASC0_RXD0: Pull Device: P3.0 pull device: select Tristate
Pin Selection: Alternate Pin Selection: Configure pins ASC0_RXD0 and ASC0_TXD0: ASC0_TXD0: ASC0_TXD0 Pin Selection: click Use pin P3.1 as ASC0 output signal

Exit and Save this dialog now by clicking the close button.
Control: Receiver Enable: tick ✓ Enable receiver (REN)
**Baud Rate**: Baud Rate: Required baud rate [kBaud] insert 9,600 < ENTER >

**Baud Rate**: Baud Rate Selection Bit / Fractional Divider: tick ✓ Use fractional divider

---

**Note:**
Validate each alpha numeric entry by pressing ENTER.
Interrupts: (do nothing)

Note: For the serial communication with a terminal program running on your host computer the myprintf function is used. The myprintf function uses Software-Polling-Mode therefore we do not need to configure any interrupts for this task.
Functions: Initialization Function: tick ✓ ASC0_vInit
Functions: Function Library (Part 1): tick ✓ ASC0_vSendData
Functions: Function Library (Part 1): tick ✓ ASCO_usGetData
Functions: Function Library (Part 1): tick ✓ ASC0_ubTxBufFree

Note:
You can change function names (e.g. ASC0_vInit) and file names (e.g. ASC0.c) anytime.
Parameters: (do nothing)

Notes: If you wish, you can insert your comments here.

Exit and Save this dialog now by clicking the close button.
Configuration of the STM:

The configuration window/dialog can be opened by clicking the specific block/module.

Note:
The LED on Port_1 Pin_0 will blink (after program start and if selected in the main menu) at a frequency of 1 second (done in the STM-Interrupt-Service-Routine). Therefore we now have to configure the STM.
Module Clock: Sleep Mode Enable Control: tick ✓ Disable the sleep mode for the STM module

Module Clock: Module Run Mode Clock Control: Clock divider for normal operation mode: select System clock / 4 ( = 50 ns)
Resolutions: (do nothing)
CMP0: Compare Register Size of CMP0: Number of bits for compare: insert 25 <ENTER>
CMP0: Compare Register 0: Required compare value (CMP0): insert 20000000 <ENTER>

Note:
20,000,000 * 50 ns = 1 s
CMP1: (do nothing)

<table>
<thead>
<tr>
<th>Module Clock</th>
<th>Resolutions</th>
<th>CMP0</th>
<th>CMP1</th>
<th>Interrupt Control</th>
<th>Interrupts</th>
<th>Functions</th>
<th>Parameters</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start Bit Location of CMP1</td>
<td>Lowest bit number (0 to 24) of STM which is compared with the content of register CMP1 bit 0 (MSTART1)</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Compare Register Size of CMP1</td>
<td>Number of bits (1 to 32) in register CMP1 (starting from 0), which are used for the compare operation with STM (MSIZE1)</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Compare Register 1</td>
<td>Required compare value (CMP1)</td>
<td>0x00000000</td>
<td>Real compare value (dependent of MSIZE1)</td>
<td>0x00000000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

TC1676 (new project)
**Interrupt Control:** Compare Register CMP0 Interrupt Control:
- **tick ✓** Enable request on compare match with CMP0

**Interrupt Control:** Interrupt Control of STMIR0: **tick ✓** Enable SRC0 interrupt

![System Timer (STM) Interface](image)
Interrupts: drag and drop STM SRN 0 from Level 0 to CPU Interrupt: Level 9

Note:
The LED on Port_1 Pin_0 will blink (after program start and if selected in the main menu) at a frequency of 1 second (done in the STM-Interrupt-Service-Routine STM_viSRN0).
Functions: Initialization Function: tick ✓ STM_vInit

Note: The LED on Port_1 Pin_0 will blink (after program start and if selected in the main menu) at a frequency of 1 second (done in the STM-Interrupt-Service-Routine STM_viSRN0).
Parameters: (do nothing)

Notes: If you wish, you can insert your comments here.

Exit and Save this dialog now by clicking the close button.
Port Configuration:

The configuration window/dialog can be opened by clicking the specific block/module.

Note: The User LED (orange) is connected to Port_1 Pin_0.
Ports: click Configure Port 1
Ports: Configure Port 1:
Port 1: Functionality: tick ✓ Use P1.0 as general IO, General Direction: click ☐ Out
Input Characteristic: (do nothing)
Output Characteristic: (do nothing)
Parameters: (do nothing)

Notes: If you wish, you can insert your comments here.

Exit and Save this dialog now by clicking the close button.
Functions: Initialization Function: tick ✓ IO_vInit
Functions: Function Library (Part 1): tick ✓ IO_vSetPin
Functions: Function Library (Part 1): tick ✓ IO_vResetPin
Functions: Function Library (Part 1): tick ✓ IO_vTogglePin
Parameters: (do nothing)

Notes: If you wish, you can insert your comments here.

Exit and Save this dialog now by clicking the close button.
Save the project:

File
Save

Save project: Save in C:\TC1766 [create new directory]
File name: TC1766 (2)
Generate Code:

<table>
<thead>
<tr>
<th>File Generate Code</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

or click

DAvE will show you all the files he has generated (File Viewer opens automatically).
Close DAvE:

File

Exit

Save changes?

click Yes
4.) Using the TASKING - EDE Development Tools:

Write programs for execution from **OnChipFlash (PFLASH)**

Abbreviations:
- ICACHE: Instruction Cache
- SRAM: Scratch-pad RAM
- LDRAM: Local Data RAM
- OVRAM: Overlay RAM
- BRAM: Boot ROM
- PFlash: Program Flash
- DFlash: Data Flash
- PRAM: Parameter RAM in PCP
- PCODE: Code RAM in PCP

1) Depending on the version 0 or 8 KB of Overlay RAM will be guaranteed to be functional
Install the Tasking Development Tools TriCore v2.3r1

Start Tasking EDE, select directory and include the DAvE Files:

If you see an open project – close it: File – Close Project Space
File - Change Directory…
Select a Directory: choose C:\TC1766

Select a Directory

C:\TC1766

- Documents and Settings
- DRIVERS
- FILES
- flexim
- Garmin
- help
- INST
- Program Files
- Programs
- TC1766
- TC1766-001-HelloWorld
- TC1766-002-HelloWorld

OK
File - New Project Space…
Create a New Project Space: Filename: insert TC1766
Click: “Add new project to project space”
Add New Project to Project Space: Filename: insert TC1766

Add New Project to Project Space

OK
Click: “Add existing files to project”
Select One or More Files to Add to Project: select ASC0.c
Select One or More Files to Add to Project: select ASC0.h
Select One or More Files to Add to Project: select IO.c
Select One or More Files to Add to Project: select IO.h
Select One or More Files to Add to Project: select MAIN.c
Select One or More Files to Add to Project: select MAIN.h
Select One or More Files to Add to Project: select STM.c
Select One or More Files to Add to Project: select STM.h
Select One or More Files to Add to Project: select TC1766Regs.h
OK
Configure Compiler, Assembler, Linker, Locator and Build – Control:

**Project – Project Options**

**Processor:** Processor Definition: Target processor: select TC1766B

![TriCore VX-toolset Project Options](image)
Processor: Bypasses: CPU Functional Problem
Bypasses:
tick ✓ All bypasses TC1762/TC1764/TC1766

Note: The system startup code (libstart.asm) must have been added to your project.
Processor: Startup: (do nothing)
Processor: Startup: Startup Code: (do nothing)
Processor: Startup: Startup Code: Trap Vectors: (do nothing)
C Compiler: Preprocessing: deactivate (click to untick)  □ Automatic inclusion of .sfr file
C Compiler: Language: (do nothing)
C Compiler: Debug Information: (do nothing)
C Compiler: Code Generation: (do nothing)
C Compiler: Optimization: Optimization level: select No optimization
C Compiler: Allocation: (do nothing)
C Compiler: Warnings: (do nothing)
C Compiler: MISRA-C: (do nothing)
C Compiler: Miscellaneous: (do nothing)
Linker: (do nothing)
Linker: Output Format: (do nothing)
Linker: Script File: (do nothing)
Linker: Script File: Special Areas: RESET start address: insert 0xA0000000 (PFLASH)
Linker: Script File: Special Areas: Libraries start address: insert 0xA0080000 (PFLASH)
Linker: Script File: Special Areas: Interrupt table start address: insert 0xA0100000 (PFLASH)
Linker: Script File: Special Areas: Trap table start address: insert 0xA0102000 (PFLASH)

Linker: Script File: Special Areas: CSA start address: insert/check 0xD0000000 (LDRAM)
Additional information: Program Memory / Data Memory:

The On Chip PMU_PFLASH memory has a capacity of 1.504 KBytes:
The On Chip DMI_LDRAM memory has a capacity of 56 KBytes.
Additional information: Interrupt Vector Table:

Remember:
**Additional information:** Interrupt Vector Table:

Interrupt Vector Table:

<table>
<thead>
<tr>
<th>PN</th>
<th>Interrupt</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>255</td>
<td>0x010.1FFF</td>
<td>256 * 8 Words</td>
</tr>
<tr>
<td>9</td>
<td>0x010.0120</td>
<td>256 * 32 Bytes</td>
</tr>
<tr>
<td>1</td>
<td>0x010.0020</td>
<td>8192 Bytes</td>
</tr>
<tr>
<td>0</td>
<td>0x010.0000</td>
<td>0x2000 Bytes</td>
</tr>
</tbody>
</table>

Note:
PN … Priority Number (CPU Interrupt Level)

Note:
[Click here to see the Map File](#)
Additional information: TRAP Vector Table:

TRAP Vector Table:

<table>
<thead>
<tr>
<th>Trap Vector Table</th>
<th>Class_7</th>
<th>Class_6</th>
<th>Class_5</th>
<th>Class_4</th>
<th>Class_3</th>
<th>Class_2</th>
<th>Class_1</th>
<th>Class_0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xA010.20FF</td>
<td>0xA010.20E0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0xA010.20C0</td>
<td>0xA010.20A0</td>
<td>0xA010.2080</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0xA010.2060</td>
<td>0xA010.2040</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0xA010.2020</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note:

1 Word = 32 Bits
1 Word = 4 Bytes
8 Words = 32 Bytes

Note:

Click here to see the Map File
The on-chip \texttt{PMU\_PFLASH} memory has a capacity of 1.504 KBytes:
Additional information: Program Memory (Source: User’s Manual):

The on-chip PMU_PFLASH memory has a capacity of 1.504 KBytes:

<table>
<thead>
<tr>
<th>Numbering</th>
<th>Size</th>
<th>Cached Address Range</th>
<th>Non-Cached Address Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>PFLASH Bank</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PB</td>
<td>1504 Kbyte</td>
<td>8000 0000_H - 8017 7FFF_H</td>
<td>A000 0000_H - A017 7FFF_H</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PFLASH Sectors</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>PS0</td>
<td>16 Kbyte</td>
<td>8000 0000_H - 8000 3FFF_H</td>
<td>A000 0000_H - A000 3FFF_H</td>
</tr>
<tr>
<td>PS1</td>
<td>16 Kbyte</td>
<td>8000 4000_H - 8000 7FFF_H</td>
<td>A000 4000_H - A000 7FFF_H</td>
</tr>
<tr>
<td>PS2</td>
<td>16 Kbyte</td>
<td>8000 8000_H - 8000 BFFF_H</td>
<td>A000 8000_H - A000 BFFF_H</td>
</tr>
<tr>
<td>PS3</td>
<td>16 Kbyte</td>
<td>8000 C000_H - 8000 FFFF_H</td>
<td>A000 C000_H - A000 FFFF_H</td>
</tr>
<tr>
<td>PS4</td>
<td>16 Kbyte</td>
<td>8001 0000_H - 8001 3FFF_H</td>
<td>A001 0000_H - A001 3FFF_H</td>
</tr>
<tr>
<td>PS5</td>
<td>16 Kbyte</td>
<td>8001 4000_H - 8001 7FFF_H</td>
<td>A001 4000_H - A001 7FFF_H</td>
</tr>
<tr>
<td>PS6</td>
<td>16 Kbyte</td>
<td>8001 8000_H - 8001 BFFF_H</td>
<td>A001 8000_H - A001 BFFF_H</td>
</tr>
<tr>
<td>PS7</td>
<td>16 Kbyte</td>
<td>8001 C000_H - 8001 FFFF_H</td>
<td>A001 C000_H - A001 FFFF_H</td>
</tr>
<tr>
<td>PS8</td>
<td>128 Kbyte</td>
<td>8002 0000_H - 8003 FFFF_H</td>
<td>A002 0000_H - A003 FFFF_H</td>
</tr>
<tr>
<td>PS9</td>
<td>256 Kbyte</td>
<td>8004 0000_H - 8007 FFFF_H</td>
<td>A004 0000_H - A007 FFFF_H</td>
</tr>
<tr>
<td>PS10</td>
<td>512 Kbyte</td>
<td>8008 0000_H - 800F FFFF_H</td>
<td>A008 0000_H - A00F FFFF_H</td>
</tr>
<tr>
<td>PS11</td>
<td>480 Kbyte</td>
<td>8010 0000_H - 8017 7FFF_H</td>
<td>A010 0000_H - A017 7FFF_H</td>
</tr>
</tbody>
</table>
Additional information: Program Memory (Source: User’s Manual):

The on-chip PMU_PFLASH memory has a capacity of 1.504 KBytes:
Additional information: Program Memory (Source: User’s Manual):

The on-chip PMU_PFLASH memory has a capacity of 1.504 KBytes:

Note:
There is a typing error in Table 8-2, page 8-6, TC1766 User’s Manual, System Units (Vol. 1 of 2). The correct address range for the 1.504 Kbyte PMU_PFLASH is A000 0000H - A017 7FFFH.
Linker: Script File: Defines/Stack/Heap: (do nothing)
Linker: Script File: Internal Memory: change from brom to PMU_BROM
Linker: Script File: Internal Memory: change from ovram to PMU_OVRAM
Linker: Script File: Internal Memory: change from ldram to DMI_LDRAM
Linker: Script File: Internal Memory: change from spram to PMI_SPRAM
Linker: Script File: Internal Memory: change from pram to PCP_PRAM
Linker: Script File: Internal Memory: change from pcode to PCP_CMEM
Linker: Script File: Internal Memory: Name: insert PMU_PFLASH
Linker: Script File: Internal Memory: Alloc: select ON
Linker: Script File: Internal Memory: Type: select ROM
Linker: Script File: Internal Memory: Size: insert 1504k
Linker: Script File: Internal Memory: Address insert 0xA0000000
Linker: Script File: Internal Memory: Name=PCP_PRAM: Alloc: select OFF
Linker: Script File: Internal Memory: Name=PCP_CMEM: Alloc: select OFF
Additional information: Memory (Source: User’s Manual):

### Abbreviations:
- **ICACHE**: Instruction Cache
- **SPRAM**: Scratch-Pad RAM
- **LDRAM**: Local Data RAM
- **O'VRAM**: Overlay RAM
- **BROM**: Boot ROM
- **PFlash**: Program Flash
- **DFlash**: Data Flash
- **PRAM**: Parameter Memory in PCP
- **CMEM**: Code Memory in PCP
- **LMB**: Local Memory Bus
- **SPB**: System Peripheral Bus
- **Shaded**: Only available in TC1766ED
Linker: Script File: External Memory: (do nothing)
Linker: Script File: Sections: (do nothing)
Linker: Script File: Output Sections: (do nothing)
Linker: Script File: Reserved: (do nothing)
Linker: Map File: tick ✓ Memory usage info

Note:
Click here to see Memory usage info
Linker: Script File: Libraries: (do nothing)
Linker: Script File: Optimization: (do nothing)

Options string:
```
-ow -O2 -Wall -g -DCOMMON_H -DCURRENT_H -DTC1766 -DTC1766_BUG=844 -O0 -L"$PRODDIR\lib" -Wl,-noundefined
```

![TriCore VX-toolset Project Options](image)

OK Cancel Default... Help
Linker: Script File: Warnings: (do nothing)
Linker: Script File: Miscellaneous: (do nothing)

OK
Insert your application specific program:

Note:
DAvE doesn’t change code which is inserted between ‘// USER CODE BEGIN’ and ‘// USER CODE END’. Therefore, whenever adding code to DAvE’s generated code, write it between ‘// USER CODE BEGIN’ and ‘// USER CODE END’.
If you wish to change DAvE’s generated code or add code outside these ‘USER CODE’ sections, you will have to insert/modify your changes each time after letting DAvE regenerate code!
Double click: Main.c insert User Code (Global Variables):

```c
const char menu[] =
"\n\n\n"
"TC1766, Program execution out of OnChipFlash:\n"
"============================================\n"
"1 ... LED IO_Port_1_Pin_0 ON\n"
"2 ... LED IO_Port_1_Pin_0 OFF\n"
"3 ... LED IO_Port_1_Pin_0 blinking\n"
"    \n";
const char question[] =
"your choice: ";
const char message1[] =
"\n\r*** LED is ON ***\r\n";
const char message2[] =
"\n\r*** LED is OFF ***\r\n";
const char message3[] =
"\n\r*** LED is BLINKING ***\r\n";
volatile int RS232_wait=2;
volatile unsigned int blinking=ON;
char select= ' ';
```
// Tasking Variables

using namespace TPDU;

// Define global variables

// constants

// variables

// main function

int main(int argc, char* argv[]) {
    int constant = 42;
    // code
    return 0;
}

// function declarations

// function definitions

// end of program

// TRIPLET Tasking IDE for Infineon TriCore VX-toolset
Double click: Main.c insert User Code (function: input()):

```c
char input (void)
{
    char in=' '; 
    do 
    {
        myprintf(question);
        // ASC0_RSRC_SRR ... ASC0_ReceiveInterrupt Service Request Control Register_Service Request Flag
        // ASC0_RSRC_CLRR ... ASC0_ReceiveInterrupt Service Request Control Register_Request Clear Bit
        while (!ASC0_RSRC_SRR) ;
        ASC0_RSRC_CLRR=1; // Clear SRR bit
        in = (unsigned char)ASC0_RBUF;
    }while (in!='1' && in!='2' && in!='3');
    return in;
}
```

![Image of code editor interface]
Double click: `Main.c` insert User Code:

```c
while(RS232_wait);
while (1)
{
    myprintf(menu);
    select=input();

    switch (select)
    {
        case '1': blinking=OFF, IO_P1_0=LED_ON, myprintf(message1); break;
        case '2': blinking=OFF, IO_P1_0=LED_OFF, myprintf(message2); break;
        case '3': blinking=ON, myprintf(message3); break;
    }
}
```
Double click: Main.h and insert the following Defines:

```c
#define OFF 0  
#define ON 1    
#define LED_ON 0 
#define LED_OFF 1
```
Double click: Main.h and insert Global Variables:

extern volatile unsigned int blinking;
extern volatile int RS232_wait;
Double click: STM.c insert User Code for interrupt service routine:

```c
STM_CMP0=STM_CMP0+20000000; // 20.000.000 * 50 ns = 1 s

if(RS232_wait)
    RS232_wait--;

if(blinking)
    IO_vTogglePin(IO_P1_0);
```

**Note:**

20.000.000 * 50 ns = 1 s

To get an STM interrupt every 1 second you must change the Compare Value to “STM_CMP0+=20000000;”!
Reason for “myprintf.c“

Unfortunately, a low-level I/O implementation similar to example project “IO” (which consists of “serio.c” and “serio.h” files for generating an output stream for “printf” using ASC0) using tool chain C166/ST10 is currently not available for Tasking TriCore tools.
For the moment, Tasking has only got the following “Change Request”:

CR32186 CR: Example for _write function implementation using serial interface.

DESCRIPTION
Change request for a low-level I/O (_write function implementation) example which does not use simulated I/O but uses the real serial interface of the controller.

EXAMPLE

WORKAROUND
File – New
Open a New File for Editing: Filename: insert myprintf.c

OK
Insert User Code for myprintf():

```c
#include "main.h"
#include "ASC0.H"

void myprintf(const char *p)
{
    while(*p)
    {
        if (ASC0_ubTxBufFree())
            ASC0_vSendData(*p++);
    }
}

// Example 1 (use of myprintf):
void main(void)
{
    myprintf("Hello World!
\n");
}

// Example 2 (use of myprintf):
void main(void)
{
    char mb[200]; // message buffer for sprintf()
    int dummy;

    sprintf(mb,"Variable wait = %d",dummy); // Write formatted data to string mb
    myprintf(mb);
}
/*
// Example 1 (use of myprintf):
void main(void)
{
    myprintf("Hello World!
\n");
}

// Example 2 (use of myprintf):
void main(void)
{
    char mb[200]; // message buffer for sprintf()
    int dummy;

    sprintf(mb,"Variable wait = %d",dummy); // Write formatted data to string mb
    myprintf(mb);
}
*/
```
```c
#include "main.h"
#include "ASC0.H"

void sprintf(const char *p|)
{
  while(*p)
  {
    if (ASC0_vBufferFree())
      ASC0_vSendData(*p--);
  }
}
/
 // Example 1 (use of sprintf): void main(void)
 {    printf("Hello World\n");
 // Example 2 (use of sprintf): void main(void)
 { printf("Variable width = %d",dummy); // Write formatted data to string mb  sprintf(mb,"Variable width = %d",dummy);   
 

File - Save all

(Project Window File View) – TC1766 (Files) – right mouse button click – Add Existing Files – Browse
Select myprintf.c

Open - OK
```c
#include "main.h"
#include "ASC0.h"

void myprint(const char *p)
{
    while(*p)
        if (ASC0_putcharFree())
            ASC0_putcharData(*p--);

    /*
     * Example 1 (use of myprint):
     * myprint("Hello World\n");
     */
    while(*p)
    {
        ASC0_putcharData(*p--);
    }
    ASC0_putcharData('\n');

    /*
     * Example 2 (use of myprint):
     * void main(void)
     * {
     *     char ob[200]; // message buffer for sprintf()
     *     int dummy;
     *     sprintf(ob,"Variable wait = %d",dummy); // Write formatted data to string ob
     *     myprint(ob);
     * }
     */
```
Double click: Main.h and insert Prototypes of Global Functions:

```c
extern void myprintf(const char *p);
```
Double click: Main.h and insert required Header for sprintf:

```c
#include <stdio.h>  // for sprintf (for myprintf)
```
Generate your application program:

**Build - Rebuild**

or

![Screenshot of TASKING EDE interface]

```c
MAIN() {

    // USER CODE BEGIN (Main,1)
    while (PSS2.wait()) {
        ...
        myprintf(message); select=select(input); switch {select} {
            case '1': blinking=OFF, IO_PID=LED_OK, myprintf(message); break;
            case '2': blinking=OFF, IO_PID=LED_OFF, myprintf(message); break;
            case '3': blinking=OFF, myprintf(message); break;
        }
        // USER CODE END
        return(pmSecCur); // End of function main
    }

    // Compiling and assembling "stm.c"
    // Compiling and assembling "asc0.c"
    // ctc USR2: ["asc0.c"] 262/10] possible truncation at implicit conversion to type "short"
    // 0 errors, 1 warnings
    // Assembling "cstart.asm"
    // Compiling and assembling "myprintf.c"
    // Linking and locating to TC1766rlk in ELF/MIYR 2 format
}
```
Insert Map File:

(Project Window File View) – TC1766 (Files) – right mouse button click – Add Existing Files – Browse

Select TC1766.map

Open - OK
See Map File:

Interrupt Vector Table:

Note:

Click here to see Memory Map
Trap Vector Table:

<table>
<thead>
<tr>
<th>Chip</th>
<th>Scope</th>
<th>Section</th>
<th>Size (Bd)</th>
<th>Space addr</th>
<th>Chip addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>apic</td>
<td>MPU_FFLASH</td>
<td>.text.libc.reset</td>
<td>0x00000000</td>
<td>0x00000000</td>
<td>0x00000000</td>
</tr>
<tr>
<td>apic</td>
<td>MPU_FFLASH</td>
<td>.data.libc</td>
<td>0x00000004</td>
<td>0x00000000</td>
<td>0x00000000</td>
</tr>
<tr>
<td>apic</td>
<td>MPU_FFLASH</td>
<td>.rodata.libc</td>
<td>0x00000004</td>
<td>0x00000000</td>
<td>0x00000000</td>
</tr>
<tr>
<td>apic</td>
<td>MPU_FFLASH</td>
<td>.text.xsoc.ASCI_text</td>
<td>0x0000000c</td>
<td>0x00000000</td>
<td>0x00000000</td>
</tr>
<tr>
<td>apic</td>
<td>MPU_FFLASH</td>
<td>.text.xsoc.ASCI_data</td>
<td>0x00000008</td>
<td>0x00000000</td>
<td>0x00000000</td>
</tr>
<tr>
<td>apic</td>
<td>MPU_FFLASH</td>
<td>.text.xsoc.ASCI_uninit</td>
<td>0x00000008</td>
<td>0x00000000</td>
<td>0x00000000</td>
</tr>
<tr>
<td>apic</td>
<td>MPU_FFLASH</td>
<td>.text.wx.GDT</td>
<td>0x00000012</td>
<td>0x00000000</td>
<td>0x00000000</td>
</tr>
<tr>
<td>apic</td>
<td>MPU_FFLASH</td>
<td>.text.libc_esx_main</td>
<td>0x00000008</td>
<td>0x00000000</td>
<td>0x00000000</td>
</tr>
<tr>
<td>apic</td>
<td>MPU_FFLASH</td>
<td>.text.main.MAIL_VVisit</td>
<td>0x0000001c</td>
<td>0x00000000</td>
<td>0x00000000</td>
</tr>
<tr>
<td>apic</td>
<td>MPU_FFLASH</td>
<td>.text.main.MAIL_VIF</td>
<td>0x00000006</td>
<td>0x00000000</td>
<td>0x00000000</td>
</tr>
<tr>
<td>apic</td>
<td>MPU_FFLASH</td>
<td>.text.main.input</td>
<td>0x00000002</td>
<td>0x00000000</td>
<td>0x00000000</td>
</tr>
<tr>
<td>apic</td>
<td>MPU_FFLASH</td>
<td>.text.main.input2</td>
<td>0x00000002</td>
<td>0x00000000</td>
<td>0x00000000</td>
</tr>
<tr>
<td>apic</td>
<td>MPU_FFLASH</td>
<td>.text.main.write</td>
<td>0x00000002</td>
<td>0x00000000</td>
<td>0x00000000</td>
</tr>
<tr>
<td>apic</td>
<td>MPU_FFLASH</td>
<td>.text.main_write2</td>
<td>0x00000002</td>
<td>0x00000000</td>
<td>0x00000000</td>
</tr>
<tr>
<td>apic</td>
<td>MPU_FFLASH</td>
<td>.text.xsoc_.STB_VVisit</td>
<td>0x00000008</td>
<td>0x00000000</td>
<td>0x00000000</td>
</tr>
<tr>
<td>apic</td>
<td>MPU_FFLASH</td>
<td>.text.xsoc_.STB_VIF</td>
<td>0x00000006</td>
<td>0x00000000</td>
<td>0x00000000</td>
</tr>
<tr>
<td>apic</td>
<td>MPU_FFLASH</td>
<td>.text.xsoc_.STB_input</td>
<td>0x00000002</td>
<td>0x00000000</td>
<td>0x00000000</td>
</tr>
<tr>
<td>apic</td>
<td>MPU_FFLASH</td>
<td>.text.xsoc_.STB_input2</td>
<td>0x00000002</td>
<td>0x00000000</td>
<td>0x00000000</td>
</tr>
<tr>
<td>apic</td>
<td>MPU_FFLASH</td>
<td>.text.xsoc_.STB_write</td>
<td>0x00000002</td>
<td>0x00000000</td>
<td>0x00000000</td>
</tr>
<tr>
<td>apic</td>
<td>MPU_FFLASH</td>
<td>.text.xsoc_.STB_write2</td>
<td>0x00000002</td>
<td>0x00000000</td>
<td>0x00000000</td>
</tr>
<tr>
<td>apic</td>
<td>MPU_FFLASH</td>
<td>.text.xsoc_.STB_read</td>
<td>0x00000002</td>
<td>0x00000000</td>
<td>0x00000000</td>
</tr>
<tr>
<td>apic</td>
<td>MPU_FFLASH</td>
<td>.text.xsoc_.STB_read2</td>
<td>0x00000002</td>
<td>0x00000000</td>
<td>0x00000000</td>
</tr>
</tbody>
</table>

Note:
Click here to see Memory Map
Memory Usage:
Now you can close your project and Tasking EDE:

File - Close Project Space
File - Exit
5.) Programming is now complete. You can now load and run your program:

Start pls-Debugger

![UDE Desktop](image)
File – New Workspace

Open
Click OK

Press Default
Create or use default: ☑ Use a default target configuration: expand Debugger
Create or use default: ☑ Use a default target configuration:
select Triboard with TC1766 (JTAG/OCDS)

Click Finish
New Target Configuration: Save in: select C:\TC1766 (1)
New Target Configuration: File name: change/insert TC1766 (2)

Save
TC1766 “Cookery Book” for a hello world application

Application Note 144 V2.0, 2008-10
Config – Add-In Components
UDE Add-In Components Load State:
UDE Add-In Component Description check/tick ✓ FLASH/OTP Memory Programming Tool

OK
Tools – FLASH Programming …

- PDF Viewer Window
- Macros...
- Manage Macro Storage...
- Run selected Macro
- Break Macro
- Reload selected Macro

FLASH Programming ...

Execution Time Setup

Open FLASH programming interface
FLASH/OTP – Memory Device: check/select 1,5 MByte OnChip Program FLASH
FLASH/OTP – Memory Device: check/tick ✓ Enable
Click Setup …
Program: **tick ✓** Automatic Verify after Program

**OK**

**Exit**
File – Load Program
Open program file: Look in: select TC1766
Open program file: File name: select tc1766.elf
Click Program All
Exit
Exit
File – Close Workspace

Yes

File – Exit
Execute any terminal program
(9600 Baud, 8 bit Data, no Parity-Bit, 1 Stop-Bit, Xon/Xoff Protocol):

Power-On the Board and see the result:
Conclusion:

In this step-by-step book you have learned how to use the TC1766 Starter Kit together with the Tasking tool chain.

Now you can easily expand your “hello world” program to suit your needs!

You can connect either a part of - or your entire application to the TC1766 Starter Kit.

You are also able to benchmark any of your algorithms to find out if the selected microcontroller fulfils all the required functions within the time frame needed.

Have fun and enjoy working with the TC1766 Starter Kit!

Note:

There are step-by-step books for 8 bit microcontrollers (e.g. XC866, XC88x, and XC878), 16 bit microcontrollers (e.g. C16x, XC16x, and XE16x) and 32 bit microcontrollers (e.g. TC1796 and TC1130).

All these step-by-step books use the same microcontroller resources and the same example code.

This means: configuration steps, function names, and variable names are identical.

This should give you a good opportunity to get in touch with another Infineon microcontroller family or tool chain!

There are even more programming examples using the same style available [e.g. ADC examples, CAPCOM6 examples (e.g. BLDC-Motor, playing music), Simulator examples, C++ examples] based on these step-by-step books.
6.) Feedback (TC1766): Your opinions, suggestions and/or criticisms

Contact Details (this section may remain blank should you wish to offer feedback anonymously):

If you have any suggestions please send this sheet back to:

Email: mcdocu.comments@infineon.com
FAX: +43 (0) 4242 3020 5783

Your suggestions:

________________________________________________________________________
________________________________________________________________________
________________________________________________________________________
________________________________________________________________________
________________________________________________________________________
________________________________________________________________________
________________________________________________________________________
________________________________________________________________________