Application Note, V3.1, June 2006

AP32082

TC176x

TC176x Examples Collection

Microcontrollers

Never stop thinking

Edition 2006-06-16

Published by
Infineon Technologies AG
81726 München, Germany

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.

AP32082		
Revision History:	2006-06	V3.1
Previous Version:	V3.0	

Page	Subjects (major changes since last revision)
6	Removed 'whetstone' example
7	Changed the name of the installation directory from `TC176x` to `TC176x_examples` (cap. 3).
7	Added entry `cstart.asm` to the directory structure (cap 3).
9	Changed the description of the Interrupt System configuration (cap. 4.4.1).
10,11	Changed Altium Tasking version from 2.2r3 to 2.3r1.
11	Reformulated chapter 5.1.2.
11	Changed the name of the installation directory from `TC176x` to `TC176x_examples` (cap. 5.1.3).
12	Changed the description of the Liker/Locator configuration (cap. 5.2).

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 Page

1 Introduction

The TC176x Examples Collection is a collection of software examples aimed to guide users to get a quick start to work with microcontrollers belonging to the Infineon Technologies TriCore TC176x family[1].

The TC176x Examples Collection shows the main steps the user has to go through in order to have a full-functional application. In particular configuration and usage of Clock and Interrupt Systems, System Timer (STM), Ports and Peripheral Input/Output (GPIO) and Asynchronous/Synchronous Serial Interface (ASC) are addressed.

All software examples are self-contained applications in form of Tasking projects, developed for the TC176x TriBoard. Therefore the TC176x Examples Collection can also be used to learn how to setup a Tasking project for the TriCore TC176x and the TC176x TriBoard.

All the information contained in this document is retrieved from the TC176x official documentation and the TC176x TriBoard User's Manual. Reference should always be made to these documents.

In this document we will use following abbreviations:

- 'Tasking' for the Altium Tasking VX-Toolset for TriCore toolchain (C compiler, assembler, linker).
- 'CrossView' for the Altium Tasking CrossView Pro debugger.
- 'EDE' for the Tasking Embedded Development Environment.

2 TC176x Examples Collection Content

The TC176x Examples Collection contains 5 software examples, showing the usage of some modules of the TriCore TC176x (e.g. Clock and Interrupt Systems, System Timer, GPIO, and ASC).

All software examples are self-contained applications in form of Tasking projects, developed for the TC176x TriBoard.

Here below a description of the available examples.

2.1 led_pol

This example makes the TriBoard led blink, by using a polling algorithm based on the System Timer (STM). The blinking period is set to 1 second, with a duty cycle of 50%.

This example shows how to configure and use the TriCore STM module.

2.2 led_int

This example makes the TriBoard led blink, by using a System Timer (STM) interrupt. The blinking period is set to 1 second, with a duty cycle of 50%.

This example shows how to configure and use the TriCore STM module.

2.3 rs232_o

This example simply writes a message on a dumb terminal (e.g. Windows 2000 HyperTerminal).

This example shows how to configure and use the TriCore ASC module.

[1] The term "TriCore TC176x family" denotes a set of code compatible microcontroller derivatives, all of them named TC176 plus a derivative number x. An example is the TriCore TC1766.

2.4 tc_clock

This example implements a running clock using the System Timer. A dumb terminal (e.g. Windows 2000 HyperTerminal) via the TriBoard RS232 interface is used to display the running clock.

This example shows how to realize a clock function with the TriCore on-board STM and ASC modules.

2.5 queens

This example is a demonstration of the Eight Queens Problem. It shows all possible ways in which eight queens can be placed on an 8x8 chessboard without threatening each other. It finishes when it has found all 92 solutions to the problem. Each solution is displayed as a chess diagram on a VT100 terminal (e.g. Windows 2000 HyperTerminal) via the TriBoard RS232 interface.

This example is a translation/adaptation of the original Pascal version in "Algorithms+Data Structures=Programs", Niklaus Wirth, Prentice-Hall, 1976.

This example shows how to port a standard algorithm on the TriCore as well as how to implement the VT100 protocol on the TriCore ASC module.

3 TC176x Examples Collection Organisation

The TC176x Examples Collection is organized in several subdirectories, one subdirectory for each example. An additional subdirectory, `common_src`, contains sources which may be used by all examples.

Each example directory has a `src` subdirectory containing the C sources of the example and a `make_t` subdirectory containing all files related to the Tasking toolchain.

The resulting directory structure is therefore:

```
TC176x_examples
   |
   |_ common_src\
   |     |_ comm.c
   |     |_ comm.h
   |     |_ init.c
   |     |_ init.h
   |     |_ reg176x.h
   |     |_ types.h
   |     |_ util.c
   |     |_ util.h
   |
   |_ <example_x>\
   |     |
   |     |_ make_t\
   |     |     |_ cstart.asm
   |     |     |_ <example_x>_tc176<n>.opt
   |     |     |_ <example_x>_tc176x.pjt
   |     |
   |     |_ src\
   |           |_ <example_x>.c
   |
   |_ examples_tc176x.psp
   |_ readme.txt
```

where:

Directory or File	Description
common_src	This directory contains sources which may be used by all of the examples.
comm.c, comm.h	Serial communication routines.
init.c, init.h	Initialisation functions for Clock, Interrupt Systems, GPIO and STM.
reg176x.h	TC176x registers definitions.
types.h	Basic types.
util.c, util.h	Common definitions, macros and functions for the TC176x core.
<example_x>	Example-specific directory. There is one directory for each example.
make_t	This directory contains all files related to the Tasking toolchain. Please note that after compilation, much more files are to be found in this directory.
cstart.asm	Tasking TriCore start-up code.
<example_x>_tc176<n>.opt	Tasking option file for a specific TriCore derivative. There will be one option file for each supported derivative.
<example_x>_tc176x.pjt	Tasking project file.
src	This directory contains the example specific C source code file.
<example_x>.c	Example-specific C source code file.
examples_tc176x.psp	Tasking project space file.
readme.txt	Readme file.

4 TC176x Examples Collection Implementation

All definitions, initialisation functions, general purpose functions, as well as functions used by more than one example, are to be found in the source files stored under directory `common_src` (i.e. files `types.h` `reg176x.h`, `util.h`, `util.c`, `init.h`, `init.c`, `comm.h`, `comm.c`).

Each example is implemented in one or more dedicated sources, stored in the `src` subdirectory of the example-specific directory.

However the whole collection has been developed using the Tasking toolchain, particular attention has been given to portability issues, so that the user should be able to easily port the whole code to other toolchains.

4.1 File `types.h`

This C header file contains the definition of some basic types used in the examples.

4.1.1 Integer Data Types

For integer variables following types are defined:

Data Type	Data Size
`uint8_t`	8 bit, unsigned
`int8_t`	8 bit, signed
`uint16_t`	16 bit, unsigned
`int16_t`	16 bit, signed
`uint32_t`	32 bit, unsigned
`int32_t`	32 bit, signed
`uint64_t`	64 bit, unsigned
`int64_t`	64 bit, signed

4.2 File `reg176x.h`

This C header file contains the definition of all TC176x registers used in the examples.

Each register definition has the form:

```
#define <NAME> (*((uint32_t volatile *) <ADDRESS>))
```

where `<NAME>` is the register name and `<ADDRESS>` the register 32-bit address, in hexadecimal notation.

4.3 Files `util.c` and `util.h`

These files contain common definitions, macros and functions for the TC176x core.

4.3.1 Intrinsic Functions

The examples make use of some TriCore C intrinsic functions. For portability reasons, these functions are not used directly as they are defined in the Tasking C compiler, but they are redefined through `#define` statements in header file `util.h`. In particular following instruction are to be found:

`DISABLE` = Disable Interrupts

`ENABLE` = Enable Interrupts

`MTCR` = Move to Core Register

`MFCR` = Move from Core Register

`ISYNC` = Insert ISYNC instruction

4.3.2 EndInit Bit Management

EndInit protected registers can be unlocked and locked by using the functions `ClearEndinit()` and `SetEndinit()` respectively.

Function `ClearEndinit()` clears the EndInit bit, which controls access to system critical registers. Clearing the EndInit bit unlocks all EndInit protected registers. Modifications of the EndInit bit are monitored by the watchdog timer such that after clearing EndInit, the watchdog timer enters a defined time-out mode. EndInit must be set again before the time-out expires.

Function `SetEndinit()` sets the EndInit. Setting the EndInit bit locks all EndInit protected registers.

4.4 Files `init.c` and `init.h`

These files contain initialisation functions and values for almost all TC176x modules used in the examples.

4.4.1 System Clock and Interrupt System

Function `InitSystem()` initialises the Clock and Interrupt Systems.

The Clock System is configured in PLL mode, i.e. the CPU clock is derived from the oscillator clock, divided by the input divider P, multiplied by the feedback divider N and divided by the output divider K:

$$f_{CPU} = \frac{N}{P \cdot K} \cdot f_{OSC}$$

For this TriCore family the CPU Clock to System Clock ration is always 1:1.

Assuming an external (i.e. on the TC176x TriBoard) 15 MHz oscillator, P = 1, N = 40 and K = 10, we will have a CPU and System Clock frequency of 60 MHz.

The Interrupt System is configured with two arbitration cycles (max. 15 interrupt sources) and two clocks per arbitration cycle.

4.4.2 System Timer

Function `InitSTM()` does a minimal configuration of the STM module. It simply configures the STM Clock equal to the System Clock.

Function `InitSTMCmp(uint32_t CmpTime)`, beside configuring the STM Clock equal to the System Clock, configures also the STM capture match interrupt control logic. Compare register 0 is here used. Parameter `CmpTime` specifies the time interval, in ms, between two successive interrupts. Constant `STM_ISR_PRIO` specifies the STM ISR priority number.

4.4.3 GPIO

Function `InitTriboardLed()` initialises the microcontroller port pin connected to the TriBoard led, i.e. pin 0 of port 1 (P1.0). This pin is configured as a GPIO push/pull output pin, medium driver, and is set to high level.

4.5 Files `comm.c` and `comm.h`

These files contain functions and values for the RS232 serial communication via the TriCore ASC module.

4.5.1 ASC0 Configuration

Function `InitRS232(uint32_t BaudRate)` configures the ASC0 module to operate in asynchronous mode, with 8 data bit, 1 stop bit, no parity check, no framing check. The receiver is disabled. The baud rate

generator is programmed according to parameter `BaudRate` (e.g. 9600 Baud). The ASC0 clock is set equal to the System Clock.

The serial output of ASC0 is connected to GPIO pin P3.1 (i.e. pin 1 of port 3), which in turn is configured as an output pin in Alternate Mode 1 (ALT1) and set to low level.

The communication protocol is interrupt driven, where constant `RS232_TX_INT` specifies the ASC0 transmission ISR priority number.

4.5.2 Communication Management

Function `_interrupt(RS232_TX_INT) RS232TxISR()` is the ASC0 transmission Interrupt Service Routine. Constant `RS232_TX_INT` specifies its priority number.

Function `RS232Write(char *pchMsg)` writes the string pointed by parameter `*pchMsg` on the ASC0 serial output.

Function `VT100InitScreen()` initializes the VT100 terminal. It writes a VT100 escape sequence on the ASC0 serial output for clearing the screen, switching off inverse mode, turning off the cursor and setting its position at the upper left corner of the screen.

Function `VT100ExitScreen()` restores some terminal settings. It writes a VT100 escape sequence on the ASC0 serial output for changing to the next line, switching off inverse mode and turning the cursor on again.

Function `VT100SetCursor(uint16_t u16Line, uint16_t u16Col)` writes a VT100 escape sequence on the ASC0 serial output for setting the cursor position on the coordinates specified by parameters `u16Line` and `u16Col`.

Function `VT100PrintNormal (const char *pchStr)` writes the string pointed by parameter `*pchStr` on the ASC0 serial output, in normal video mode.

Function `VT100PrintReversed (const char *pchStr)` writes the string pointed by parameter `*pchStr` on the ASC0 serial output, in reversed video mode.

4.6 Example-Specific Sources

Each example is implemented in one or more dedicated sources, stored in the `src` subdirectory of the example-specific directory.

We don't give here a description of these files. Please refer to the source code itself, which is generously commented.

4.7 System Header Files

Following system header files, provided with the Tasking C compiler are used:

Header File	Purpose
`ctri.h`	TriCore intrinsic functions
`stdio.h`	I/O functions
`math.h`	Arithmetic functions

4.8 Portability

The TC176x Examples Collection has been developed using the Tasking toolchain. Nevertheless, a great effort has been spent to reduce to the minimum the dependencies to this specific toolchain. Here below the remaining critical points.

System header file `ctri.h` contains the definition of the TriCore C intrinsic functions. It is toolchain-specific and therefore may have different names in other toolchains.

In header file `util.h` all used TriCore C intrinsic functions are redefined by macros. These definitions may need to be modified in other toolchain. Please do this accordingly to chapter "Intrinsic functions"

5 TC176x Examples Collection Usage

The TC176x Examples Collection was developed with the Altium Tasking VX-Toolset for TriCore ver. 2.3r1 toolchain, including the Altium Tasking CrossView Pro ver. 2.3r1.

Target hardware is the Infineon Technologies TC176x TriBoard rev. 300, with a 15 MHz on-board oscillator.

Windows 2000 is assumed on the host side.

5.1 Configuring the Board, Building and Running the Code

Windows 2000 is assumed on the host side. Although in this document the `c:` drive is used, the TC176x Examples Collection is not drive-specific.

5.1.1 Setups

Verify that the TriBoard on-board oscillator (on 14-pin DIP) socket is marked with 15 MHz.

Verify that you have an Altium Tasking VX-Toolset for TriCore ver. 2.3r1 toolchain installed, CrossView debugger included.

Check that the TriBoard DIP switches are set as follows:

- SW1 = ON
- SW2 = OFF
- SW3 = ON
- SW4 = ON
- SW5 = OFF
- SW6 = ON
- SW7 = OFF
- SW8 = OFF

Connect the TriBoard BD9 connector to a "dumb terminal" like Windows 2000 HyperTerminal, configured as follows:

- COM1
- 9600 Baud
- 8 data bits
- 1 stop bit
- No parity
- Hardware flow control
- VT100 emulation

5.1.2 Extract the Sources

Run the installer `ap3208231_tc176x_examples_collection.exe`. This will create the directory structure described in chapter "TC176x Examples Collection Organisation".

5.1.3 Build the Code

In the installation directory `TC176x_examples` you can find the already prepared Tasking project space file `examples_tc176x.psp`. By double clicking on this file, the Tasking EDE will start and automatically load a full-configured environment, containing all examples in form of projects. The related project files (`<example_x>_tc176x.pjt`) are to be found in the example-specific `make_t` subdirectories.

In same cases (e.g. if you are using an evaluation version of the Tasking toolchain) when loading a new project the Tasking EDE may rise a message windows claiming that the loaded project was developed for

another toolchain version. Please quit this message window by clicking on the `OK` button. As the result a dialog box listing the available toolchains is shown. Please select the toolchain you want to use and then click on the `OK` button. In the next windows select `Continue`. Now your project has been converted for the toolchain version currently in use.

The examples are implemented to run on different derivatives of the TriCore TC176x family. For each supported derivative, in each example-specific `make_t` subdirectory there is a Tasking option file (`<example_x>_tc176<n>.opt`). By loading one of these option files in the Tasking EDE, all the settings for the related target microcontroller are taken on.

In the Tasking EDE, the main steps to go through in order to bring a specific example to run on the desired target (i.e. desired TriCore TC176x derivative) are:

- Select the example you want to compile and set it as the current project (`Project -> Set Current`).
- Load the option file related to your specific target (e.g. `<example_x>_tc1766.opt` for the TriCore TC1766) in order to correctly setup up the environment (`Project -> Load Options`).
- Compile the example (`Build -> Rebuild`).
- As the result an executable file (`<example_x>_tc176x.elf`) is produced in the example-specific `make_t` subdirectory.

5.1.4 Load and Run the Executables

The Altium Tasking CrossView Pro used for downloading the code to the TC176x TriBoard.

You can start CrossView directly form Tasking EDE. Doing this, CrossView is automatically configured for the TC176x device and the executable of the example being the current project is automatically downloaded on the target.

You can also start CrossView as a standalone program. In this case you need to select manually the target as well as the executable you want to download.

Once in CrossView, you can directly start the execution, set breakpoints, step through the code, and so one.

Please refer to the CrossView documentation for more details.

5.2 Linker/Locator Configuration

All examples are linked at address `0xD4000000`, i.e. the starting address of internal Code Scratch-Pad RAM.

http://www.infineon.com

Published by Infineon Technologies AG