TriCore

AP32178
cstart

Application Note
V1.0 2011-08

Microcontrollers
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 the 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 the 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
Is there any information in this document that you feel is wrong, unclear or missing?
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

1  Preface ......................................................................................................................... 5
2  Introduction ................................................................................................................... 5
3  Overview ....................................................................................................................... 5
4  Implementation and Usage ............................................................................................. 9
1 Preface

This application note describes a user’s startup code implementation on the TriCore processor architecture [1,2] for the AUDO MAX-family. The document is aimed at developers who write or design applications for the TriCore.

This application note assumes that readers have access to the TriCore Architecture Manual [1] and TriCore User Manual [3-6], and have at least some knowledge about the following sections of the user’s manual:

- Startup SoftWare (SSW) (see section BootROM content in [3-5])
- Clock system of the System Control Unit (SCU) (see section Clock System overview in [3-5])
- ENDINIT protection and watchdog timer (WDT) (see section Watchdog Timer in [3-5])
- The TriCore instruction set.

See References on Page 13 for more information on the TriCore manuals and other relevant documentation.

2 Introduction

Compilers for the TriCore processor are available by third party Infineon tool partners and offers user’s startup code with their tool chain. It is provided as C source code or assembler source code. The source file for the user’s startup code named cstart.c for Tasking, crt0.S for Hightec and crt0.s for Wind River. This application note is written explicitly for Tasking users. It improves and extends the default Tasking startup file cstart.c in four ways. First it improves the PLL initialization and implements a program flow exactly as described in the user’s manual. Second it extends the number of registers that could be configured in the startup code. It especially offers configuration for most ENDINIT protected registers. ENDINIT bit protected register are typically needed to be configured only once at startup. Grouping them together makes it possible to clear and set the ENDINIT bit only once. This practice saves execution time which is often critical at startup. An endinit_clear()/endinit_set() programming sequence typically requires about 0.5 μs running at 180 MHz CPU frequency. Third a fast ENDINIT bit clear and set routines are offered as inline functions. Fourth the cstart.h header files comes with PLL initialization values for most popular configurations of the TriCore AUDO-MAX family. To limit the jump of the dynamic current consumption the PLL initialization uses a ramp-up sequence.

Together these modifications of the default Tasking startup code give the user a quick start programming the TriCore. With entering the C main() function the processor is already running at the configured CPU frequency and configured modules frequencies.

3 Overview

The PLL uses two different start-up mechanisms depending on the triggering reset. Upon a power-on reset the PLL starts to supply the system in Preccsaler Mode. The starting frequency is 16.6 MHz. A system reset brings the PLL control register in the SCU to the defined reset values and the system clock operates in free-running mode at $f_{VCORE}$/16. In both cases the SSW in the BootROM restores the clock system to free-running mode before jumping to the user’s startup code located at the User STArtup ADDress STADD. Tasking named this address the RESET vector. Two addresses are valid: 0xA0000000 for starting from internal flash memory module (internal start) or 0xA1000000 for starting from external EBU space (external start). The SSW therefore evaluates the HWCFG[7..0] pins. For external start the EBU reads its configuration parameter from internal memory 0000040, (see section External Bus Unit in [7], chapter ‘Boot Process’ respectively ‘Configuration Word Fetch Process’).

The major design goal of the user’s startup code is to initialize the processor and to bring up the PLL quickly, to configure major CSFR and other ENDINIT protected SFR registers. The steps are illustrated in Figure 1. The changes made to the original code are mainly related to the PLL ramp-up sequence and the ability to configure more ENDINIT protected sfr registers. The execution time on a TC1798 running at 300 MHz CPU frequency of the startup code is about 250-350 μs, where the largest single part (230 μs) is the ramp-up sequence using six steps with a delay in between two steps of 20 μs. Details of the PLL ramp-up sequence are illustrated in Figure 2. A block diagram of the Clock Generation Unit (CGU) is shown in Figure 3. The current consumption during a PLL ramp-up sequence with just four steps is shown in Figure 4. Formulas for the dynamic current consumption are given in the data sheet.

The internal Watchdog starts after reset in Time-Out Mode. With the startup code presented by this application node the watchdog would enter Prewarning Mode after $4 \times f_{FIR}/16384$ which is measured to 950 μs. The
execution time of the startup code as configured in this application note is less than 300 μs. To save time the watchdog is serviced and the ENDINIT bit is set after all ENDINIT protected registers are configured.

Figure 1  Startup code Flow Diagram
Figure 2  PLL initialization Flow Diagram (TC1798 300MHz)
Figure 3  CGU and CCU

Figure 4  Current consumption during frequency Ramp-up sequence.
4 Implementation and Usage

The implementation follows the default Tasking startup file cstart.c but modifies or adds certain parts as explained in section 3. The cstart.c and cstart.h files that come with this application notes replaces the Tasking C startup files.

The new cstart.h header file offers popular configurations for the AUDO-MAX TriBoards.

- TC172x 80/132 MHz
- TC178x 132/180 MHz
- TC179x 240/270/300 MHz

To enable one of these configuration the control program cctc should be called with option __<TriCore Derivative>__ and __fCPU=<frequency[MHz]>, for example –D__TC1798__ -D__fCPU=300. Select Project > Properties and navigate to C/++ Build > Settings > C/C++ Compiler > Preprocessing and add these symbols to the list of defined symbols (Figure 5).

These macros will select the appropriate settings in cstart.h. Listing 1 shows this configuration for the TC1798 running at 300MHz.

More changes to the cstart.h are also reflected by more options in the cstart editor within the Tasking EDE. Figure 6 for example shows the register page with ADC, CAN, GPTA, SCU, SSC and STM registers.

Details of the configurations are listed in Table 1 to Table 3.

![Add Preprocessor symbols](image)

| 239 | `#elif (defined __TC1798__ || defined __TC1793__ || defined __TC1791__)` |
| 240 | `#if __fCPU==300` |
| 241 | `// fPLL=600MHz` |
| 242 | `// fPCP=200MHz` |
Listing 1  PLL specific configuration in cstart.h. TC1798 with fCPU=300Mhz shown

```c
#define __SCU_PLLCON0_INIT      1
#define __SCU_PLLCON0_VALUE     0x1017600
#define __SCU_PLLCON1_INIT      1
#define __SCU_PLLCON1_VALUE     0x0
#define __SCU_PLLK2RAMPUP_INIT  1
#define __SCU_PLLK2RAMPUP_VALUE 0x08040201
#define __SCU_PLLK2RAMPUP_WAIT  6000
#define __SCU_CCUCON0_INIT      1
#define __SCU_CCUCON0_VALUE     0x2030105
#define __SCU_CCUCON1_INIT      1
#define __SCU_CCUCON1_VALUE     0x30B03
#define __SCU_CCUCON2_INIT      1
#define __SCU_CCUCON2_VALUE     0x701
#define __SCU_FDR_INIT          1
#define __SCU_FDR_VALUE         0x43FE
#define __FLASH0_FCON_INIT      1
#define __FLASH0_FCON_VALUE     0x00074804
#define __FLASH1_FCON_INIT      1
#define __FLASH1_FCON_VALUE     0x00074804
```

Figure 6  start editor: Register page
### Table 1  TC179x PLL configuration examples

<table>
<thead>
<tr>
<th>Parameter</th>
<th>TC179x 300MHz</th>
<th>TC179x 270MHz</th>
<th>TC179x 240MHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clock Diver Option</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>$f_{OSC}$</td>
<td>20 MHz</td>
<td>20 MHz</td>
<td>20 MHz</td>
</tr>
<tr>
<td>$f_{OCOBASE}$</td>
<td>200 MHz</td>
<td>200 MHz</td>
<td>200 MHz</td>
</tr>
<tr>
<td>PLLCON0.PDIV</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>PLLCON0.NDIV</td>
<td>0x3B</td>
<td>0x35</td>
<td>0x2F</td>
</tr>
<tr>
<td>PLLCON1.K1DIV</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>PLLCON1.K2DIV</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>CCUCON0.PCPDIV</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>CCUCON0.FSIDIV</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>CCUCON0.SRIDIV</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CCUCON0.FPIDIV</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>CCUCON1.EDBBBDIV</td>
<td>0xB</td>
<td>0xB</td>
<td>0xB</td>
</tr>
<tr>
<td>CCUCON1.MCDSDIV</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>CCUCON2.EBUDIV</td>
<td>7</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>FDR.STEP</td>
<td>0x3FE</td>
<td>0x3FE</td>
<td>0x3FF</td>
</tr>
</tbody>
</table>

$$f_{VCO} = f_{OSC} \times \frac{(NDIV+1)}{(PDIV+1)}$$  
$$f_{PLL} = f_{OSC} \times \frac{(NDIV+1)}{((PDIV+1) \times (K2DIV+1))}$$  
$$f_{PCP} = \frac{f_{PLL}}{(PCPDIV+1)}$$  
$$f_{FSI} = \frac{f_{PLL}}{(FSIDIV+1)}$$  
$$f_{SRI} = \frac{f_{PLL}}{(SRIDIV+1)}$$  
$$f_{FPI} = \frac{f_{PLL}}{(FPIDIV+1)}$$  
$$f_{EDBBB} = \frac{f_{PLL}}{(EDBBBDIV+1)}$$  
$$f_{REFCLK} = \frac{f_{PLL}}{2(REFCLKDIV+1)}$$  
$$f_{MCDS} = \frac{f_{PLL}}{(MCSDIV+1)}$$  
$$f_{EBU} = \frac{f_{PLL}}{(EBUDIV+1)}$$  
$$f_{ERAY} = \frac{f_{PLL}}{(ERAYDIV+1)}$$  
$$f_{OUT} = \frac{f_{FPI}}{1/(0x400-STEP)}$$  

PLL ramp up sequence  
6 steps: 20, 66.7, 120, 200, 300, 600 MHz  
6 steps: 20, 67.5, 135, 180, 270, 540 MHz  
5 steps: 20, 68.6, 120, 240, 480 MHz
## Table 2 TC178x PLL configuration examples

<table>
<thead>
<tr>
<th>Parameter</th>
<th>TC178x 180MHz</th>
<th>TC178x 132MHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clock Diver Option</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>( f_{OSC} )</td>
<td>20 MHz</td>
<td>20 MHz</td>
</tr>
<tr>
<td>( f_{VCOBASE} )</td>
<td>200 MHz</td>
<td>200 MHz</td>
</tr>
<tr>
<td>PLLCON0.PDIV</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>PLLCON0.NDIV</td>
<td>0x47</td>
<td>0x41</td>
</tr>
<tr>
<td>PLLCON1.K1DIV</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>PLLCON1.K2DIV</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>CCUCON0.PCPDIV</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>CCUCON0.LMBDIV</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>CCUCON0.FPIDIV</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CCUCON1.REFCLKDIV</td>
<td>0xB</td>
<td>0xB</td>
</tr>
<tr>
<td>CCUCON1.MCDSDIV</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>FDR.STEP</td>
<td>0x3FE</td>
<td>0x3FF</td>
</tr>
</tbody>
</table>

\[
f_{VCO} = f_{OSC} \times (NDIV+1)/(PDIV+1)
\]
\[
f_{PLL} = f_{OSC} \times (NDIV+1)/((PDIV+1)\times(K2DIV+1))
\]
\[
f_{PCP} = f_{PLL} / (PCPDIV+1)
\]
\[
f_{LMB} = f_{PLL} / (LMBDIV+1)
\]
\[
f_{FPI} = f_{PLL} / (FPIDIV+1)
\]
\[
f_{REFCLK} = f_{PLL} / 2/(REFCLKDIV+1)
\]
\[
f_{MCDS} = f_{PLL} / (MCDSDIV+1)
\]
\[
f_{OUT} = f_{FPI} \times \frac{1}{(0x400-\text{STEP})}
\]

PLL ramp up sequence

<table>
<thead>
<tr>
<th></th>
<th>3 steps: 20, 120, 180 MHz</th>
<th>3 steps: 20, 110, 132 MHz</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3 steps:</td>
<td></td>
</tr>
</tbody>
</table>
Table 3  TC172x PLL configuration examples

<table>
<thead>
<tr>
<th>Parameter</th>
<th>TC172x 132MHz</th>
<th>TC172x 80MHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clock Diver Option</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>$f_{OSC}$</td>
<td>20 MHz</td>
<td>20 MHz</td>
</tr>
<tr>
<td>$f_{VCOBASE}$</td>
<td>200 MHz</td>
<td>200 MHz</td>
</tr>
<tr>
<td>PLLCON0.PDIV</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>PLLCON0.NDIV</td>
<td>0x41</td>
<td>0x3F</td>
</tr>
<tr>
<td>PLLCON1.K1DIV</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>PLLCON1.K2DIV</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>CCUCON0.PCPDIV</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>CCUCON0.LMBDIV</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>CCUCON0.FPIDIV</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>CCUCON1.REFCLKDIV</td>
<td>0xB</td>
<td>0xB</td>
</tr>
<tr>
<td>CCUCON1.MCDSDIV</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CCUCON2.ERAYDIV</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>FDR.STEP</td>
<td>0x3FF</td>
<td>0x3FF</td>
</tr>
</tbody>
</table>

$f_{VCO} = f_{OSC} \times (NDIV+1)/(PDIV+1)$: 660 MHz 640 MHz

$f_{PLL} = f_{OSC} \times (NDIV+1)/(PDIV+1)*(K2DIV+1)$: 132 MHz 80 MHz

$f_{PCP} = f_{PLL} / (PCPDIV+1)$: 132 MHz 80 MHz

$f_{LMB} = f_{PLL} / (LMBDIV+1)$: 132 MHz 80 MHz

$f_{FPI} = f_{PLL} / (FPIDIV+1)$: 66 MHz 80 MHz

$f_{REFCLK} = f_{PLL} / 2 / (REFCLKDIV+1)$: 5.5 MHz 3.33 MHz

$f_{MCDS} = f_{PLL} / (MCDSDIV+1)$: 66 MHz 40 MHz

$f_{ERAY} = f_{PLL} / (ERAYDIV+1)$: 66 MHz 40 MHz

$f_{OUT} = f_{FPI} \times (1 / (0x400\text{-STEP}))$: 33 MHz 40 MHz

PLL ramp up sequence

- 3 steps: 20, 110, 132 MHz
- 2 steps: 20, 80 MHz

5 References

[1] TriCore Architecture V1.3.8 2007-11, Infineon Technologies AG