TC1130
"Cookery-Book" for a "Hello world" application using Tasking and pls Tools
Introduction:

This “Appnote” is an Infineon Hands-On-Training.
It will help inexperienced users to get an TC1130 Evaluation-Board / Starter-Kit-Board up and running.

With this Hands-On-Training / Cookery-Book / 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 your needs.
You can connect either a part of - or your entire application to the Starter-Kit-Board.
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.

The main chapters are:
- Chapter 4: Program_Execution_From_Code-Scratch-Pad-RAM (PMI_SPRAM)
- Chapter 5: Program_Execution_From_OnBoardProgramFlash
- Chapter 6: Program_Execution_From_OnBoardProgramFlash_Burst-Mode

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!
Programming Examples

TC1130

Starter Kit

infineon
TC1130 Block Diagram (Source: Product Marketing)
TC1130 Block Diagram (Source: code generator DAvE)
For your first programming example for the TC1130 Starter-Kit-Board:

<table>
<thead>
<tr>
<th>Chapter/Step:</th>
<th>*** Recipes ***</th>
</tr>
</thead>
</table>
| 1. | **TC1130 Board**  
Power Supply, Jumper Setting, Serial cable to the notebook, pls-Debugger |
| 2. | **DAvE – program generator**  
DAvE installation (mothersystem) + DAvE Update-installation for TC1130 (DIP-file) |
| 3. | **Using DAvE**  
Microcontroller initialization for your programming example |
| 4. | **Using the TASKING Development Tools (C/C++/EC++ Compiler)**  
Programming of your application with TASKING (Altium) tool chain (EDF) - v2.3r1  
Locating programs into the 32 KBytes code scratchpad RAM, (PMI_SPRAM), using OnChipSRAM) |
| 5. | **Using the TASKING Development Tools (C/C++/EC++ Compiler)**  
Programming of your application with TASKING (Altium) tool chain (EDF) - v2.3r1  
Locating programs into the 32 MBytes OnBoardFlash, using OnChipSRAM + OnBoardSDRAM) |
| 6. | **Using the TASKING Development Tools (C/C++/EC++ Compiler)**  
Programming of your application with TASKING (Altium) tool chain (EDF) - v2.3r1  
Locating programs into the 32 MBytes OnBoardFlash (Burst-Mode!), using OnChipSRAM (28 KBytes DMI_SRAM + 64 KBytes DMU SRAM) + 64 MBytes OnBoardSDRAM |
**Additional exercises**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>7.)</td>
<td>Time – Measurement <em>(Using an oscilloscope / a logic analyzer)</em></td>
</tr>
<tr>
<td>8.)</td>
<td>Time – Measurement <em>(Using the SystemTimer (STM))</em></td>
</tr>
</tbody>
</table>

**Feedback**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>9.)</td>
<td>Feedback</td>
</tr>
</tbody>
</table>
1.) TC1130 Starter Kit Board:
Ordering information:

http://www.infineon.com/cgi-bin/ifx/portal/ep/programView.do?channelId=-65591&programId=34747&programPage=%2Fep%2Fprogram%2Finformation.jsp&pageTypeId=17099&BV

Screenshot of the TC1130 Starter-Kit Homepage:

SK-TCA130 Starter Kit

MCU Derivates:
- SAF-TCA130-L100E, SAF-TCA130-L100E
- SAF-TCA1718-L100E, SAF-TCA1718-L100E
- SAF-TCA1700-L100E, SAF-TCA1700-L100E

CPU Clock: 100/150 MHz

On-Chip Memory:
- 28 KBytes data memory (SPRAM)
- 32 KBytes code memory (SPRAM)
- 16 KBytes instruction cache (ICACHE)
- 4 KBytes data cache (DCACHE)
- 64 KBytes SRAM Data Memory Unit (CMU)
- 16 KBytes boot ROM

On-Boards Memory:
- Burst FLASH up to 64 MBytes (default: 32 MBytes),
- PC100 SDRAM 2 banks up to 64 MBytes per bank (default: 1 banks with 64 MBytes) or Micron SyncFlash 2 banks with 16 MBytes per bank

Interfaces:
- DB9 for RS232-0
BERG10 for RS232-1
- two BERG10 for CAN-0/1 with Transceiver
- BERG10 for ODCS1
- SAMTEC GSH-030-01-F-D-A for ODCS2
- RJ45 Connector with LED’s for Ethernet
- USB connector (type B) for USB connections
- DB25 On Board Wiggler for ODCS1
- four 80-pin connectors (male) with all I/O signals
- four 80-pin connectors (female) with all I/O signals

Includings
- TC1130 User manuals (pdf)
- TriCore Architecture manual (pdf)
- TriBoard manual (pdf)
- Tools: Compiler, Debugger, Operating System from Infineon Tool Partners (Evaluation versions)
- Cable
- 1 Extension Board
- Power plug

Order Nr.: B1NS8-H8559-X-X-7600
Price*: 400,- EUR

How to order?
Buy Online

or please contact your local distributor: http://www.infineon.com/distribution

Documentation

<table>
<thead>
<tr>
<th>Description</th>
<th>File Name</th>
<th>Size</th>
<th>Date</th>
<th>Revision</th>
</tr>
</thead>
<tbody>
<tr>
<td>TriBoard TC1130 - Hardware Manual</td>
<td>TriBoardManual-TC1130-V12.pdf</td>
<td>1.5 MB</td>
<td>1 Jul 2005</td>
<td></td>
</tr>
</tbody>
</table>
Connecting the TC1130-Board to the Environment:

Connect a Power Supply:
The TC1130 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.
500mA are sufficient for the TC1130 Starterkit.

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):

Connect the pls-Debugger (used for: On-Board-Flash-Programming and Debugging):

For further information, please refer to the TriBoard TC1130 V1.2, June 2004 Hardware Manual.
**Jumper Settings (Jumper JP501):**

**Note:**
We use the pls-Debugger in this document.

**Jumper JP501**
1-2 … Enable On Board Wiggler (use parallel-on-board-interface)
2-3 … Disable On Board Wiggler (use pls-Debugger)
TC1130-Execution-Environment-SPRAM:

Note:
At the beginning we use the default configuration of Jumper S301.

**Jumper Settings (HW-Configuration DIP-Switch S301)**

**S301:**
1, 2, 4, 5, 6, 7, 8 : OFF (default)
3 : ON (default)
Accessories for the TC1130 Starter Kit – Extension Boards

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

Ordering information:

Name: TriBoard+XC16x-Adapter-Platine
Price: approximately € 32,- apiece, (4 required)

Purpose:
Extension Boards are used to measure the signals on the extension connectors (see next page) and/or to connect the Starter-Kit-Board with your application.

You can order them at:

TQ Components GmbH
Schulstraße 29a
D-82234 Weßling
Deutschland
T: +49-8153-9308-161
Mr. Rolf Müller
Connecting the Extension Boards to the Starter-Kit / Using stencils
2.) DAve – Installation for TC1130 microcontrollers:

Install DAve:

Download @ [http://www.infineon.com/DAve](http://www.infineon.com/DAve) the DAve-mothersystem setup.exe

<table>
<thead>
<tr>
<th>Tool Package</th>
<th>Date</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>DAve - Mothersystem - latest version</td>
<td>06 Feb 2007</td>
<td>14.8 MB</td>
</tr>
<tr>
<td>DAve - Mothersystem</td>
<td>04 Jul 2008</td>
<td>15.1 MB</td>
</tr>
</tbody>
</table>

and execute setup.exe to install DAve.
Install the TC1130 microcontroller Update:

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

Unzip the zip-file TC1130_v1.5.zip and save “TC1130_v1.5.dip” @ e.g. D:\DAvE\TC1130-2007-04-02\TC1130_v1.5.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\TC1130-2007-04-02
Forward>
Select: Available Products
click ✓ TC1130
Forward>
Install
End

4.) DAvE is now ready to generate code for the TC1130 microcontroller.
3.) DAvE - Microcontroller Initialization after Power-On:

Start the program generator DAvE and select the TC1130 microcontroller:

File
New
32-Bit Microcontrollers
TC1130
Create
Choose the Project Settings as you can see in the Screenshots:

**General: Compiler Settings:**
For the **Tasking Compiler** choose/check  ☑ Tasking in the **Compiler Settings:**
System Clock: CPU Clock will be 150 MHz:

System Clock: External Clock Frequency: External clock frequency insert 20 [MHz]

Yes

Note:
We would be very grateful if you checked that your board is equipped with a 20 MHz Crystal (default).

Note:
Validate each alpha numeric entry by pressing ENTER.
System Clock: External Clock Frequency: Input divider (PDIV) select fp=fosc/2=10,000 MHz

Yes
System Clock: Voltage Controlled Oscillator:
Feedback divider (NDIV) select \( f_{vco} = f_{osc} / P \times 60 = 600,000 \text{ MHz} \)

Yes
System Clock: Output Divider: Output divider (KDIV) select $f_{cpu} = f_{vco}/4 = 150,000$ MHz

Yes
Note:

The final result should be 150 MHz CPU frequency and 75 MHz system frequency.
Interrupt System: CPU Global Interrupt Enable: click ✓ Enable globally the interrupt system (IE)
Startup Configuration: Hardware Booting Scheme: (do not care !!!)

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

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

The configuration window can be opened by clicking the specific block/module.
Module Clock: Module Disable Request: select/check  
Disable the ASC0 module
Module Clock: Module Run Mode Clock Control: choose  System clock/1 (=75,0000 MHz)
Module Clock: Sleep Mode Enable Control: click  
✓ Disable the sleep mode
Control: Mode Control (M): click 8-bit data (asynchronous)
Control: Transmit Pin Selection: click TxPin (P2.1) selected
Control: Receive Pin Selection: click RxPin (P2.0) selected
Control: Receiver Enable: click Enable receiver (REN)
Baud Rate: Baud Rate: Required baud rate [kBaud] insert 9,600
Baud Rate: Baud Rate Selection Bit / Fractional Divider: click ✓ Use fractional divider
Baud Rate: Baud Rate Generator Run Control (R): click ✓ Enable baud rate generator

FIFO: (do nothing)
IrDA: (do nothing)
Interrupts: (do nothing)
Functions: Initialization Function: click ✓ ASC0_vInit
Functions: Function Library (Part 1): click ✓ ASC0_vSendData
Functions: Function Library (Part 1): click ✓ ASC0_usGetData
Functions: Function Library (Part 1): click ✓ ASC0_ubTxBufFree

Parameters: (do nothing)
Notes: If you wish, you can insert your comments here.

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

The configuration window can be opened by clicking the specific block/module.
**Module Clock: Module Run Mode Clock Control:** select System clock = 75 MHz ( = 13,3 ns)

**Resolutions:** (do nothing)
Control: Compare Register 0: Register Size: select CMP0[26:0]
Control: Compare Register 0: Compare Register 0 Value: insert 75000000
Control: Compare Register 0: Interrupt Control: click ☑ Enable Compare Register 0 interrupt
Control: Compare Register 0: Interrupt Output Selection: click ☑ Enable Interrupt Output

Note:
75,000,000 * 13,3333 ns = 1 s
There will be a System-Timer-Interrupt every 1 second.
The LED on IO_Port_0_Pin_7 will be blinking at a frequency of 1 second (done in the System-Timer-Interrupt-Service-Routine).
Interrupts: drag and drop STM SRN 0 from Level 0 (non interrupting) to Level 9
Functions: Initialization Function: click ✔ STM_vInit

Parameters: (do nothing)

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

Exit this dialog now by clicking the close button.

Note: The “function-header” for the System-Timer-Interrupt-Service-Routine will be generated automatically by DAvE
Port Configuration:

The configuration window can be opened by clicking the specific block/module.
Ports: click Configure Port 0
Ports: Configure Port 0
Port 0: Functionality: click ✓ Use P0.7 as general IO, General Direction: click ✗ Out

Note:
The LED used by our „hello-world-application“ is connected to Port 0 Pin 7 on the Starter-Kit-Board.

Input Characteristic: (do nothing)
Parameters: (do nothing)
Notes: If you wish, you can insert your comments here.

Exit this dialog now by clicking the close button.
**Functions:** Initialization Function: click ✔ IO_vInit
Functions: Function Library (Part 1): click ✔ IO_vSetPin
Functions: Function Library (Part 1): click ✔ IO_vResetPin
Functions: Function Library (Part 1): click ✔ IO_vTogglePin

**Parameters:** (do nothing)

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

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

File
Save

Save project: Save in: C:\TC1130  (create directory)
File name: TC1130

Save
Generate Code:

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

Please read this document carefully and note
the red-colored hints.

If you miss a file in the generated files list
maybe you have forgotten to select the
initialisation function of the related module.

Generated Files:

- TC1130REGS.H
- MAIN.H
- MAIN.C
- IO.H
- IO.C
- ASCO.H
- ASCO.C
- STM.H
- STM.C
- TC1130.ASM

Save changes?

click Yes
We recommend now to copy and store your project-directory “C:\TC1130” to “03_TC1130_DAvE-only”.

![Image of file explorer with selected project]

- Select an item to view its description.
- See also:
  - My Documents
  - My Network Places
  - My Computer
4.) Using of the TASKING - EDE Development Tools:

“ROM”: Locating programs into the 32 KBytes code scratchpad RAM (PMI_SPRAM),
“RAM”: Using OnChipSRAM (28 KBytes DMI_SPRAM + 64 KBytes DMU_SRAM)

Write programs for execution from PMI-Scratch-Pad RAM (PMI_SPRAM)
Note:

“ROM”:
Program, constant data and initialization-values for variables are located in „ROM-space“.

“RAM”:
Variables are located in “RAM-space”.
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
Choose C:\TC1130

OK
File
New Project Space
Insert TC1130
Click: “Add new project to project space”
Insert TC1130

OK
Click: “Add existing files to project”
Select ASC0.c
Select ASC0.h
Select IO.c
Select IO.h
Select MAIN.c
Select MAIN.h
Select STM.c
Select STM.h
Select TC1130Regs.h

Open
Configure Compiler, Assembler, Linker, Locator and Build – Control:

Project – Project Options

**Processor: Processor Definition: Target processor: select TC1130**
Processor: Bypasses: CPU Functional Problem Bypasses: click ✓ All bypasses TC1130

Note: The system startup code (lib/startup.asm) must have been added to your project.
C Compiler: Preprocessing: click to deactivate □ Automatic inclusion of .sfr file
C Compiler: Optimization: Optimization level: select No optimization
C Compiler: Allocation: click to deactivate  \(\Box\) Default \texttt{\_near} allocation for objects below threshold
Linker: Script File: Special Areas: RESET start address: insert 0xD4002100 (PMI_SPRAM)
Linker: Script File: Special Areas: Libraries start address: insert 0xD4004000 (PMI_SPRAM)
Linker: Script File: Special Areas: Interrupt table start address: insert 0xD4000000 (PMI_SPRAM)
Linker: Script File: Special Areas: Trap table start address: insert 0xD4002000 (PMI_SPRAM)
Linker: Script File: Special Areas: CSA start address: insert/checkout 0xD0000000 (DMI_LDRAM)

Note:
DMI Local Data RAM = DMI_LDRAM = “DMI_SPRAM”
**TC1130 (Vol. 1 of 2)**

**System Units**

**Memory Map of On-Chip Local Memories**

**Table 7-1  TC1130 Block Address Map (cont’d)**

<table>
<thead>
<tr>
<th>Segment</th>
<th>Address Range</th>
<th>Size</th>
<th>Description</th>
<th>DMI Acc.</th>
<th>PMI Acc.</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>c</td>
</tr>
<tr>
<td>12</td>
<td>C000 0000 -</td>
<td>64 KB</td>
<td>DPU</td>
<td>via L1MB</td>
<td>via L1MB</td>
<td>a</td>
</tr>
<tr>
<td></td>
<td>C3FF FFFF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>C001 0000 -</td>
<td>≈ 256 MB</td>
<td>Reserved</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>C3FF FFFF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>D000 0000 -</td>
<td>256 KB</td>
<td>DMI Local Data RAM</td>
<td>DMI local</td>
<td>via LMB</td>
<td>n</td>
</tr>
<tr>
<td></td>
<td>D000 FFFF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>≈ 64 MB</td>
<td>Reserved</td>
<td></td>
<td></td>
<td>n</td>
</tr>
<tr>
<td></td>
<td>D000 7000 -</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>n</td>
</tr>
<tr>
<td></td>
<td>D3FF FFFF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>n</td>
</tr>
<tr>
<td></td>
<td>D400 0000 -</td>
<td>32 KB</td>
<td>PMU Local Data Stack RAM (SPRAM)</td>
<td>via L1MB</td>
<td></td>
<td>n</td>
</tr>
<tr>
<td></td>
<td>D400 FFFF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>n</td>
</tr>
<tr>
<td></td>
<td>D400 8000 -</td>
<td>≈ 64 MB</td>
<td>Reserved</td>
<td></td>
<td></td>
<td>n</td>
</tr>
<tr>
<td></td>
<td>D7FF FFFF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>n</td>
</tr>
</tbody>
</table>

**AP32117**

TC1130 "Cookery-Book"
Additional Information:

Interrupt Vector Table:

For the full range of 256 interrupt entries an alignment to an 8 KBytes boundary is required.

BTV = 0xD400.0000 = 3.556.769.792 / 8.192 = 434.176

Trap Vector Table:

There are eight different trap classes, resulting in Trap Classes from 0 to 7. The contents of BTV should therefore be at least set to a 256 Byte boundary (8 Trap Classes * 8 word spacing).

BIV = 0xD400.2000 = 3.556.777.984 / 256 = 13.893.664
Linker: Script File: Internal Memory: change from Imbram to DMU_SRAM
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: DMU_SRAM: Alloc: select ON!!!
Linker: Script File: Internal Memory: PMI_SPRAM: Alloc: select ON!!!
Linker: Script File: Internal Memory: PMI_SPRAM: Type: select ROM

(Configuration of this dialog-window is now finished)
Linker: Script File: External Memory: ext_c: Alloc: select OFF
Linker: Script File: External Memory: ext_d: Alloc: select OFF
Linker: Script File: External Memory: vectable: Alloc: select OFF
Linker: Script File: Sections: linear Alloc: select intmem
Linker: Script File: Sections: linear Location: insert PMI_SPRAM
Linker: Map File: click ✓ Memory usage info

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[] =
"\r\n\n\n\n\n"
"Program execution out of PMI_SPRAM\r\n"
"==================================\r\n"
"1 ... LED IO_Port_0_Pin_7 ON\r\n"
"2 ... LED IO_Port_0_Pin_7 OFF\r\n"
"3 ... LED IO_Port_0_Pin_7 blinking\r\n"
"
\r\n",

const char question[] =
"your choice: ";

cost char message1[] =
"\n\n*** LED is ON ***\r\n";

const char message2[] =
"\n\n*** LED is OFF ***\r\n";

const char message3[] =
"\n\n*** LED is BLINKING ***\r\n";

volatile int RS232_wait=2;
volatile unsigned int blinking=ON;
char select=' ';```
TASKING EDE [ TriCore VX-toolset - C:\TC1130\TC1130.pjt ] - [C:\TC1130\MAIN.c ]

#endif

int main(void)
{
  char test[] = "Hello, world!"
  return 0;
}

int main()
{
  char test[] = "Hello, world!"
  return 0;
}
Double click: Main.c insert User Code [function: input()]:

```c
char input (void)
{
    char in=' '; 
    do 
    {
        myprintf(question);
        // ASC0_RSRC_SRR ... ASC0_Receive Interrupt Service Request Control Register_Service Request Flag
        // ASC0_RSRC_CLRR ... ASC0_Receive Interrupt 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;
}
```
while(RS232_wait);
while (1)
{
    myprintf(menu);
    select=input();

    switch (select)
    {
        case '1': blinking=OFF, IO_P0_7=LED_ON, myprintf(message1); break;
        case '2': blinking=OFF, IO_P0_7=LED_OFF, myprintf(message2); break;
        case '3': blinking=ON, myprintf(message3); break;
    }
}
Double click: Main.h and insert the following Defines:

```
#define OFF 0
#define ON 1
#define LED_ON 0
#define LED_OFF 1
```
Double click: Main.h and insert Global Variables (Extern Declarations):

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

STM_CMP0=STM_CMP0+75000000; // 75.000.000 * 13,3333 ns = 1 s

if(RS232_wait)
    RS232_wait--;

if(blinking)
    IO_vTogglePin(IO_P0_7);

Note:
75.000.000 * 13,3333 ns = 1 s

To get an STM-interrupt every 1 second you must change the Compare-Value to “STM_CMP0+=75000000;“, because there is no “reload-functionality”!

Warning:
If you forget to do so you will get the following undesired functionality:
1. Interrupt: 0b100011110000110100011000000 = 75000000 => 1 sec
2. Interrupt: 0b1100011110000110100011000000 = 209217728 => 2,789 sec
3. Interrupt: 0b10100011110000110100011000000 = 343435456 => 4,579 sec
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 not available for Tasking TriCore tools for the time being.

For the moment, Tasking has only got 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

Note:
We use myprintf(); instead of printf(); for all 32-bit-Cookery-Books just to be independent from the compiler-vendor-realisation of low-level-I/O-software for printf().
File – New
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!\r\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 "ASCO.H"

void myprintf(const char *p)
{
    while(*p)
    {
        if (ASCO_isHexadecimal(p))
            ASCO_printHex(*p++);
        else
            ASCO_printChar(*p++);
    }

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

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

        sprintf(sh, "Variable set = \%d", dummy); // Write formatted data to string sh
        myprintf(sh);
    }
}*/
```
File
Save all

(Project Window File View) – TC1130 (Files) – right mouse button click – Add Existing Files – Browse
Select myprintf.c

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

void myprintf(const char *p)
{
    while(*p)
    {
        if (ASC0_asciitofive(*p))
            ASCII_vdi08Note[0++];
        else
            ASCII_vdi08Note[1++];
    }
}

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

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

    sprintf(mb,"Variable unit = %d",dummy); // Write formatted data to string mb
    myprintf(mb);
}
```

Double click: Main.h and insert Prototypes of Global Functions (Extern Declaration):

```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

Execute 'Rebuild' command.

Now you can close both your project and Tasking EDE:

File - Close Project Space
File – Exit
Programming is now complete. You can now load and run your program
(Therefore we are going to use the pls Debugger):

Start pls-Debugger
File – New Workspace

![UDE Desktop](image)

Create new workspace
Look in: select C:\TC1130

Open
For Help, press F1

New
Click ☑ Use a default target configuration
Select Triboard with TC1130 (JTAG/OCDS)

Finish

Note: The correct value for „64MB external SDRAM at 0xA2000000” will be 0xA4000000 later!
Save in: select C:\TC1130
File name: insert TC1130

Save
OK
File – Load Program
Select TC1130.elf

Open
```c
// USER CODE EMD

swReturn = 0;

switch(RESET_INDICATOR) {
    case DEEPSLEEP_WAKEUP:   // the last reset was an external triggered
                              // hardware reset in Deep sleep mode
                            // (wake-up reset)
        // USER CODE BEGIN (Main,3)
        // USER CODE END
```
Debug – Start Program Execution

// USER CODE BEGIN (Main, 4)

// USER CODE END

MSG: Controller0.Core: USBDebugServer: Program with ID 0x1 - code size 3658 bytes was loaded!
MSG: Controller0.Core: USBDebugServer: Program with ID 0x1 - code size 3658 bytes was loaded!
Execute any terminal-program
(9600 Baud, 8 bit Data, no Parity-Bit, 1 Stop-Bit, Xon/Xoff Protocol):

And see the result:

Program execution out of PMI_SRAM
===================================
1 ... LED IO_Port_0_Pin_T ON
2 ... LED IO_Port_0.Pin_T OFF
3 ... LED IO_Port_0.Pin_T blinking

your choice: 3
*** LED is BLINKING ***

Program execution out of PMI_SRAM
===================================
1 ... LED IO_Port_0.Pin_T ON
2 ... LED IO_Port_0.Pin_T OFF
3 ... LED IO_Port_0.Pin_T blinking

your choice: 1

Note:
As an example for “any terminal-program” we use mttty.
MTTY can be downloaded @ http://www.freeware.de/software/DetailEN_MTTY_19383.html
File – Close Workspace

```c
// USER CODE END

switch(RESET_INDICATOR)
{
    case DEEPSLEEP_WAKEUP:
       // the last reset was an external triggered
       // hardware reset in deep sleep mode
       // (wake-up reset)

       // USER CODE BEGIN (Main3)
       // USER CODE END

    case WATCHDOG_RESET:
    case USER CODE BEGIN:

    // USER CODE BEGIN
    // USER CODE END
```
We recommend now to **copy and store** your project-directory “C:\TC1130” to “04_TC1130_SPRAM”:
5.) Using of the TASKING - EDE Development Tools:

“ROM”:
Locating programs into the Intel’s 32 MBytes OnBoardFlash,

“RAM”:
Using
OnChipSRAM (DMI_SPRAM+DMU_SRAM) + Infineon’s 64 MBytes OnBoardSDRAM

Write programs for execution from OnBoardFlash.
TC1130-Execution-Environment: OnBoardFlash

**Jumper S301 Settings** (HW-Configuration DIP-Switch):
### Additional Information:

<table>
<thead>
<tr>
<th>/BRK_IN</th>
<th>CFG[2...0]</th>
<th>Type of Boot</th>
<th>PC Start value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>000</td>
<td>Serial boot from ASC to PMI scratchpad, run loaded program</td>
<td>0xD4000000</td>
</tr>
<tr>
<td>1</td>
<td>001</td>
<td>Serial boot from CAN to PMI scratchpad, run loaded program</td>
<td>0xDFFFFF7FC</td>
</tr>
<tr>
<td>1</td>
<td>010</td>
<td>Serial boot from SSC to PMI scratchpad, run loaded program</td>
<td>0xDFFFFF7FC</td>
</tr>
<tr>
<td>1</td>
<td>011</td>
<td>External memory, EBU as master</td>
<td>0xA0000000</td>
</tr>
<tr>
<td>1</td>
<td>100</td>
<td>External memory, EBU as slave</td>
<td>0xA0000000</td>
</tr>
<tr>
<td>1</td>
<td>101</td>
<td>External memory, EBU as master</td>
<td>0xA0000000</td>
</tr>
<tr>
<td>1</td>
<td>110</td>
<td>PMI scratchpad</td>
<td>0xD4000000</td>
</tr>
<tr>
<td>1</td>
<td>111</td>
<td>reserved; don’t use this combination</td>
<td>-</td>
</tr>
<tr>
<td>0</td>
<td>01x, 1xx</td>
<td>reserved; don’t use this combination</td>
<td>-</td>
</tr>
<tr>
<td>0</td>
<td>001</td>
<td>go to external emulator space</td>
<td>0xDE000000</td>
</tr>
<tr>
<td>0</td>
<td>000</td>
<td>put chip in tristate (deep sleep)</td>
<td>-</td>
</tr>
</tbody>
</table>

---

S301:

3 : ON
1, 2, 4, 5, 6, 7, 8 : OFF
252 MBytes External Memory Space used for:
“ROM”: 32 MBytes OnBoardFlash = 33,554,432 Bytes @ 0xA000.0000
“RAM”: 64 MBytes OnBoardSDRAM = 67,108,864 Bytes @ 0xA400.0000
Memory Map: Used OnBoard Memories:

64 MBytes OnBoard SDRAM

64,108,864 Bytes

32 MBytes OnBoard FLASH

33,554,432 Bytes
Start DAvE and open the TC1130 project:

File
Open
Look in: select C:\TC1130
File name: select TC1130.dav
File – Project Settings

Startup Configuration: **Hardware Booting Scheme**: click ☑ EBU mode

Startup Configuration: **Hardware Booting Scheme**: Boot type: select/check 011

Exit this dialog now by clicking the close button.
Generate Code:

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

Please read this document carefully and note the red-colored hints.

If you miss a file in the generated files list maybe you have forgotten to select the initialisation function of the related module.

Generated Files:
- TC1130REGS.H
- MAIN.H
- MAIN.C
- IO.H
- IO.C
- ASC0.H
- ASC0.C
- STM.H
- STM.C
- TC1130.ASM

File
Exit
Save changes?

click Yes
Start Tasking EDE and open the project:

File – Open Project Space

Look in: select C:\TC1130
File name: select TC1130.psp

Open
default:
    
    // USER CODE BEGIN (Xglw,8)
    
    // USER CODE END

MAIN_visit();

// USER CODE BEGIN (main,9)
while(RS232_wait);
while (1)
{
    xprintf(emacs);
    selectInput();

    switch (select)
    {
        case '1': blinking=OFF, IO_PEG, LED_0H, xprintf(emacs); break;
        case '2': blinking=OFF, IO_PEG, LED_OFF, xprintf(emacs); break;
        case '3': blinking=ON, xprintf(emacs); break;
    }

    // USER CODE END

    return(xReturn);

    // End of function main

    // USER CODE BEGIN (MAIN_Genera2,10)
    
    // USER CODE END
Configure Compiler, Assembler, Linker, Locator and Build – Control:

**Project – Project Options**

**Processor:** Startup: Startup Code: BootMemory: Boot Memory: Value: **insert 0x00000806D**

**Note:**

**External Boot Memory Configuration Word:**
If external boot is selected, the EBU will perform (exactly) one external bus read access to a specific address (0x0000004 / 0x000010) of the memory device attached to CS0. The result of this read access is used to configure the EBU.

To see “Troubleshooting” click here.
Table 14-11  EBU Address Regions, Registers, and Chip Selects

<table>
<thead>
<tr>
<th>Address Region</th>
<th>Address Select Register</th>
<th>Bus Control Register</th>
<th>Bus Access Parameter Register</th>
<th>Chip Select</th>
</tr>
</thead>
<tbody>
<tr>
<td>User region 0</td>
<td>ADDRSEL0</td>
<td>BUSCON0</td>
<td>BUSAP0</td>
<td>CS0</td>
</tr>
<tr>
<td>User region 1</td>
<td>ADDRSEL1</td>
<td>BUSCON1</td>
<td>BUSAP1</td>
<td>CS1</td>
</tr>
<tr>
<td>User region 2</td>
<td>ADDRSEL2</td>
<td>BUSCON2</td>
<td>BUSAP2</td>
<td>CS2</td>
</tr>
<tr>
<td>User region 3</td>
<td>ADDRSEL3</td>
<td>BUSCON3</td>
<td>BUSAP3</td>
<td>CS3</td>
</tr>
<tr>
<td>Emulator region</td>
<td>EMUAS</td>
<td>EMUBC</td>
<td>EMUBAP</td>
<td>CSEMU</td>
</tr>
</tbody>
</table>

Processor: Bus Configuration: Address Select Registers: AddrSel0: insert 0xA0000021
Table 14-11  EBU Address Regions, Registers, and Chip Selects

<table>
<thead>
<tr>
<th>Address Region</th>
<th>Address Select Register</th>
<th>Bus Control Register</th>
<th>Bus Access Parameter Register</th>
<th>Chip Select</th>
</tr>
</thead>
<tbody>
<tr>
<td>User region 0</td>
<td>ADDRSEL2</td>
<td>BUSCON0</td>
<td>BUSAP0</td>
<td>CS0</td>
</tr>
<tr>
<td>User region 1</td>
<td>ADDRSEL1</td>
<td>BUSCON1</td>
<td>BUSAP1</td>
<td>CS1</td>
</tr>
<tr>
<td>User region 2</td>
<td>ADDRSEL2</td>
<td>BUSCON2</td>
<td>BUSAP2</td>
<td>CS2</td>
</tr>
<tr>
<td>User region 3</td>
<td>ADDRSEL3</td>
<td>BUSCON3</td>
<td>BUSAP3</td>
<td>CS3</td>
</tr>
<tr>
<td>Emulator region</td>
<td>EMUAS</td>
<td>EMUBC</td>
<td>EMUBAP</td>
<td>CSEMU</td>
</tr>
</tbody>
</table>

**ADDRSEL[3:0]**

EBU Address Select Register x

| ADDRSEL[3:1] Reset Value: 0000 0000<sub>H</sub> |
|ADDRSEL0 Reset Value (internal boot): 0000 0000<sub>H</sub> |
|ADDRSEL0 Reset Value (external boot): A000 0001<sub>H</sub> |

```
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16
BASE
rw
```

```
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
BASE ALTSEG MASK 0 ALTERN AB REGEN AB
rw rw rw rw rw
```

**Note:**
Memory Region Address Mask: Specifies the number of rightmost bits in the base address starting at bit 26, which should be included in the address comparison. Bits (31:27) will always be part of the comparison.

32 MBytes OnBoardFlash = 10000000000000000000000000000000 b
Base = A0000
Mask = 2

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>= Mask</th>
</tr>
</thead>
<tbody>
<tr>
<td>[31:27]</td>
<td>[26]</td>
<td>[25] [24:0]</td>
</tr>
<tr>
<td>1</td>
<td>00000000000000000000000000000000</td>
<td>32 MBytes OnBoardFlash</td>
</tr>
</tbody>
</table>

Note: Memory Region Address Mask: Specifies the number of rightmost bits in the base address starting at bit 26, which should be included in the address comparison. Bits (31:27) will always be part of the comparison.

32 MBytes OnBoardFlash = 10000000000000000000000000000000 b
Base = A0000
Mask = 2

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>= Mask</th>
</tr>
</thead>
<tbody>
<tr>
<td>[31:27]</td>
<td>[26]</td>
<td>[25] [24:0]</td>
</tr>
<tr>
<td>1</td>
<td>00000000000000000000000000000000</td>
<td>32 MBytes OnBoardFlash</td>
</tr>
</tbody>
</table>
Table 14-11  EBU Address Regions, Registers, and Chip Selects

<table>
<thead>
<tr>
<th>Address Region</th>
<th>Address Select Register</th>
<th>Bus Control Register</th>
<th>Bus Access Parameter Register</th>
<th>Chip Select</th>
</tr>
</thead>
<tbody>
<tr>
<td>User region 0</td>
<td>ADDRSEL0</td>
<td>BUSCON0</td>
<td>BUSAP0</td>
<td>CS0</td>
</tr>
<tr>
<td>User region 1</td>
<td>ADDRSEL1</td>
<td>BUSCON1</td>
<td>BUSAP1</td>
<td>CS1</td>
</tr>
<tr>
<td>User region 2</td>
<td>ADDRSEL2</td>
<td>BUSCON2</td>
<td>BUSAP2</td>
<td>CS2</td>
</tr>
<tr>
<td>User region 3</td>
<td>ADDRSEL3</td>
<td>BUSCON3</td>
<td>BUSAP3</td>
<td>CS3</td>
</tr>
<tr>
<td>Emulator region</td>
<td>EMUAS</td>
<td>EMUBC</td>
<td>EMUBAP</td>
<td>CSEMU</td>
</tr>
</tbody>
</table>

Processor: Bus Configuration: Address Select Registers: AddrSel1: insert 0xA4000011
Note:
64 MBytes are an address-space between 0x0000.0000 and 0x0400.0000. Therefore you should use a boundary of 0x0400.0000.

<table>
<thead>
<tr>
<th>Address Region</th>
<th>Address Select Register</th>
<th>Bus Control Register</th>
<th>Bus Access Parameter Register</th>
<th>Chip Select</th>
</tr>
</thead>
<tbody>
<tr>
<td>User region 0</td>
<td>ADDRSEL0</td>
<td>BUSCON0</td>
<td>BUSAP0</td>
<td>CS0</td>
</tr>
<tr>
<td>User region 1</td>
<td>ADDRSEL1</td>
<td>BUSCON1</td>
<td>BUSAP1</td>
<td>CS1</td>
</tr>
<tr>
<td>User region 2</td>
<td>ADDRSEL2</td>
<td>BUSCON2</td>
<td>BUSAP2</td>
<td>CS2</td>
</tr>
<tr>
<td>User region 3</td>
<td>ADDRSEL3</td>
<td>BUSCON3</td>
<td>BUSAP3</td>
<td>CS3</td>
</tr>
<tr>
<td>Emulator region</td>
<td>EMUAS</td>
<td>EMUBC</td>
<td>EMUBAP</td>
<td>CSEMU</td>
</tr>
</tbody>
</table>
Note:
Memory Region Address Mask: Specifies the number of rightmost bits in the base address starting at bit 26, which should be included in the address comparison. Bits (31:27) will always be part of the comparison.

64 MBytes OnBoardSDRAM = 0x0400.0000 = 100000000000000000000000000000000 b
Base = A4000
Mask = 1

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ADDRSEL0</td>
<td>ADDRSEL0 Reset Value (internal boot): 0000 0000 H</td>
</tr>
<tr>
<td>ADDRSEL0</td>
<td>ADDRSEL0 Reset Value (external boot): A000 0001 H</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ADDRSEL0</th>
<th>BUSCON0</th>
<th>BUSAP0</th>
<th>CS0</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADDRSEL1</td>
<td>BUSCON1</td>
<td>BUSAP1</td>
<td>CS1</td>
</tr>
<tr>
<td>ADDRSEL2</td>
<td>BUSCON2</td>
<td>BUSAP2</td>
<td>CS2</td>
</tr>
<tr>
<td>ADDRSEL3</td>
<td>BUSCON3</td>
<td>BUSAP3</td>
<td>CS3</td>
</tr>
<tr>
<td>EMUAS</td>
<td>EMUBC</td>
<td>EMUBAP</td>
<td>CSEMU</td>
</tr>
</tbody>
</table>

64 MBytes OnBoardSDRAM = 0x0400.0000 = 100000000000000000000000000000000 b
Base = A4000
Mask = 1

<table>
<thead>
<tr>
<th>31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16</th>
</tr>
</thead>
<tbody>
<tr>
<td>BASE</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>BASE</td>
</tr>
<tr>
<td>rw</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>[31:27]</th>
<th>[26]</th>
<th>[25:0]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>000000000000000000000000000000000</td>
<td>= 64 MBytes OnBoardSDRAM</td>
</tr>
</tbody>
</table>
### Table 14-11  EBU Address Regions, Registers, and Chip Selects

<table>
<thead>
<tr>
<th>Address Region</th>
<th>Address Select Register</th>
<th>Bus Control Register</th>
<th>Bus Access Parameter Register</th>
<th>Chip Select</th>
</tr>
</thead>
<tbody>
<tr>
<td>User region 0</td>
<td>ADDRSEL0</td>
<td>BUSCON0</td>
<td>BUSAP0</td>
<td>CS0</td>
</tr>
<tr>
<td>User region 1</td>
<td>ADDRSEL1</td>
<td>BUSCON1</td>
<td>BUSAP1</td>
<td>CS1</td>
</tr>
<tr>
<td>User region 2</td>
<td>ADDRSEL2</td>
<td>BUSCON2</td>
<td>BUSAP2</td>
<td>CS2</td>
</tr>
<tr>
<td>User region 3</td>
<td>ADDRSEL3</td>
<td>BUSCON3</td>
<td>BUSAP3</td>
<td>CS3</td>
</tr>
<tr>
<td>Emulator region</td>
<td>EMUAS</td>
<td>EMUBC</td>
<td>EMUBAP</td>
<td>CSEMU</td>
</tr>
</tbody>
</table>

**Note:** “User region 2” = NOT USED !!!

**Processor:** Bus Configuration: Address Select Registers: AddrSel2: insert 0xA200030

![Processor Configuration Diagram](image)
<table>
<thead>
<tr>
<th>Address Region</th>
<th>Address Select Register</th>
<th>Bus Control Register</th>
<th>Bus Access Parameter Register</th>
<th>Chip Select</th>
</tr>
</thead>
<tbody>
<tr>
<td>User region 0</td>
<td>ADDRSEL0</td>
<td>BUSCON0</td>
<td>BUSAP0</td>
<td>CS0</td>
</tr>
<tr>
<td>User region 1</td>
<td>ADDRSEL1</td>
<td>BUSCON1</td>
<td>BUSAP1</td>
<td>CS1</td>
</tr>
<tr>
<td>User region 2</td>
<td>ADDRSEL2</td>
<td>BUSCON2</td>
<td>BUSAP2</td>
<td>CS2</td>
</tr>
<tr>
<td>User region 3</td>
<td>ADDRSEL3</td>
<td>BUSCON3</td>
<td>BUSAP3</td>
<td>CS3</td>
</tr>
<tr>
<td>Emulator region</td>
<td>EMUAS</td>
<td>EMUBC</td>
<td>EMUBAP</td>
<td>CSEMU</td>
</tr>
</tbody>
</table>

**Processor:** Bus Configuration: Bus Access Parameters Registers: Busap0: insert 0xC6DB0000
### Processor: Bus Configuration: Bus Access Parameters Registers: Busap1: insert 0x42000000

### Table 14-11  EBU Address Regions, Registers, and Chip Selects

<table>
<thead>
<tr>
<th>Address Region</th>
<th>Address Select Register</th>
<th>Bus Control Register</th>
<th>Bus Access Parameter Register</th>
<th>Chip Select</th>
</tr>
</thead>
<tbody>
<tr>
<td>User region 0</td>
<td>ADDRSEL0</td>
<td>BUSCON0</td>
<td>BUSAP0</td>
<td>CS0</td>
</tr>
<tr>
<td>User region 1</td>
<td>ADDRSEL1</td>
<td>BUSCON1</td>
<td>BUSAP1</td>
<td>CS1</td>
</tr>
<tr>
<td>User region 2</td>
<td>ADDRSEL2</td>
<td>BUSCON2</td>
<td>BUSAP2</td>
<td>CS2</td>
</tr>
<tr>
<td>User region 3</td>
<td>ADDRSEL3</td>
<td>BUSCON3</td>
<td>BUSAP3</td>
<td>CS3</td>
</tr>
<tr>
<td>Emulator region</td>
<td>EMUAS</td>
<td>EMUBC</td>
<td>EMUBAP</td>
<td>CSEMU</td>
</tr>
</tbody>
</table>

![Image of TriCore VX-toolset Project Options](image)
### Table 14-11  EBU Address Regions, Registers, and Chip Selects

<table>
<thead>
<tr>
<th>Address Region</th>
<th>Address Select Register</th>
<th>Bus Control Register</th>
<th>Bus Access Parameter Register</th>
<th>Chip Select</th>
</tr>
</thead>
<tbody>
<tr>
<td>User region 0</td>
<td>ADDRSEL0</td>
<td>BUSCON0</td>
<td>BUSAP0</td>
<td>CS0</td>
</tr>
<tr>
<td>User region 1</td>
<td>ADDRSEL1</td>
<td>BUSCON1</td>
<td>BUSAP1</td>
<td>CS1</td>
</tr>
<tr>
<td>User region 2</td>
<td>ADDRSEL2</td>
<td>BUSCON2</td>
<td>BUSAP2</td>
<td>CS2</td>
</tr>
<tr>
<td>User region 3</td>
<td>ADDRSEL3</td>
<td>BUSCON3</td>
<td>BUSAP3</td>
<td>CS3</td>
</tr>
<tr>
<td>Emulator region</td>
<td>EMUAS</td>
<td>EMUBC</td>
<td>EMUBAP</td>
<td>CSEMU</td>
</tr>
</tbody>
</table>

**Processor: Bus Configuration: Bus Control Registers: Buscon0: insert 0x00922300**
Table 14-11  EBU Address Regions, Registers, and Chip Selects

<table>
<thead>
<tr>
<th>Address Region</th>
<th>Address Select Register</th>
<th>Bus Control Register</th>
<th>Bus Access Parameter Register</th>
<th>Chip Select</th>
</tr>
</thead>
<tbody>
<tr>
<td>User region 0</td>
<td>ADDRSEL0</td>
<td>BUSCON0</td>
<td>BUSAP0</td>
<td>CS0</td>
</tr>
<tr>
<td>User region 1</td>
<td>ADDRSEL1</td>
<td>BUSCON1</td>
<td>BUSAP1</td>
<td>CS1</td>
</tr>
<tr>
<td>User region 2</td>
<td>ADDRSEL2</td>
<td>BUSCON2</td>
<td>BUSAP2</td>
<td>CS2</td>
</tr>
<tr>
<td>User region 3</td>
<td>ADDRSEL3</td>
<td>BUSCON3</td>
<td>BUSAP3</td>
<td>CS3</td>
</tr>
<tr>
<td>Emulator region</td>
<td>EMUAS</td>
<td>EMUBC</td>
<td>EMUBAP</td>
<td>CSEMU</td>
</tr>
</tbody>
</table>

Processor: Bus Configuration: Bus Control Registers: Buscon1: check/insert 0x30B20000
Processor: Bus Configuration: SDRAM Registers: Sdrmref0: insert 0x000000C9
Processor: Bus Configuration: SDRAM Registers: Sdrmref1: insert/check 0x00000000
Processor: Bus Configuration: SDRAM Registers: Sdrmcon0: insert 0x219E2075
Processor: Bus Configuration: SDRAM Registers: Sdrmcon1: insert/check 0x00000000
Processor: Bus Configuration: SDRAM Registers: Sdrmod0: insert/check 0x00000023
**Processor:** Bus Configuration: SDRAM Registers: Sdrmod1: insert/check 0x00000000

---

**Table: SDRAM Mode Register 1 (SDRAM0)**

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-2</td>
<td>BURSTL Burst length</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>BTYP Burst type</td>
<td>sequential burst</td>
</tr>
<tr>
<td>4-6</td>
<td>CASLAT CAS latency</td>
<td></td>
</tr>
<tr>
<td>7-13</td>
<td>OPMODE Operation Mode</td>
<td>burst write</td>
</tr>
</tbody>
</table>
Processor: Bus Configuration: Other Registers: S_bcucon: (do nothing)
**Processor:** Bus Configuration: Other Registers: Ebuclc: (do nothing)
Processor: Bus Configuration: Other Registers: Ebucon: insert/check 0x00F9FF68
Processor: Bus Configuration: Other Registers: Bfcon: insert 0x00002993
252 MBytes External Memory Space used for:
“ROM”: 32 MBytes OnBoardFlash = 33,554,432 Bytes @ 0xA000.0000
“RAM”: 64 MBytes OnBoardSDRAM = 67,108,864 Bytes @ 0xA400.0000
To avoid a Linker/Locater-Problem (Function Calling Mode):

From: Altium Support [mailto:support.ap@altium.com]
Sent: Wednesday, January 31, 2007 11:34 AM
To: (IFAT S FAE)
Subject: Your case 00023116 has been updated

Hi,

Your case # 00023116: 23116 - relocation patch error [tricore] has been updated.

Please click on the link below to view this case in the SUPPORTcenter.
http://www.altium.com/supportcenter

The following comment was added to your case:

I order to explain your case let me start by repeating your error message:

ltc E121: relocation patch error in "task1": relocation value 0xa11004b0 for relocation of type rel24 or abs24 at offset 92 in section ".text.main.main" at address 0xa0000188 is not a valid address in R_TRICORE_24REL. Hint: check the mapfile for a section that occupies this address.

I admit this is a bit of a mouthful, and it means:

patch error at offset 92 of section .text.main.main

What I always do then is to enable list files and tick the "section directives" CHECKBOX after you've done that. Since the offending section is called .text.main.main listfile main.lst must be studied. Lookup section .text.main.main and then offset 92 (0x5C). You'll end up at:

005C 6Drmmm  2   30   276         call    sprintf

Since the linker says "patch error" it means that the call to sprintf is too big to fit expression 'rrrrrr'. To make this work either the call must be changed or the libraries must be located closer to the application code. Try locating the libraries at 0xA1100000 from the same PAGE where you located the reset vector. It will do magic.

Regards,
Altium Customer Support

you have to locate the target address of your code (e.g. RESET start address and Libraries start address) within the range +/- 16 MByte of any 256 MByte Segment [The Linker/Locater generates an error (see above) when the target address appears to be out of reach].

Note (Function Calling Mode Indirect):
The __indirect keyword can be used to force the less efficient indirect call [the target address may be any 32 bit address within the whole memory space (4 GByte)].
Additional Information:

See also the Online-Manual:

Click TriCore User’s Manual
Linker: Script File: Special Areas: RESET start address: insert 0xA0000000 (OnBoardFLASH)
Linker: Script File: Special Areas: Libraries start address: insert 0xA0100000 (OnBoardFLASH)
Linker: Script File: Special Areas: Interrupt table start address: insert 0xA0200000 (OnBoardFLASH)
Linker: Script File: Special Areas: Trap table start address: insert 0xA1FFFFFF (OnBoardFLASH)
Additional Information:

Memory-Map OnBoard-Flash:

- **Trap Vector Table**
  - Size = \(8 \times 8\) Words = \(8 \times 32\) Bytes = \(256\) Bytes = \(6.25\) KBytes = \(6400\) Bytes

- **Interrupt Vector Table**
  - Size = \(256 \times 3\) Words = \(256 \times 32\) Bytes = \(8192\) Bytes = \(0x2000\) Bytes = \(8\) KBytes

- **Libraries start address**
  - Size = \(1048576\) Bytes = \(1\) MByte

- **Reset start address**
  - Size = \(1048576\) Bytes = \(0x100000\) Bytes = \(1\) MByte

- **Libraries start address**
  - Size = \(1048576\) Bytes = \(0x100000\) Bytes = \(1\) MByte

- **Libraries start address**
  - Size = \(1048576\) Bytes = \(0x100000\) Bytes = \(1\) MByte

- **Libraries start address**
  - Size = \(1048576\) Bytes = \(0x100000\) Bytes = \(1\) MByte
Additional Information:

Interrupt–Vector–Table:

DAvE:
Additional Information:

Interrupt–Vector–Table:

Note:
PN … Priority Number

Click here to see the Map File
Additional Information:

TRAP–Vector–Table:

<table>
<thead>
<tr>
<th>Address</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xA1FF.FFF0</td>
<td>Class_7</td>
</tr>
<tr>
<td>0xA1FF.FFE0</td>
<td>Class_6</td>
</tr>
<tr>
<td>0xA1FF.FFC0</td>
<td>Class_5</td>
</tr>
<tr>
<td>0xA1FF.FFA0</td>
<td>Class_4</td>
</tr>
<tr>
<td>0xA1FF.FF80</td>
<td>Class_3</td>
</tr>
<tr>
<td>0xA1FF.FF60</td>
<td>Class_2</td>
</tr>
<tr>
<td>0xA1FF.FF40</td>
<td>Class_1</td>
</tr>
<tr>
<td>0xA1FF.FF20</td>
<td>Class_0</td>
</tr>
</tbody>
</table>

Note:
1 Word = 32 Bit
1 Word = 4 Bytes
8 Words = 32 Bytes

Click here to see the Map File
Additional Information:

Interrupt Vector Table:

For the full range of 256 interrupt entries an alignment to an 8 KBytes boundary is required.

\[ BTV = 0xA020.0000 = \frac{2.686.451.712}{8.192} = 327.936 \]

Trap Vector Table:

There are eight different trap classes, resulting in Trap Classes from 0 to 7. The contents of BTV should therefore be at least set to a 256 Byte boundary (8 Trap Classes * 8 word spacing).

\[ BIV = 0xA1FF.FF00 = \frac{2.717.908.736}{256} = 10.616.831 \]
Linker: Script File: Internal Memory: PMI_SPRAM: Alloc: select OFF

(Configuration of this dialog-window is now finished)
Linker: Script File: External Memory: Name: “vecttable” change to “TRAP_vecttable”
Linker: Script File: External Memory: Size (TRAP_vectable): change to 256
Linker: Script File: External Memory: Address (TRAP_vectable): change to 0xA1FFFF00

Note:

32 Mbytes OnBoardFlash from A000.0000 to A1FF.FFFF – 256 Bytes “TRAP_vectable” = from A000.0000 to 0xA1FFDFFF = 33,546,240 Bytes
Linker: Script File: External Memory: Name=ext_c: Alloc: select “ON”
Linker: Script File: External Memory: Name=ext_d: Alloc: select “ON”
Linker: Script File: External Memory: Name=TRAP_vector_table: Alloc: select “ON”
Linker: Script File: External Memory: Name=ext_c: Size: insert  “33546240” [ROM]
Linker: Script File: External Memory: Name=ext_d: Size: insert “64M” [RAM]

Note:

32 Mbytes OnBoardFlash from A000.0000 to A1FF.FFFF – 256 Bytes “TRAP_vecttable” = from A000.0000 to 0xA1FFDDFF = 33.546.240 Bytes
Linker: Script File: External Memory:

Name=ext_c: Address: check/insert 0xA0000000 (ROM)
Name=ext_d: Address: insert 0xA4000000 (RAM)
Name=TRAP_vecttable: Address: check/insert 0xA1FFFF00 (ROM)
Linker: Script File: Sections: Space=linear: Alloc: select “extmem”
Linker: Script File: Sections: Space=linear: Location: insert “ext_c”
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 and change Global Variable menu from

```
const char menu[] =
"\r\n\n\n\n"
"Program execution out of PMI_SPRAM\r\n"
"==================================\r\n"
"1 ... LED IO_Port_0_Pin_7 ON\r\n"
"2 ... LED IO_Port_0_Pin_7 OFF\r\n"
"3 ... LED IO_Port_0_Pin_7 blinking\r\n"
" \r\n";
```

to

```
const char menu[] =
"\r\n\n\n\n"
"Program execution out of 32 MBytes OnBoardFlash\r\n"
"===============================================\r\n"
"1 ... LED IO_Port_0_Pin_7 ON\r\n"
"2 ... LED IO_Port_0_Pin_7 OFF\r\n"
"3 ... LED IO_Port_0_Pin_7 blinking\r\n"
" \r\n";
```
Insert Map File:

(Project Window File View) – TC1130 (Files) – right mouse button click – Add Existing Files – Browse

Select TC1130.map

Open - OK
Generate your application program:

Build
Rebuild

or

Execute 'Rebuild' command.
See Map File:

Interrupt Vector Table:

```
<table>
<thead>
<tr>
<th>Level</th>
<th>CPU Interrupt (max 255)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level 16</td>
<td></td>
</tr>
<tr>
<td>Level 15</td>
<td></td>
</tr>
<tr>
<td>Level 14</td>
<td></td>
</tr>
<tr>
<td>Level 13</td>
<td></td>
</tr>
<tr>
<td>Level 12</td>
<td></td>
</tr>
<tr>
<td>Level 11</td>
<td></td>
</tr>
<tr>
<td>Level 10</td>
<td></td>
</tr>
<tr>
<td>Level  9</td>
<td>STM SRN 0</td>
</tr>
</tbody>
</table>
```

Click here to see Memory Map (Interrupt Vector Table).
See Map File:

Trap Vector Table:

Click here to see Memory Map (Trap Vector Table).

Now you can close both your project and Tasking EDE:

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

Start pls-Debugger

For Help, press F1
File – Open Workspace

Look in: select C:\TC1130
File name: select TC1130.wsp

Open
Cancel
Config – Add-In Components
Click ✓ UDE FLASH/OTP Memory Programming Tool

OK
Tools – FLASH Programming …
FLASH/OTP – Memory Device: select 32MB External Flash (not ready)
FLASH/OTP – Memory Device: click ✔ Enable
click Setup …
Program: click ✓ Automatic Verify after Program

OK
Exit
File – Load Program

Look in: select TC1130
File name: select TC1130.elf

Open
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:

The application runs with the Debugger!
The application does NOT run after “Power-ON” !!!
Troubleshooting:

The application runs with the Debugger!
The application does NOT run after “Power-ON” !!!

➔ The only difference between “Debugger” and “Power-ON” program-execution is that the debugger does NOT read the “external-boot-memory-configuration-word”.
Therefore we take a closer look at this item:

External Boot Memory Configuration Word:
If external boot is selected, the EBU will perform (exactly) one external bus-read-access to a specific address (0x0000004) of the memory device attached to CS0.
The result of this read-access is used during “Power-ON” to configure the EBU.
To see the configuration value click here.

➔ Because of the 32-bit-alignment we need the “external-boot-memory-configuration-word” at address 0xA0000010.

![EBU Boot Configuration Value Table]

```
<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>CONFIG BIT</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>CMULT</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>BCGEN</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>WAIT</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>WATRCD</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>ADDRDC</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>ADDRDC</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>ADDRDC</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>ADDRC</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>ADDRC</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>ADDRC</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>ADDRC</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>ADDRC</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>ADDRC</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>ADDRC</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>ADDRC</td>
<td>3</td>
</tr>
</tbody>
</table>
```

“external-boot-memory-configuration-word” at address 0xA0000010
( 0xA0000004 << 2 = 0xA0000010, due to 32 bit alignment )
To check this we use the pls-debugger:

```
// USER CODE END

// swReturn = 0;

switch(RESET_INT) {
  case DEEPSLEEP:
    // USER CODE
    // USER CODE
    case WATCHDOG:
    // USER CODE
    // USER CODE
```

Views – Memory Window
Insert 0xA0000000 <ENTER>

External Boot Memory Configuration Word @ 0xA000.0004
Address Alignment = 0

External Boot Memory Configuration Word @ 0xA000.0010
Address Alignment = 1
As you can see:
The “external-boot-memory-configuration-word” value (0x0000806D) is NOT @ 0xA0000010 - it is still @ 0xA0000004

Wrong Value (0x……..)
Correct Location (0xA0000010)

Correct Value (0x0000806D)
Wrong Location (0xA0000004)

=> We consider this as a Compiler/EDE-Error and therefore we change the cstart.asm file!
Start Tasking EDE and open the project:

File – Open Project Space

Look in: select C:\TC1130
File name: select TC1130.psp

Open
Insert your application specific program:

Double click: cstart.asm and insert code:

```assembly
; external boot memory configuration word at address 0xA0000010
; ( 0xA0000004 << 2 = 0xA0000010, due to 32 bit alignment )
.word   (_BOOTCFG_VALUE)
.word   (_BOOTCFG_VALUE)
.word   (_BOOTCFG_VALUE)
```
Generate your application program:

Build
Rebuild

or

Execute 'Rebuild' command.
Programming and Troubleshooting is now complete. You can now load and run your program:

Start pls-Debugger
File – Open Workspace

**Look in:** select C:\TC1130  
**File name:** select TC1130.wsp

Open
Load
Program All
Exit

Exit
Views – Memory Window

```c
// USER CODE BNI

swReturn = 0;
switch(RESET_INTE)
{
    case DEEPSLEEP:
         // USER C
         // USER C
         // USER C
    case WATCHDOG:
         // USER C
         // USER C
    case SOFTWARE_RESET:
        // the last reset was an internally
        // triggered software reset
```

MSG: Workspace: Target configuration file C:\TC1130\TC1130.cfg used.
MSG: Controller0.Core: UDE\ComDev: TriCore ATAG/OCDS Debug Protocol, V3.6.4, ID 3 opened.
MSG: Controller0.Core: UDE\MemTool: FLASH programming for device '32MHz External Flash' ready.
MSG: Controller0.Core: UDE\DebugServer: Connection to TC1130 target monitor established: TriCore (a)
MSG: Controller0.Core: UDE\DebugServer: Program with ID 0x1  - code size 3051 bytes was loaded.
MSG: Controller0.Core: Flash: Program sections succeeded.

Application Note 191 V2.0, 2007-04
Insert 0xA0000000 <ENTER>
As you can see:
The "external-boot-memory-configuration-word" value (0x0000806D) is now @ 0xA0000010:

- Correct Value (0x0000806D)
- Correct Location (0xA0000010)
- Wrong Value (0x0000806D)
- Wrong Location (0xA0000004)

⇒ The “external-boot-memory-configuration-word” value (0x0000806D) is now @ 0xA0000010!
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:
The application runs with the Debugger!
The application runs after Power-ON !!!
We learnt:

1.) The debugger cares for the correct initialization of the EBU.
2.) If we start after "Power-On" without the debugger (Power-on) we have to take care of the settings ourselves.
3.) We have to change the start-up-file (cstart.asm) to correct the External-Boot-Memory-Configuration-Word.
4.) How to create a workaround.
5.) How to use the pls-debugger for debugging (memory-view).
OnBoardSDRAM-Pattern-Test:

Start Tasking EDE and open the project:

File – Open Project Space

Look in: select C:\TC1130
File name: select TC1130.psp

Open

Insert your application specific program:

Double click: Main.h and insert the following Defines:

```c
#define SDRAM_MAX 65000000  // 64 MBytes-On-Board-SDRAM ( max. 67.108.864 Bytes )
```
Double click: MAIN.C and insert Global Variables:

```
#pragma noclear
char cArray[SDRAM_MAX]; // 64 MBytes-On-Board-SDRAM ( max. 67.108.864 Bytes )
#pragma clear
```
Double click: MAIN.C and insert the following code in the main function:

```c
register unsigned int index=0;
char mb1[200]; // message buffer1 for sprintf()
```
Double click: MAIN.C and insert the following code in the main function:

```c
//****ramtest******************************************************************************
sprintf(mb1,"testing 64 MBytesOnBoardSDRAM at 0x%08x, pattern = 1010 B ...
\r\n",cArray);
myprintf(mb1);

for(index=0; index<SDRAM_MAX; index++)
cArray[index]=10; // 1010 b = 0xA = 10 d
for(index=0; index<SDRAM_MAX; index++)
{
    if(cArray[index]==10);
    else
    {
        myprintf("OnBoardSDRAM ERROR !!!\r\n");
        while(1){} //Loop_For_Ever
    }
}

sprintf(mb1,"testing 64 MBytesOnBoardSDRAM at 0x%08x, pattern = 0101 B ...
\r\n",cArray);
myprintf(mb1);

for(index=0; index<SDRAM_MAX; index++)
cArray[index]=5; // 01010 b = 0x5 = 5 d
for(index=0; index<SDRAM_MAX; index++)
{
    if(cArray[index]==5);
    else
    {
        myprintf("OnBoardSDRAM ERROR !!!\r\n");
        while(1){} //Loop_For_Ever
    }
}
sprintf(mb1,"64 MBytesOnBoardSDRAM at 0x%08x: (%u Byte(s)) ...
\r\n",cArray,SDRAM_MAX); // Write formatted data to string mb
myprintf(mb1);
//******************************************************************************
```
Note:

Testing 65.000.000 Bytes take about 11 minutes 13 seconds. Therefore we suggest to make the array (cArray) smaller (e.g. `#define SDRAM_MAX 100`).
Configure Linker – Control:

**Project – Project Options**

**Linker:** Map File: check/click  ✓ Memory usage info

**OK**
Generate your application program:

Build
Rebuild

or

Execute 'Rebuild' command.
See the Memory Usage:

Now you can close both your project and Tasking EDE:

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

Start pls-Debugger

File – Open Workspace

Look in: select C:\TC1130
File name: select TC1130.wsp

Open
Cancel

File – Load Program

Look in: select TC1130
File name: select TC1130.elf

Open

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:
We recommend now to **copy and store** your project-directory “C:\TC1130” to “05_TC1130_OnBoardFlash+OnBoardSDRAM+Memorytest”:
6.) Using of the TASKING - EDE Development Tools:

Write programs
for execution from Intel’s 32 Mbytes OnBoardFlash (BURST-Mode)
Start Tasking EDE and open the project:

File – Open Project Space

Look in: select C:\TC1130
File name: select TC1130.psp

Open
Configure Compiler, Assembler, Linker, Locator and Build – Control:

Project – Project Options

**Linker**: Script File: Defines/Stack/Heap: Heap start address: insert 0xA4000000

![Screenshot of TASKING EDE TriCore VX-toolset Project Options][1]

**OK**
Insert your application specific program:

**Double click: Main.c** and change Global Variable **menu**

**From**

```c
const char menu[] =
  "\r\n\n\n\n"
  "Program execution out of 32 MBytes OnBoardFlash\r\n"
  "===============================================\r\n"
  "1 ... LED IO_Port_0_Pin_7 ON\r\n"
  "2 ... LED IO_Port_0_Pin_7 OFF\r\n"
  "3 ... LED IO_Port_0_Pin_7 blinking\r\n"
  "\r\n";
```

**to**

```c
const char menu[] =
    "\r\n\n\n\n"
  "Program execution out of Intel's 32 MBytes OnBoardFlash ( Burst Mode !!! )\r\n"
  "==================================================================
    ========\r\n"
    "1 ... LED IO_Port_0_Pin_7 ON\r\n"
    "2 ... LED IO_Port_0_Pin_7 OFF\r\n"
    "3 ... LED IO_Port_0_Pin_7 blinking\r\n"
    "\r\n";
```
```c
const char menu[] = "*[ * * * * * * * ]*
"Program execution out of Intel's 32 Mbytes Bootstrap Flash & Burst Mode !!! It's"
"*[ * * * * * * * ]*
"1 ... LED _IO_Port_C_Fan7_0On
"2 ... LED _IO_Port_C_Fan7_0Off
"3 ... LED _IO_Port_C_Fan7_blinking
"*[ * * * * * * * ]*

const char question[] = "your choice: ";

const char message1[] = "[* * * * * * * * *]
"LED is OFF 
*[ * * * * * * * * *]

const char message2[] = "[* * * * * * * * *]
"LED is ON 
*[ * * * * * * * * *]

const char message3[] = "[* * * * * * * * *]
"LED is BLINKING 
*[ * * * * * * * * *]

volatile int _SK3200_count = 0;
volatile unsigned int blinkingOn;
char select = 0;
```

```c
char _cyclic[256]; // 4x 32bytes-On-Board-SDRAM ( max. 67.108.964 bytes )

// USER CODE END

// Internal Prototypes
```
Double click: Main.c: insert application-specific-program:

```c
// the CPU interrupt system is globally disabled:
DISABLE();
// 32-MBytes-On-Board-Burst-Flash switch to Burst-Mode
switch_to_burst();
// the CPU interrupt system is globally enabled
ENABLE();
```

```c
// functioncall(index): index=SUCCESS_MAX: index=:
{
    if (array[index] == 5) {
        // insert error message:
        fprintf(stderr, "Error!
" );
        while(1) // Loop for Error
    }
    fprintf(stdout, "Write formatted ...
" );
    // insert code here...

    // the CPU interrupt system is globally disabled:
    DISABLE();
    // 32-MBytes-On-Board-Burst-Flash switch to Burst-Mode
    switch_to_burst();
    // the CPU interrupt system is globally enabled
    ENABLE();
}
```
File – New

Insert `switch_to_burst_mode.c`

```c
if (Array[index] == 1) {
    // code
} else {
    // code
}
```

OK
**DESCRIPTION : Activate Burst Mode**

**COPYRIGHT   : (c) 2006 Infineon Technologies AG**

**AUTHOR     : Holger Dienst (AIM MC ATV AE)**

**changed/used by : Wilhelm Brezovits, April/May 2007**

---

```c
#include <stdio.h>
#include <stdlib.h>
#include "MAIN.h"
#include "ebu.h"
#include "types.h"

void external_switch_program(void);

// Note:
// The routine which switches the Flash
// (external_switch_program) cannot be executed from the Flash.
// Therefore, this must be copied into the RAM/heap (by switch_to_burst) and
// executed there.

unsigned char mb2[200]; // message buffer2 for sprintf()

void external_switch_program(void)
{
    EBU       *psEBU;
    u_int32  *puiAdr;
    u_int32   uiCS0_Base;

    psEBU = (EBU *)(EBU_BASE);
    uiCS0_Base = psEBU->ADDSEL0;
    uiCS0_Base &= EBUBASE_MSK;

    // enable burst mode on the flash
    puiAdr = (u_int32 *)(uiCS0_Base + (0x28C2<<2));
    *puiAdr = 0x00600060;
    puiAdr = (u_int32 *)(uiCS0_Base + (0x28C2<<2));
    *puiAdr = 0x00030003;

    // setting BUSCON0
    // setting AGEN to BURST0 and WAIT to synchronous wait
    psEBU->BUSCON0 &= ~(EBUAGEN_MSK | EBUWAIT_MSK);
    psEBU->BUSCON0 | = EBUAGEN_BURST0 | EBUWAIT_SYN;
    // setting BUSAP0
    // setting WAITRDC to 1 (more waitstates are inserted via WAIT signal)
    psEBU->BUSAP0 &= ~(EBUWAITRDC_MSK);
    psEBU->BUSAP0 |= 0x00400000;
    __isync();
    __dsync();

    // we make a dummy read, to make absolutely sure that our EBU settings are
done, if we return
```
uiCS0_Base = psEBU->BUSAP0;
}

void switch_to_burst(void)
{
    u_int32 *puiSource;
    u_int32 *puiDestination;
    u_int32 *puiDataHeap;
    unsigned int byte_count = (u_int32)&switch_to_burst -
    (u_int32)&external_switch_program;

    puiDataHeap = malloc((u_int32)&switch_to_burst -
    (u_int32)&external_switch_program);
    if ( puiDataHeap == NULL )
    {
        myprintf("ERROR: Insufficient memory (heap) available !!!\r\n");
        while(1){} //Loop_For_Ever
    }
    else
    {
        puiDestination = puiDataHeap;
        sprintf(mb2, "%u Byte(s) at 0x%08x (heap)
allocated,\r\n", byte_count, puiDataHeap); //Write formatted data to string mb2
        myprintf(mb2);
        byte_count = 0;
        // first copy all codes to external sdram
        myprintf("Copy Program into heap\r\n");
        for(puiSource = (u_int32 *)&external_switch_program; puiSource <
        (u_int32 *)&switch_to_burst; puiSource++)
        {
            *puiDestination = *puiSource;
            puiDestination++;
            byte_count++;
        }
        sprintf(mb2, "%u*4=%u Byte(s) copied into
heap\r\n", byte_count, byte_count*4);
        myprintf(mb2);
        puiDestination = puiDataHeap;
        myprintf("JUMP to heap\r\n");
        ((void(*)(void))puiDestination)(); // jump to external ram (heap)
        free(puiDataHeap);
        myprintf("BACK from heap, Burst-Mode activated\r\n");
    }
}
File – Save

(Project Window File View) – TC1130 (Files) – right mouse button click – Add Existing Files – Browse

Select switch_to_burst_mode.c
Open
OK
void external_switch_program(void);

/*
 * Notes:
 * Remarks:
 * Das Problem ist, dass die Routine die das Flash verändert.
 * (external_switch_program) nicht aus dem Flash laufen kann.
 * Daher muss diese erst den RAM/Heap exportiert (mittels switch_to_burst)
 * und dort ausgeführt werden.
 * English:
 * The routine which switches the Flash
 * (external_switch_program) cannot be executed from the Flash.
 * Therefore, this must be copied into the RAM/heap (by switch_to_burst) and
 * executed there.
 */

unsigned char msg[500]; // message buffers for sprintf()
Double click: Main.h and insert Prototype of Global Function:

```c
extern void switch_to_burst(void);
```
File – New

Insert types.h

OK

Insert code:

```c
/************************************************************************
*                                                                       *
*  FILE        :  TYPES.H                                               *
*                                                                       *
*  DESCRIPTION :  Setting names for types                               *
*                                                                       *
*  COPYRIGHT   :  (c) 1999 Infineon Technologies                        *
*                                                                       *
*  AUTHOR      :  Holger Dienst (AI MC AE)                              *
*                                                                       *
************************************************************************/

#ifndef _TYPES_H
#define _TYPES_H

typedef unsigned int u_int32;
typedef int int32;
typedef unsigned short u_int16;
typedef short int16;
typedef unsigned char u_int8;
typedef char int8;
#endif  /* _TYPES_H */
```

File – Save
FILE – New
Insert EBU.h
OK
Insert code:

```c
/*************************************************************************/
/* FILE : EBU.H */
/* DESCRIPTION : Structur and Bitsettings for EBU TC1130 */
/* COPYRIGHT : (c) 2002 Infineon Technologies AG */
/* AUTHOR : Holger Dienst (AI MC MA TM) */
/* VERSION : 1.00 */
/* CHANGES : */
/*************************************************************************/
#ifndef _EBU_H
#define _EBU_H
#define EBU_BASE                (0xF8000000)
/* Access Mode:
R  - Read-only register.
32 - Only 32-bit word accesses are permitted to that register/address range.
E  - Endinit protected register/address.
PW - Password protected register/address.
For more details refer to specification.
*/
typedef struct ebu
{
    volatile unsigned int CLC;           /* Clock Control Register (E,32) */
    volatile unsigned int RESERVED0[1];  /* Reserved */
    volatile unsigned int ID;            /* Identification Register (R) */
    volatile unsigned int RESERVED1[1];  /* Reserved */
    volatile unsigned int CON;           /* Global Control Register (32) */
    volatile unsigned int RESERVED2[3];  /* Reserved */
    volatile unsigned int BFCON;         /* Burst Flash Control Register (32) */
    volatile unsigned int RESERVED3[7];  /* Reserved */
    volatile unsigned int SDRMREF0;      /* SDRAM Type 0 Refresh Control */
    volatile unsigned int RESERVED4[1];  /* Reserved */
    volatile unsigned int SDRMREF1;      /* SDRAM Type 1 Refresh Control */
    volatile unsigned int RESERVED5[1];  /* Reserved */
    volatile unsigned int SDRMCON0;      /* SDRAM Type 0 Configuration */
    volatile unsigned int RESERVED6[1];  /* Reserved */
    volatile unsigned int SDRMCON1;      /* SDRAM Type 1 Configuration */
    volatile unsigned int RESERVED7[1];  /* Reserved */
    volatile unsigned int SDRMOD0;       /* SDRAM Type 0 Mode */
    volatile unsigned int RESERVED8[1];  /* Reserved */
    volatile unsigned int SDRMOD1;       /* SDRAM Type 1 Mode */
    volatile unsigned int RESERVED9[1];  /* Reserved */
    volatile unsigned int SDRSTAT0;      /* SDRAM Type 0 Status */
    volatile unsigned int RESERVED10[1]; /* Reserved */
    volatile unsigned int SDRSTAT1;      /* SDRAM Type 1 Status */
    volatile unsigned int RESERVED11[1]; /* Reserved */
    volatile unsigned int ADDSEL0;       /* Address Select Register 0 (32) */
    volatile unsigned int RESERVED12[1]; /* Reserved */
    volatile unsigned int ADDSEL1;       /* Address Select Register 1 (32) */
    volatile unsigned int RESERVED13[1]; /* Reserved */
    volatile unsigned int ADDSEL2;       /* Address Select Register 2 (32) */
    volatile unsigned int RESERVED14[1]; /* Reserved */
    volatile unsigned int ADDSEL3;       /* Address Select Register 3 (32) */
    volatile unsigned int RESERVED15[9]; /* Reserved */
    volatile unsigned int BUSCON0;       /* Bus Configuration Register 0 (32) */
    volatile unsigned int RESERVED16[1]; /* Reserved */
    volatile unsigned int BUSCON1;       /* Bus Configuration Register 1 (32) */
    volatile unsigned int RESERVED17[1]; /* Reserved */
    volatile unsigned int BUSCON2;       /* Bus Configuration Register 2 (32) */
    volatile unsigned int RESERVED18[1]; /* Reserved */
    volatile unsigned int BUSCON3;       /* Bus Configuration Register 3 (32) */
    volatile unsigned int RESERVED19[9]; /* Reserved */
} ebu;
#endif
```
AP32117
TC1130 "Cookery-Book"

volatile
volatile
volatile
volatile
volatile
volatile
volatile
volatile
volatile
volatile
volatile
volatile
volatile
volatile
volatile
volatile
volatile

unsigned
unsigned
unsigned
unsigned
unsigned
unsigned
unsigned
unsigned
unsigned
unsigned
unsigned
unsigned
unsigned
unsigned
unsigned
unsigned
unsigned

int
int
int
int
int
int
int
int
int
int
int
int
int
int
int
int
int

BUSAP0;
/*
RESERVED20[1]; /*
BUSAP1;
/*
RESERVED21[1]; /*
BUSAP2;
/*
RESERVED22[1]; /*
BUSAP3;
/*
RESERVED23[17];/*
EMUAS;
/*
RESERVED24[1]; /*
EMUBC;
/*
RESERVED25[1]; /*
EMUBAP;
/*
RESERVED26[1]; /*
EMUOVL;
/*
RESERVED27[5]; /*
USERCON;
/*

Bus access parameter Register 0 (32) */
Reserved */
Bus access parameter Register 1 (32) */
Reserved */
Bus access parameter Register 2 (32) */
Reserved */
Bus access parameter Register 3 (32) */
Reserved */
Emulator Memory Address Select Register (32) */
Reserved */
Emulator Memory Bus Configuration Register (32) */
Reserved */
Emulator Memory access parameter (32) */
Reserved */
Overlay memory chip-select generation (32) */
Reserved */
Test/Control Configuration Register (32) */

} EBU;
/* Global Control Register
#define EBUEXTLOCK
#define EBUARBSYNC
#define EBUARBMODE_MSK
#define EBUARBMODE_EBUDIS
#define EBUARBMODE_EXTM
#define EBUARBMODE_EXTS
#define EBUARBMODE_NOARB
#define EBUTIMOUTC_MSK
#define EBUGLOBAL_CS_MSK
#define EBUGLOBAL_CS0
#define EBUGLOBAL_CS1
#define EBUGLOBAL_CS2
#define EBUGLOBAL_CS3
#define EBUBUSCLK_MSK
#define EBUBUSCLK_1
#define EBUBUSCLK_2
#define EBUBUSCLK_4
#define EBUSDCMSEL
#define EBUCS0FAM
#define EBUEMUFAM
#define EBUBFSSS

*/
0x00000010
0x00000020
0x000000C0
0x00000000
0x00000040
0x00000080
0x000000C0
0x0000FF00
0x00FF0000
0x00010000
0x00020000
0x00040000
0x00080000
0x03000000
0x00000000
0x01000000
0x02000000
0x04000000
0x08000000
0x10000000
0x20000000

/* Burst Flash Control Register */
#define EBUFETBLEN0_MSK 0x0000000F
#define EBUFETBLEN0_1 0x00000000
#define EBUFETBLEN0_2 0x00000001
#define EBUFETBLEN0_4 0x00000002
#define EBUFETBLEN0_8 0x00000003
#define EBUFBBMSEL0
0x00000010
#define EBUWAITFUNC0
0x00000020
#define EBUEXTCLK_MSK 0x000000C0
#define EBUEXTCLK_1
0x00000000
#define EBUEXTCLK_2
0x00000040
#define EBUEXTCLK_3
0x00000080
#define EBUEXTCLK_4
0x000000C0
#define EBUBFCMSEL
0x00000100
#define EBUEBSE0
0x00000200
#define EBUDBA0
0x00000400
#define EBUFDBKEN
0x00000800
#define EBUDTALTNCY
0x0000F000
#define EBUFETBLEN1_MSK 0x000F0000
#define EBUFETBLEN1_1 0x00000000
#define EBUFETBLEN1_2 0x00010000
#define EBUFETBLEN1_4 0x00020000
#define EBUFETBLEN1_8 0x00030000
#define EBUFBBMSEL1
0x00100000
#define EBUWAITFUNC1
0x00200000
#define EBUEBSE1
0x02000000
#define EBUDBA1
0x04000000

/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*
/*

EBU
EBU
EBU
EBU
EBU
EBU
EBU
EBU
EBU
EBU
EBU
EBU
EBU
EBU
EBU
EBU
EBU
EBU
EBU
EBU
EBU

External Bus Lock Control */
Arbitration Inputs Evaluation Control */
Arbitration Strategy */
is Disabled */
is External Master */
is External Slave */
Arbitration is Disabled */
Time Out Control */
chip select global mask */
chip select global 0 */
chip select global 1 */
chip select global 2 */
chip select global 3 */
bus clock */
bus clock = LMB clock */
bus clock = LMB clock/2 */
bus clock = LMB clock/4 */
SDRAM clock gated */
CS0 Fills Address Map */
CSemu Fills Address Map */
Burst Flash Single Stage Synchronisation */

Fetch burst length Type 0 */
Fetch burst length 1 data access Type 0 */
Fetch burst length 2 data access Type 0 */
Fetch burst length 4 data access Type 0 */
Fetch burst length 8 data access Type 0 */
Flash burst buffer mode select Type 0 */
Operation of /WAIT input Type 0 */
BFCLK external clock */
BFCLK = LMBCLK */
BFCLK = LMBCLK/2 */
BFCLK = LMBCLK/3 */
BFCLK = LMBCLK/4 */
BFCLK only present during burst access */
ADV and BAA not 1/2LMBCLK delayed Type 0 */
Disable Burst Address Wrapping Type 0 */
Burst Clock Feedback Enable */
Latency Cycle Control */
Fetch burst length Type 1 */
Fetch burst length 1 data access Type 1 */
Fetch burst length 2 data access Type 1 */
Fetch burst length 4 data access Type 1 */
Fetch burst length 8 data access Type 1 */
Flash burst buffer mode select Type 1 */
Operation of /WAIT input Type 1 */
ADV and BAA not 1/2LMBCLK delayed Type 1 */
Disable Burst Address Wrapping Type 1 */

/* SDRAM Refresh Register */
#define EBUREFRESHC_MSK
0x0000003F
#define EBUREFRESHR_MSK
0x000001C0
#define EBUSELFREXST
0x00000200
#define EBUSELFREX
0x00000400
#define EBUSELFRENST
0x00000800
#define EBUSELFREN
0x00001000
#define EBUAUTOSELFR
0x00002000

/*
/*
/*
/*
/*
/*
/*

Refresh Counter Period Value */
Number of Refresh Commands */
Self refresh exit status */
Self refresh exit */
Self refresh entry status */
Self refresh entry */
Automatic Self Refresh (only SDRMREF0) */

/* SDRAM Configuration */
#define EBUCRAS_MSK
#define EBUCRFSH_MSK
#define EBUCRSC_MSK
#define EBUCRP_MSK

/*
/*
/*
/*

Row Activate to Precharge Cycle Counter */
Refresh Commands Counter Value */
Mode Register Setup Time Counter Value */
Row Precharge Time Counter Value */

Application Note

0x0000000F
0x000000F0
0x00000300
0x00000C00

221

V2.0, 2007-04


endif /* RBO H */

File – Save
(Project Window File View) – TC1130 (Files) – right mouse button click – Add Existing Files – Browse

Select EBU.h
Select types.h
Generate your application program:

Build
Rebuild

or

Execute 'Rebuild' command.

Now you can close both your project and Tasking EDE:

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

Start pls-Debugger

File – Open Workspace

Look in: select C:\TC1130
File name: select TC1130.wsp

Open
Cancel

File – Load Program

Look in: select TC1130
File name: select TC1130.elf

Open

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:
We recommend now to copy and store your project-directory “C:\TC1130” to “06_TC1130_OnBoardFlash-BURST-MODE+OnBoardSDRAM+Memorytest”: 
7.) Time – Measurement (Using an oscilloscope / a logic analyzer):
Insert application specific program (“time-measurement”, see below) into the following programming examples:

**Chapter 4**: Program\_Execution\_From\_PMI\_Scratch\_Pad\_RAM (PMI\_SPRAM)
**Chapter 5**: Program\_Execution\_From\_OnBoardProgramFlash
**Chapter 6**: Program\_Execution\_From\_OnBoardProgramFlash\_Burst\_Mode

**Double click: Main.c** and **change** Global Variable menu from

```c
volatile unsigned int blinking=ON;
```

to

```c
volatile unsigned int blinking=OFF;
```

**Double click: Main.c** and **insert** Local Variables into “void main (void)”

```c
register unsigned int i = 0;
register unsigned int j = 0;
```

**Double click: Main.c** and **insert** the following endless loop (before “while (1)”):

```c
// - the CPU interrupt system is globally disabled
DISABLE();

while(1) // endless loop
{
    IO\_vTogglePin(IO\_P0\_7);
    // time-consuming, dummy operations
    for (i=0; i<=100000; i++)
    {
        for (j=2; j<=5; j++)
        {
            __nop(); __nop(); __nop(); __nop(); __nop(); __nop();
            __nop();
            __nop(); __nop(); __nop(); __nop(); __nop(); __nop();
            __nop();
        }
    }
    __nop(); __nop(); __nop(); __nop(); __nop(); __nop(); __nop();
    __nop(); __nop(); __nop(); __nop(); __nop(); __nop();
    __nop(); __nop();
}
```
The frequency/time of the toggling pin “IO_Port_0_Pin_7” is a “value” for the speed of the application:

Program execution sorted by speed:

<table>
<thead>
<tr>
<th>Program execution out of</th>
<th>Time [s] $^1$</th>
<th>Frequency [Hz] $^2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>PMI_SPRAM</td>
<td>0.104</td>
<td>9.60</td>
</tr>
<tr>
<td>OnBoardFlash (Burst Mode)</td>
<td>1.086</td>
<td>0.92</td>
</tr>
<tr>
<td>OnBoardFlash</td>
<td>2.265</td>
<td>0.44</td>
</tr>
</tbody>
</table>

$^1$ … lower is better  
$^2$ … higher is better
We recommend now to **copy and store** your project-directories “C:\TC1130” to “04_TC1130+Time-Measurement-using-a-scope” and “05_TC1130+Time-Measurement-using-a-scope” and “06_TC1130+Time-Measurement-using-a-scope”:

![Image of file browser](image-url)
8.) Time – Measurement [Using the SystemTimer (STM)]

8.1.) Creating a Software-Clock Using the STM-Interrupt

**Note:**
We are going to use the “Software-Clock Using the STM-Interrupt” to check everything concerning “time-measurement” in chapter 8.2.
We recommend now to copy and store
“06_TC1130_OnBoardFlash-BURST-MODE+OnBoardSDRAM+Memorytest”
to your working project-directory “C:\TC1130”:

Start Tasking EDE and open the project:

File – Open Project Space

Look in: select C:\TC1130
File name: select TC1130.psp
Open
Insert your application specific program:

Double click: Main.c and change Global Variable menu
From

```c
const char menu[] =
"\r\n\n\n\n"
"Program execution out of Intel's 32 MBytes OnBoardFlash ( Burst Mode !!! )\r\n"
"==================================================================
========
"
"1 ... LED IO_Port_0_Pin_7 ON\r\n"
"2 ... LED IO_Port_0_Pin_7 OFF\r\n"
"3 ... LED IO_Port_0_Pin_7 blinking\r\n"
" \r\n";
```

to

```c
const char menu[] =
"\r\n\n\n\n"
"Program execution out of Intel's 32 MBytes OnBoardFlash ( Burst Mode !!! )\r\n"
"==================================================================
========
"
"1 ... LED IO_Port_0_Pin_7 ON\r\n"
"2 ... LED IO_Port_0_Pin_7 OFF\r\n"
"3 ... LED IO_Port_0_Pin_7 blinking\r\n"
"4 ... Set   The Software Clock Using The STM-Interrupt \r\n"
"5 ... Start The Software Clock Using The STM-Interrupt \r\n"
"6 ... Stop  The Software Clock Using The STM-Interrupt \r\n"
"7 ... Show  The Software Clock Using The STM-Interrupt \r\n"
"z ... Back To Main Menu  \r\n"
" \r\n";
```
// USER CODE BEGIN

DECLARE Variables

// USER CODE BEGIN (AfKl_General,1)

const char menu[] = "Listening!

"Program execution out of Intel's 32 MBytes OnBoardFlash ( Burst Mode !!! )!

"================================================================================

"0 ... LED 3G_Port_G_Pin7 On/Off"

"1 ... LED 3G_Port_G_Pin9 On/Off"

"2 ... LED 3G_Port_G_Pin4 Blinking"

"3 ... Start The Software Clock Using The STM-Interrupt" //

"4 ... Stop The Software Clock Using The STM-Interrupt" //

"5 ... Show The Software Clock Using The STM-Interrupt" //

"6 ... Back To Main Menu" //

" 
"

const char question[] = "your choice ?

const char message1[] = "0: LED is On \x1b[31m\x1b[40m"

const char message2[] = "1: LED is Off \x1b[37m\x1b[40m"

const char message3[] = "2: LED is Blinking \x1b[32m\x1b[40m"

volatile int doesWait=0;

volatile unsigned int blinking=0;

char select;"
Double click: Main.c and change Code ["char input (void)" - Function]

From

```c
} while (in!= '1' && in!= '2' && in != '3');
```

To

```c
} while (in!= '1' && in!= '2' && in != '3' && in != '4' && in != '5' && in != '6' && in != '7');
```
Double click: Main.h and insert the following Defines:

```c
#define FALSE 0
#define TRUE 1
```
Double click: Main.c insert User Code (Global Variables):

```c
volatile unsigned int STM_Clock_Running = FALSE;
```
Double click: Main.h and insert Global Variables (Extern Declarations):

```c
extern volatile unsigned int STM_Clock_Running;
```
Double click: Main.c insert User Code (Global Variables):

```c
const char message4[] = 
"\n\r*** STM Clock is RUNNING ***\r\n";

const char message5[] = 
"\n\r*** STM Clock is STOPPED ***\r\n";
```
Double click: Main.c insert User Code [ void main (void) - Function ]:

```c
    case '4': Set_STM_Clock(); break; // Set STM clock
    case '5': if (STM_Clock_Running == FALSE) STM_Clock_Running=TRUE; myprintf(message4); break; // Start STM clock
    case '6': if (STM_Clock_Running == TRUE) STM_Clock_Running=FALSE; myprintf(message5); break; // Stop STM clock
    case '7': Show_STM_Clock(); break; // Show STM clock
```

```c
while (1) {
    myprintf(message); 
    select_user();

    switch (select) {
    case '1': blinking=0; ID_PA7_LED=0; myprintf(message1); break;
    case '2': blinking=0; ID_PA7_LED=0; myprintf(message2); break;
    case '3': blinking=0; myprintf(message3); break;
    case '4': Set_STM_Clock(); break; // Set STM clock
    case '5': if (STM_Clock_Running == FALSE) STM_Clock_Running=TRUE; myprintf(message4); break; // $1,
    case '6': if (STM_Clock_Running == TRUE) STM_Clock_Running=FALSE; myprintf(message5); break; // $2,
    case '7': Show_STM_Clock(); break; // Show STM clock
    }

    // USER CODE END
    return(0); // End of function main
}
```

```
USER CODE BEGIN (USER1, USER2)
```

Double click: Main.h and insert Prototypes of Global Functions (Extern Declaration):

```c
extern void Set_STM_Clock(void);
extern void Show_STM_Clock(void);
```
Double click: Main.h and insert Typedefs (struct Definition):

```c
struct clock
{
    unsigned int day;
    unsigned char hour;
    unsigned char minute;
    unsigned char second;
};
```

```c
// USER CODE BEGIN (MAIN_Reader.A)
struct clock
{
    unsigned int day;
    unsigned char hour;
    unsigned char minute;
    unsigned char second;
};
// USER CODE END
```

```
// Exported Global Variables
```

```c
```
Double click: Main.c insert User Code (Global Variables):

```c
volatile struct clock STM_Clock;
```
Double click: Main.h and insert Extern Declaration:

```c
extern volatile struct clock STM_Clock;
```
File – New
Insert STM_Clock.c

OK
Insert User Code:

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

void Set_STM_Clock(void)
{
  char in=' ';
  char help_hour[3];
  char help_minute[3];
  char help_second[3];
  char mb[200]; // message buffer for sprintf()


  STM_Clock.day=STM_Clock.hour=STM_Clock.minute=STM_Clock.second=0;
  myprintf("\r\nTime = ? (Syntax: hhmmss, e.g.: 051207) = ");

  while (!ASC0_RSRC_SRR) ;
  ASC0_RSRC_CLRR=1;    // Clear SRR bit
  help_hour[0] = (unsigned char)ASC0_RBUF;
  while (!ASC0_RSRC_SRR) ;
  ASC0_RSRC_CLRR=1;    // Clear SRR bit
  help_hour[1] = (unsigned char)ASC0_RBUF;
  STM_Clock.hour   = (unsigned char)atoi(help_hour);

  while (!ASC0_RSRC_SRR) ;
  ASC0_RSRC_CLRR=1;    // Clear SRR bit
  help_minute[0] = (unsigned char)ASC0_RBUF;
  while (!ASC0_RSRC_SRR) ;
  ASC0_RSRC_CLRR=1;    // Clear SRR bit
  help_minute[1] = (unsigned char)ASC0_RBUF;
  STM_Clock.minute   = (unsigned char)atoi(help_minute);

  while (!ASC0_RSRC_SRR) ;
  ASC0_RSRC_CLRR=1;    // Clear SRR bit
  help_second[0] = (unsigned char)ASC0_RBUF;
  while (!ASC0_RSRC_SRR) ;
  ASC0_RSRC_CLRR=1;    // Clear SRR bit
  help_second[1] = (unsigned char)ASC0_RBUF;
  STM_Clock.second   = (unsigned char)atoi(help_second);

  sprintf(mb,"\r\n*** Time = %02u:%02u:%02u
  ***\r\n",STM_Clock.hour,STM_Clock.minute,STM_Clock.second);
  myprintf(mb);
}

void Show_STM_Clock(void)
{
  char mb[200]; // message buffer for sprintf()
  unsigned char in;

  myprintf("\r\n
***** TIME *****
");
  do
    in = (unsigned char)ASC0_RBUF;
  while (in != '\n');
  ASC0_RSRC_CLRR=1; // Clear SRR bit

  mb[200] = (unsigned char)in;
  while (mb[200] != '\n')
    ASC0_RSRC_CLRR=1; // Clear SRR bit
    mb[200] = (unsigned char)ASC0_RBUF;

  myprintf(mb);
}
```
{  sprintf(mb,"*** %02u:%02u:%02u
 ***\r",STM_Clock.hour,STM_Clock.minute,STM_Clock.second);
  myprintf(mb);
   in = (unsigned char)ASC0_RBUF;
 }while (in!='z' && in!='Z');
File
Save all

(Project Window File View) – TC1130 (Files) – right mouse button click – Add Existing Files – Browse
Select STM_Clock.c

Open - OK
Double click: STM.c insert User Code for system-timer's interrupt-service-routine:

```c
if (STM_Clock_Running)
{
    if (++STM_Clock.second==60)
    {
        STM_Clock.second=0;
        if (++STM_Clock.minute==60)
        {
            STM_Clock.minute=0;
            if (++STM_Clock.hour==24)
            {
                STM_Clock.hour=0, ++STM_Clock.day;
            }
        }
    }
}
```

```c
void __interrupt [STM_INT3] STM_WWDRH(void)
{
    // USER CODE BEGIN
    // USER CODE END

    if(CMPR_CMPD0 == 1) // if compare match of CMP0 is pending
    {
        USER CODE BEGIN (SWAP 3)
        STM_CMP0=STM_CMP0+75000000; // 76,000,000 * 23,333 ns = 1 s
        USER CODE END
        if(R8232_wait())
          R8232_wait();
    }
    // USER CODE BEGIN
    if(blinking)
      IO_Vout=pin[10_P1_7];
    // USER CODE END
    if (STM_Clock_Running)
    {
        if (++STM_Clock.second==60)
        {
            STM_Clock.second=0;
            if (++STM_Clock.minute==60)
            {
                STM_Clock.minute=0;
                if (++STM_Clock.hour==24)
                {
                    STM_Clock.hour=0, ++STM_Clock.day;
                }
            }
        }
    }

    // USER CODE END

    // clear request bit of CMP0
    ```
Generate your application program:

Build
Rebuild

or

Execute 'Rebuild' command.

Now you can close both your project and Tasking EDE:

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

Start pls-Debugger

File – Open Workspace

Look in: select C:\TC1130
File name: select TC1130.wsp

Open

Load

Click  Program All

Exit
Exit
// USER CODE END

sword main(void)
{
sword swReturn;

// USER CODE BEGIN (Main.2)
register unsigned int index=0;
char mb[200]; // message buffer for sprintf()

// USER CODE END

swReturn = 0;

switch(RESET_INDICATOR)
{


}


MSG: Controller0.Core::UD2Command::TriCore STAG/SCDS Debug Protocol. V3.6.4, ID 3 opened
MSG: Controller0.Core::UDRamtset.: FLASH programming for device ‘32MB External Flash’ ready
MSG: Controller0.Core::UDEdebugServer. Connection to TC130 target monitor established. TriCore.
MSG: Controller0.Core::UDEdebugServer. Program with ID 0x1 - code size 8402 bytes was loaded.
MSG: Controller0.Core::Flash: Program sections succeeded

Command/
Views – Watch
Right mouse button click at “new variable” select Browse
Select “STM_Clock”
sword main (void) {
    sword swReturn;

    // USER CODE BEGIN (Main,2)
    register unsigned int index=0;
    char mb1[200];  // message buffer1 for spri
    // USER CODE END
    swReturn = 0;

    switch(RESET_INDICATOR) {
        case DEEPSLEEP_WAKEUP:  // the last
            // Hardware
            // (wake-up

            // USER CODE BEGIN (Main,3)
            // USER CODE END

    MSG: Controller0.Core::UAD2CommDev: TriCore iTAG/OCIDS Debug Protocol, V3.6.4, ID 3 opened
    MSG: Controller0.Core::UDMMember: FLASH programming for device '32MB External Flash' ready
    MSG: Controller0.Core::UDEDebugServer: Connection to TC1130 target monitor established. TriCore
    MSG: Controller0.Core::UDEDebugServer: Program with ID 0x1 - code size 8402 bytes was loaded!
    MSG: Controller0.Core::Flash: Program sections succeeded
Right mouse button click at “STM_Clock” select Refresh Period
Running target: **click** ✓ Enable refresh
Running target: **select** 250
Halted target: **click** ✓ Enable refresh
Halted target: **select** 300

OK
Execute any terminal-program
(9600 Baud, 8 bit Data, no Parity-Bit, 1 Stop-Bit, Xon/Xoff Protocol):
Debug – Start Program Execution

```
// USER CODE BEGIN (Main, 3)
```

```
// USER CODE END
```

MSG: Controller0.Core::UDMCommCore: TriCore UART/OCDS Debug Protocol, V3.6.4, ID 3 opened
MSG: Controller0.Core::UDMCommCore: Flash programming for device '32MB External Flash' ready
MSG: Controller0.Core::UDMDebugServer: Connection to TCI130 target monitor established. TriCore
MSG: Controller0.Core::UDMDebugServer: Program with ID 0x1 - code size 8402 bytes was loaded!
 MSG: Controller0.Core::Flash: Program sections succeeded

Start Program Execution Controller0.Core C:\TC1130\TC1130.cfg Controller0.Core halted by user break Function disabled
And see the result:

Program execution out of Intel's 32 MBytes OnBoard Flash (Burst Mode ???)
==============================================================================
1 ... LED IO_Port_0_Pin_7 ON
2 ... LED IO_Port_0_Pin_7 OFF
3 ... LED IO_Port_0_Pin_7 blinking
4 ... Set The Software Clock Using The STM-Interrupt
5 ... Start The Software Clock Using The STM-Interrupt
6 ... Stop The Software Clock Using The STM-Interrupt
7 ... Show The Software Clock Using The STM-Interrupt
8 ... Back To Main Menu

your choice: [ ]
Insert 7

```
Multi-threaded TTY
```

128 Byte(s) at 0x40000004 (heap) allocated
Copy Program into heap
32*4=128 Byte(s) copied into heap
JUMP to heap
BACK from heap, Burst-Mode activated

Program execution out of Intels 32 MBytes OnBoardFlash (Burst Mode !!!)
******************************************************************************
1 ... LED IO_Port_0_Pin_7 ON
2 ... LED IO_Port_0_Pin_7 OFF
3 ... LED IO_Port_0_Pin_7 blinking
4 ... Set The Software Clock Using The STM-Interrupt
5 ... Start The Software Clock Using The STM-Interrupt
6 ... Stop The Software Clock Using The STM-Interrupt
7 ... Show The Software Clock Using The STM-Interrupt
z ... Back To Main Menu

your choice: 7

```
**** TIME ****

00:00:00 0
```

Insert z
Insert 4

**Multi-threaded TTY**

- **Port**: COM1
- **Baud**: 9600
- **Parity**: None
- **Data Bits**: 8
- **Stop Bits**: 1

- **Flow Control**: None
- **Timeouts**: None

**Multi-threaded TTY**

6. Stop The Software Clock Using The STM-Interrupt
7. Show The Software Clock Using The STM-Interrupt
Z. Back To Main Menu

**Program execution out of Intel’s 32 MBytes OnBoardFlash ( Burst Mode !!!)**

1. LED IO_Port_0_Pin_7 ON
2. LED IO_Port_0_Pin_7 OFF
3. LED IO_Port_0_Pin_7 blinking
4. Start The Software Clock Using The STM-Interrupt
5. Stop The Software Clock Using The STM-Interrupt
6. Show The Software Clock Using The STM-Interrupt
7. Back To Main Menu

**your choice: your choice: 4**

**Program execution out of Intel’s 32 MBytes OnBoardFlash ( Burst Mode !!!)**

1. LED IO_Port_0_Pin_7 ON
2. LED IO_Port_0_Pin_7 OFF
3. LED IO_Port_0_Pin_7 blinking
4. Start The Software Clock Using The STM-Interrupt
5. Stop The Software Clock Using The STM-Interrupt
6. Show The Software Clock Using The STM-Interrupt
7. Back To Main Menu

**your choice: your choice: 4**

**Time = ? (Syntax: h:mma, e.g.: 051207) = **
Insert 164500

```
Multi-threaded TTY

---

5 ... Start The Software Clock Using The STM-Interrupt
6 ... Stop The Software Clock Using The STM-Interrupt
7 ... Show The Software Clock Using The STM-Interrupt
z ... Back To Main Menu

your choice: your choice: 4
Time = ? (Syntax: hhmms, e.g.: 051207) = 164500
*** Time = 16:45:00 ***

Program execution out of Intel’s 32 MBytes OnBoardFlash ( Burst Mode !!! )

1 ... LED IO_Port_0_Pin_7 ON
2 ... LED IO_Port_0_Pin_7 OFF
3 ... LED IO_Port_0_Pin_7 blinking
4 ... Set The Software Clock Using The STM-Interrupt
5 ... Start The Software Clock Using The STM-Interrupt
6 ... Stop The Software Clock Using The STM-Interrupt
7 ... Show The Software Clock Using The STM-Interrupt
z ... Back To Main Menu

your choice: 
```
4. Set the Software Clock Using the STM-Interrupt
5. Start the Software Clock Using the STM-Interrupt
6. Stop the Software Clock Using the STM-Interrupt
7. Show the Software Clock Using the STM-Interrupt
z. Back To Main Menu

Your choice: 5

*** STM Clock is RUNNING ***

Program execution out of Intel’s 32 MBytes OnBoardFlash (Burst Mode !!!)

1. LED IO_Port_0_Pin_7 ON
2. LED IO_Port_0_Pin_7 OFF
3. LED IO_Port_0_Pin_7 blinking
4. Set the Software Clock Using the STM-Interrupt
5. Start the Software Clock Using the STM-Interrupt
6. Stop the Software Clock Using the STM-Interrupt
7. Show the Software Clock Using the STM-Interrupt
z. Back To Main Menu

Your choice: 1

---

Application Note 270 V2.0, 2007-04
Insert 7

**Multi-threaded TTY**

<table>
<thead>
<tr>
<th>File</th>
<th>TTY</th>
<th>Transfer</th>
<th>Help</th>
</tr>
</thead>
<tbody>
<tr>
<td>Port</td>
<td>Baud</td>
<td>Parity</td>
<td>Data Bits</td>
</tr>
<tr>
<td>COM1</td>
<td>9600</td>
<td>None</td>
<td>8</td>
</tr>
</tbody>
</table>

---

7 ... Show The Software Clock Using The STM-Interrupt
z ... Back To Main Menu

**your choice: 5**

***STM Clock is RUNNING***

---

**Program execution out of Intel's 32 MBytes OnBoardFlash ( Burst Mode !!! )**

1. LED IO_Port_0_Pin_7 ON
2. LED IO_Port_0_Pin_7 OFF
3. LED IO_Port_0_Pin_7 blinking
4. Set The Software Clock Using The STM-Interrupt
5. Start The Software Clock Using The STM-Interrupt
6. Stop The Software Clock Using The STM-Interrupt
7. Show The Software Clock Using The STM-Interrupt
z ... Back To Main Menu

**your choice: 7**

**xxxxxx TIME xxxxxxxx**

**16:46:17**

---

Application Note 271 V2.0, 2007-04
Compare with your “computer time”:
After 1 day (Compare with your “computer time” and with your debugger):

**Terminal:**
- `pls - Debugger`
- `terminal-program`
- `Windows – Settings – Control Panel – Date/Time Properties`
Close the pls-Debugger:

File – Close Workspace

Yes

File – Exit
We recommend now to copy and store your project-directory “C:\TC1130” to “08.1_TC1130_Software-Clock_Using_STM-Interrupt”:
8.2.) Time-Lag-Measurement Using the STM

Note:
In this chapter we are going to use the STM for Time-Lag-Measurement (‘long-time’ and ‘short time’ lag-measurement).

8.2.1.) Time-Lag-Measurement Using the STM (example: “long-time-lag-measurement”)
Start Tasking EDE and open the project:

File – Open Project Space

Look in: select C:\TC1130
File name: select TC1130.psp

Open

Double click: Main.c insert User Code (Global Variables):

```
//Time-Lag-Measurement using STM
volatile union first_read first_read_STM;
volatile union second_read second_read_STM;
unsigned long long int result_time_lag_STM; // 1x64 Bit
```
Additional Information: System Timer (STM):

Note:
Because the STM is 56 bits wide, it is not possible to read its entire contents with one instruction. It must be read with two load instructions. Since the timer would continue to count between the two load operations, there is a chance that the two values read may not be consistent (due to possible overflow from the low part of the timer to the high part between the two read operations). To enable synchronous and consistent reading of the STM contents, a capture register (CAP), is implemented. It latches the contents of the high part of the STM each time one of the registers TIM0 to TIM5 is read. Thus, it holds the upper value of the timer at exactly the same time when the lower part is read. The second read operation then reads the contents of the CAP for the complete timer value.
Note:
The TriCore architecture defines the following fundamental data types:
- An 8-bit byte
- A 16-bit short
- A 32-bit word
- A 64-bit double word
The next picture shows the mapping between these fundamental data types and the C language data type:

<table>
<thead>
<tr>
<th>Type</th>
<th>Keyword</th>
<th>Size (bit)</th>
<th>Align (bit)</th>
<th>Ranges</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>bool</td>
<td>8</td>
<td>8</td>
<td>0 or 1</td>
</tr>
<tr>
<td>Character</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Integral</td>
<td>char</td>
<td>8</td>
<td>8</td>
<td>-128..127</td>
</tr>
<tr>
<td></td>
<td>signed char</td>
<td>8</td>
<td>8</td>
<td>0..255</td>
</tr>
<tr>
<td></td>
<td>unsigned char</td>
<td>8</td>
<td>8</td>
<td>0..255</td>
</tr>
<tr>
<td></td>
<td>short</td>
<td>16</td>
<td>16</td>
<td>-32768..32767</td>
</tr>
<tr>
<td></td>
<td>signed short</td>
<td>16</td>
<td>16</td>
<td>0..65535</td>
</tr>
<tr>
<td></td>
<td>unsigned short</td>
<td>16</td>
<td>16</td>
<td>0..65535</td>
</tr>
<tr>
<td></td>
<td>int</td>
<td>32</td>
<td>32</td>
<td>-2147483648..2147483647</td>
</tr>
<tr>
<td></td>
<td>signed int</td>
<td>32</td>
<td>32</td>
<td>0..2147483647</td>
</tr>
<tr>
<td></td>
<td>unsigned int</td>
<td>32</td>
<td>32</td>
<td>0..2147483647</td>
</tr>
<tr>
<td></td>
<td>long</td>
<td>32</td>
<td>32</td>
<td>0..4294967295</td>
</tr>
<tr>
<td></td>
<td>unsigned long</td>
<td>32</td>
<td>32</td>
<td>0..4294967295</td>
</tr>
<tr>
<td></td>
<td>enum</td>
<td>32</td>
<td>32</td>
<td>0..256</td>
</tr>
<tr>
<td></td>
<td>long long</td>
<td>32</td>
<td>32</td>
<td>-2147483648..2147483647</td>
</tr>
<tr>
<td></td>
<td>signed long</td>
<td>32</td>
<td>32</td>
<td>0..2147483647</td>
</tr>
<tr>
<td></td>
<td>unsigned long long</td>
<td>64</td>
<td>32</td>
<td>0..9223372036854775808</td>
</tr>
<tr>
<td></td>
<td>long long signed</td>
<td>32</td>
<td>32</td>
<td>0..2147483647</td>
</tr>
<tr>
<td></td>
<td>long long unsigned long</td>
<td>64</td>
<td>32</td>
<td>0..9223372036854775808</td>
</tr>
<tr>
<td>Pointer</td>
<td>pointer to data</td>
<td>32</td>
<td>32</td>
<td>0..2147483648</td>
</tr>
<tr>
<td></td>
<td>pointer to func</td>
<td>32</td>
<td>32</td>
<td>0..2147483648</td>
</tr>
<tr>
<td>Floating-Point</td>
<td>float</td>
<td>8</td>
<td>8</td>
<td>-3.40282347e38..3.40282347e38</td>
</tr>
<tr>
<td></td>
<td>double</td>
<td>16</td>
<td>16</td>
<td>-1.7976931348623157e308..1.7976931348623157e308</td>
</tr>
<tr>
<td></td>
<td>long double</td>
<td>32</td>
<td>32</td>
<td>2.2250738585072014e308..1.7976931348623157e308</td>
</tr>
</tbody>
</table>

Table 3-1: Data Types

When you use the same type, the compiler will use the smallest sufficient integer type, unless you use compiler option ‘-’.
Double click: Main.h and insert Typedefs:

```c
//Time-Lag-Measurement using STM
union first_read
{
    unsigned int first_read_array[2];  // 2x32 Bit to store the 56-bit free-running STM-counter
    unsigned long long int first_read_STM;  // 1x64 Bit
};

union second_read
{
    unsigned int second_read_array[2];  // 2x32 Bit to store 56-bit free-running STM-counter
    unsigned long long int second_read_STM;  // 1x64 Bit
};
```
Double click: Main.h and insert Prototypes of Global Variables (Extern Declaration):

```c
//Time-Lag-Measurement using STM
extern volatile union first_read first_read_STM;
extern volatile union second_read second_read_STM;
extern unsigned long long int result_time_lag_STM; // 1x64 Bit
```
Note:
Using the 56 Bit width System-Timer (STM) with a resolution of 13,333 ns there will be an STM-Timer-Overflow / STM-Time-Range of over 30 years (see DAvE screenshot below). Therefore we will NOT care about the STM-Overflow!

See Dave:
File – New
Insert Time_Analysis.c

OK
#include "main.h"

/*
// Example Time_Analysis / Time_Measurement:
=============================================

void main(void)
{
    first_read_STM.first_read_array[0]=STM_TIM0;
    first_read_STM.first_read_array[1]=STM_CAP;
    __nop(); // your code

    second_read_STM.second_read_array[0]=STM_TIM0;
    second_read_STM.second_read_array[1]=STM_CAP;

    Time_Analysis();
}
*/

struct clock time_lag_clock;

void Time_Analysis(void)
{
    unsigned long long int result_time_lag = second_read_STM.second_read_STM-
first_read_STM.first_read_STM; // time-lag in STM-Timer-Ticks
    register double help;
    char mb[200]; // message buffer for sprintf()

    if (((float)second_read_STM.second_read_STM-(float)first_read_STM.first_read_STM)>0.0)
    {
        if (result_time_lag<75) // "short-time-lag": time_lag < 1000 ns
        {
            sprintf(mb,"\n\rTime-Lag=%Lf ns",(double)result_time_lag*(13.333333333333));
            myprintf(mb);
        }
        else if (result_time_lag<75000) // "short-time-lag": time_lag < 1000 µs
        {
            sprintf(mb,"\n\rTime-Lag=%Lf µs",(double)result_time_lag*(13.333333333333*0.001));
            myprintf(mb);
        }
        else if (result_time_lag<7500000) // "short-time-lag": time_lag < 1000 ms
        {
            sprintf(mb,"\n\rTime-Lag=%Lf ms",(double)result_time_lag*(13.333333333333*0.001*0.001));
            myprintf(mb);
        }
        else // "long-time-lag": time_lag >= 1000 ms
    }
```c
{
    time_lag_clock.day=time_lag_clock.hour=time_lag_clock.minute=time_lag_clock.second=0;
    help=(double)result_time_lag*(13.3333333333*0.001*0.001*0.001);
    do
    {
        ++time_lag_clock.second,help=help-1.0;
        if (time_lag_clock.second==60)
        {
            time_lag_clock.second=0;
            if (++time_lag_clock.minute==60)
            {
                time_lag_clock.minute=0;
                if (++time_lag_clock.hour==24)
                {
                    time_lag_clock.hour=0, ++time_lag_clock.day;
                }
            }
        }
    }while (help>0.0);
    myprintf("\r\n*** TIME-LAG ***\n");
    sprintf(mb,"*** %02u:%02u:%02u
***",time_lag_clock.hour,time_lag_clock.minute,time_lag_clock.second);
    myprintf(mb);
}
```
File
Save All

(Project Window File View) – TC1130 (Files) – right mouse button click – Add Existing Files – Browse
Select Time_Analysis.c

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

void main(void)
{
    // Example Time_Analysis / Time_Measurement:
    struct clock_time_log_clock;
    void Time_Analysis(void)
    {
        unsigned long long int result_time_log = second_read_STN.second_read_STN.first_read_STN.first_read_STN; // ti
        register double help;
        help = (double)(result_time_log); // message buffer for sprintf()
        if (((float)second_read_STN.second_read_STN.first_read_STN.first_read_STN) > 0)
        {
            if (result_time_log < 1000) // "short-time-log": time_log < 1000 ns
            {
                printf("time_log=(%f) ns", (double)(result_time_log/(double)13.333333333333333));
            }
            else if (result_time_log < 5000) // "short-time-log": time_log < 2000 µs
            {
            }
        }
    }
```
Double click: `Main.h` and insert Prototypes of Global Functions (Extern Declaration):

```c
extern void Time_Analysis(void);
```
Double click: Main.c and change Code [“void main (void)” - Function] from

```c
switch (select)
{
    case '1': blinking=OFF, IO_P0_7=LED_ON,
        myprintf(message1); break;
    case '2': blinking=OFF, IO_P0_7=LED_OFF,
        myprintf(message2); break;
    case '3': blinking=ON, myprintf(message3); break;
    case '4': Set_STM_Clock(); break; // Set STM clock
    case '5': if (STM_Clock_Running == FALSE)
        STM_Clock_Running=TRUE; myprintf(message4); break; // Start STM clock
    case '6': if (STM_Clock_Running == TRUE)
        STM_Clock_Running=FALSE; myprintf(message5); break; // Stop STM clock
    case '7': Show_STM_Clock(); break; // Show STM clock
}
```

to

```c
switch (select)
{
    case '1': blinking=OFF, IO_P0_7=LED_ON,
        myprintf(message1); break;
    case '2': blinking=OFF, IO_P0_7=LED_OFF,
        myprintf(message2); break;
    case '3': blinking=ON, myprintf(message3); break;
    case '4': Set_STM_Clock(); break; // Set STM clock
    case '5': if (STM_Clock_Running == FALSE)
        STM_Clock_Running=TRUE; myprintf(message4); break; // Start STM clock
    case '6': if (STM_Clock_Running == TRUE)
        STM_Clock_Running=FALSE, myprintf(message5); break; // Stop STM clock
    case '7': Show_STM_Clock(), Time_Analysis(); break;
}
```
// the CPU interrupt system is globally disabled
DISABLE();

// I/O-Monitor: On-board-Burst-Flash switch to Burst-Mode
switch_to_burst();

// the CPU interrupt system is globally enabled
ENABLE();

while (1)
{
    ApiResponse(1); // select = input
    switch (select)
    {
        case 1: blinking_OFF, I0_D0_D4_LED_0H, MyPrint(message); break;
        case 2: blinking_ON, I0_D0_D4_LED_0H, MyPrint(message); break;
        case 3: blinking_OFF, MyPrint(message); break;
        case 4: Set_STM(Clock)); break; // set STM clock
        case 5: if [STM_CLK_Panning = FALSE] | STM_CLK_Panning = TRUE,
            first_read_STM.first_read_array[0] = STM_TIME, // example: long-term-measurement //
            first_read_STM.first_read_array[1] = STM_CLK, // example: long-term-measurement //
            MyPrint(message); break; // start STM clock
        case 6: if [STM_CLK_Panning = TRUE] | STM_CLK_Panning = FALSE,
            second_read_STM.second_read_array[0] = STM_TIME, // example: long-term-measurement //
            second_read_STM.second_read_array[1] = STM_CLK, // example: long-term-measurement //
            MyPrint(message); break; // stop STM clock
        case 7: Show_STM(Clock)); Time_Analysis(); break;
    }

    // USER CODE END

    return(0); // Return
}

// End of function main
Generate your application program:

- Build
- Rebuild

or

[Image of 'Execute Rebuild command.' button]

Now you can close both your project and Tasking EDE:

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

Start pls-Debugger

File – Open Workspace

Look in: select C:\TC1130
File name: select TC1130.wsp

Open
Cancel

File – Load Program

Look in: select TC1130
File name: select TC1130.elf

Open

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:

---

testing 64 MBytes OnBoardSDRAM at 0x40004034, pattern = 1010 B ...
testing 64 MBytes OnBoardSDRAM at 0x40004034, pattern = 0101 B ...
64 MBytes OnBoardSDRAM at 0x40004034: (1212 Byte(s)) ... ok
128 Byte(s) at 0x40000004 (heap) allocated
Copy Program into heap
32x4=128 Byte(s) copied into heap
JUMP to heap
BACK from heap, Burst-Mode activated

Program execution out of Intel’s 32 MBytes OnBoardFlash (Burst Mode !!!)

1 ... LED IO_Port_0_Pin_7 ON
2 ... LED IO_Port_0.Pin_7 OFF
3 ... LED IO_Port_0.Pin_7 blinking
4 ... Set The Software Clock Using The STM-Interrupt
5 ... Start The Software Clock Using The STM-Interrupt
6 ... Stop The Software Clock Using The STM-Interrupt
7 ... Show The Software Clock Using The STM-Interrupt
8 ... Back To Main Menu

---

your choice: 1

Modem Status

CTS ✔ DSR ✔ RING ✔ RLSD ✔

Console Status

CTS Hold ✔ XOFF Hold ✔ TX Chip ✔

RING Hold ✔ XOFF Sent ✔ TX Chars: 0

RLSD Hold ✔ EOF Sent ✔ RX Chars: 0

---

---
Insert 5

Program execution out of Intel’s 32 MBytes OnBoardFlash (Burst Mode !!!)
=========================================================================
1 ... LED IO_P0_Pin_7 ON
2 ... LED IO_P0_Pin_7 OFF
3 ... LED IO_P0_Pin_7 blinking
4 ... Set The Software Clock Using The STM-Interrupt
5 ... Start The Software Clock Using The STM-Interrupt
6 ... Stop The Software Clock Using The STM-Interrupt
7 ... Show The Software Clock Using The STM-Interrupt
z ... Back To Main Menu

your choice: 1

*** STM Clock is RUNNING ***
4. Set the software clock using the STM-interrupt
5. Start the software clock using the STM-interrupt
6. Stop the software clock using the STM-interrupt
7. Show the software clock using the STM-interrupt
z. Back to main menu

Your choice: 6

*** STM Clock is STOPPED ***

Program execution out of Intel's 32 MBytes OnBoard Flash (Burst Mode )
------------------------------------------------------------------------
1. LED IO_Port_0_Pin_7 ON
2. LED IO_Port_0_Pin_7 OFF
3. LED IO_Port_0_Pin_7 blinking
4. Set the software clock using the STM-interrupt
5. Start the software clock using the STM-interrupt
6. Stop the software clock using the STM-interrupt
7. Show the software clock using the STM-interrupt
z. Back to main menu

Your choice: 4
Insert 7

**Multi-threaded TTY**

7 ... Show The Software Clock Using The STM-Interrupt
2 ... Back To Main Menu

your choice: 6

*** STM Clock is STOPPED ***

Program execution out of Intel’s 32 MBytes OnBoardFlash ( Burst Mode !!! )

1 ... LED IO_Port_0_Pin_7 ON
2 ... LED IO_Port_0_Pin_7 OFF
3 ... LED IO_Port_0_Pin_7 blinking
4 ... Set The Software Clock Using The STM-Interrupt
5 ... Start The Software Clock Using The STM-Interrupt
6 ... Stop The Software Clock Using The STM-Interrupt
7 ... Show The Software Clock Using The STM-Interrupt
2 ... Back To Main Menu

your choice: 7

****** TIME ******

*** 00:03:06 ***

Application Note 297

V2.0, 2007-04
Insert z

Back To Main Menu

Program execution out of Intel’s 32 MBytes OnBoardFlash (Burst Mode III)

1 ... LED IO_Port_0_Pin_7 ON
2 ... LED IO_Port_0_Pin_7 OFF
3 ... LED IO_Port_0_Pin_7 blinking
4 ... Set The Software Clock Using The STM-Interrupt
5 ... Start The Software Clock Using The STM-Interrupt
6 ... Stop The Software Clock Using The STM-Interrupt
7 ... Show The Software Clock Using The STM-Interrupt
z ... Back To Main Menu

your choice: 7

xxxxxx TIME xxxxxx

xxx 00:03:06 xxx

xxx TIME-LAG xxx

xxx 00:03:07 xxx
We recommend now to **copy and store** your project-directory “C:\TC1130” to “08.2.1_TC1130_UsingSTM_long-time-lag-measurement”: 

![Image of file system with highlighted folder](image-url)
8.2.2.) Time-Lag-Measurement Using the STM (example: “short-time-lag-measurement”)
Start Tasking EDE and open the project:

File – Open Project Space

Look in: select C:\TC1130
File name: select TC1130.psp

Open
Double click: Main.c insert Code [before "while(1) {;}"]:

DISABLE(); // the CPU interrupt system is globally disabled

first_read_STM.first_read_array[0]=STM_TIM0;
first_read_STM.first_read_array[1]=STM_CAP;

__nop(); // dummy code !!!

second_read_STM.second_read_array[0]=STM_TIM0;
second_read_STM.second_read_array[1]=STM_CAP;

Time_Analysis();

ENABLE(); // the CPU interrupt system is globally enabled
Generate your application program:

Build
Rebuild

or

Execute 'Rebuild' command.

Now you can close both your project and Tasking EDE:

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

**Start** pls-Debugger

**File – Open Workspace**

Look in: select C:\TC1130  
File name: select TC1130.wsp

Open  
Cancel

**File – Load Program**

Look in: select TC1130  
File name: select TC1130.elf

Open

**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:

---

Multi-threaded TTY

Program execution out of Intel's 32 kByte OnBoardFlash (Burst Mode !!! )

1. LED 10_Por_0_Pin_7 ON
2. LED 10_Por_0_Pin_7 OFF
3. LED 10_Por_0_Pin_7 blinking
4. Set the Software Clock Using the STM-Interrupt
5. Start the Software Clock Using the STM-Interrupt
6. Stop the Software Clock Using the STM-Interrupt
7. Show the Software Clock Using the STM-Interrupt
z. Back To Main Menu

---
Start Tasking EDE and open the project:

File – Open Project Space

Look in: select C:\TC1130
File name: select TC1130.psp

Open
Double click: Main.c insert Code:

```c
// Additionally: 200 "NOPs" - should be about 1,333 μs (200*6,6667ns)
__nop();__nop();__nop();__nop();__nop();__nop();__nop();__nop();
__nop();__nop();__nop();__nop();__nop();__nop();__nop();__nop();
__nop();__nop();__nop();__nop();__nop();__nop();__nop();__nop();
__nop();__nop();__nop();__nop();__nop();__nop();__nop();__nop();
__nop();__nop();__nop();__nop();__nop();__nop();__nop();__nop();
__nop();__nop();__nop();__nop();__nop();__nop();__nop();__nop();
__nop();__nop();__nop();__nop();__nop();__nop();__nop();__nop();
__nop();__nop();__nop();__nop();__nop();__nop();__nop();__nop();
__nop();__nop();__nop();__nop();__nop();__nop();__nop();__nop();
__nop();__nop();__nop();__nop();__nop();__nop();__nop();__nop();
__nop();__nop();__nop();__nop();__nop();__nop();__nop();__nop();
__nop();__nop();__nop();__nop();__nop();__nop();__nop();__nop();
__nop();__nop();__nop();__nop();__nop();__nop();__nop();__nop();
__nop();__nop();__nop();__nop();__nop();__nop();__nop();__nop();
__nop();__nop();__nop();__nop();__nop();__nop();__nop();__nop();
__nop();__nop();__nop();__nop();__nop();__nop();__nop();__nop();
__nop();__nop();__nop();__nop();__nop();__nop();__nop();__nop();
__nop();__nop();__nop();__nop();__nop();__nop();__nop();__nop();
__nop();__nop();__nop();__nop();__nop();__nop();__nop();__nop();
__nop();__nop();__nop();__nop();__nop();__nop();__nop();__nop();
__nop();__nop();__nop();__nop();__nop();__nop();__nop();__nop();
```
DISABLE(); // the CPU interrupt system is globally disabled

first_read_STM.first_read_accx();=STM_TIMO;
first_read_STM.first_read_accy();=STM_CAN;

ENABLE(); // dummy code !!!

// Additionally: 208 "MUs" ~ should be about 1,489 ms (200*6,666us)

second_read_STM.second_read_accx();=STM_TIMO;
second_read_STM.second_read_accy();=STM_CAN;

Time_Analysis();

ENABLE(); // the CPU interrupt system is globally enabled
Generate your application program:

Build
Rebuild

or

Execute 'Rebuild' command.

Now you can close both your project and Tasking EDE:

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

Start pls-Debugger

File – Open Workspace

Look in: select C:\TC1130
File name: select TC1130.wsp

Open
Cancel

File – Load Program

Look in: select TC1130
File name: select TC1130.elf

Open

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:

```
Program execution out of Intel's 32 MB onBoardFlash ( Burst Mode !!! )
=========================================
1 ... LED IO_Port_0_Pin_7 ON
2 ... LED IO_Port_0_Pin_7 OFF
3 ... LED IO_Port_0_Pin_7 blinking
4 ... Set The Software Clock Using The STM-Interrupt
5 ... Start The Software Clock Using The STM-Interrupt
6 ... Stop The Software Clock Using The STM-Interrupt
7 ... Show The Software Clock Using The STM-Interrupt
2 ... Back To Main Menu
```
Conclusion:

We expected 1,33 µs for additional 200 NOPs (200 * 6.667 ns).
We got 7,146 µs – 880 ns = 6.3 µs instead of 1,33 µs [Cause: LMB Bus Arbitration, EBU].

Just to satisfy curiosity:
Is it possibly to measure even CPU clocks (150 MHz -> 6,6667 ns) ?

We used the programming example “04_TC1130_SPRAM”, deactivated all memories - except PMI_SPRAM and DMI_SPRAM, added software from chapter 8.1. upwards and got the following values:
1,546667 µs – 213 ns = 1,3337 µs – THIS IS EXACTLY THE TIME WE CALCULATED FOR 200 NOPs (1,337 µs / 200 = 6.67 ns = ONE CPU CLOCK).

We recommend to store this additional programming example as “08.2.2_TC1130_Using_STM_short-time-lag-measurement_SPRAM-ONLY”.

What we have learnt:
LMB-Bus-Arbitration and External-Memory-Access via EBU - which both is necessary for Program-Execution out-of On-Board-Flash takes longer than execution out of internal memory.
We recommend now to copy and store your project-directory “C:\TC1130” to “08.2.2_TC1130_Using_STM_short-time-lag-measurement”:
9.) Feedback (TC1130): Your opinion, suggestions and/or criticisms

Contact Details (this section may remain empty 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: