

AP32117

TC1130

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

Microcontrollers



Never stop thinking

Edition 2008-07-17

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

**© Infineon Technologies AG 2008.
All Rights Reserved.**

LEGAL DISCLAIMER

THE INFORMATION GIVEN IN THIS APPLICATION NOTE IS GIVEN AS A HINT FOR THE IMPLEMENTATION OF THE INFINEON TECHNOLOGIES COMPONENT ONLY AND SHALL NOT BE REGARDED AS ANY DESCRIPTION OR WARRANTY OF A CERTAIN FUNCTIONALITY, CONDITION OR QUALITY OF THE INFINEON TECHNOLOGIES COMPONENT. THE RECIPIENT OF THIS APPLICATION NOTE MUST VERIFY ANY FUNCTION DESCRIBED HEREIN IN THE REAL APPLICATION. INFINEON TECHNOLOGIES HEREBY DISCLAIMS ANY AND ALL WARRANTIES AND LIABILITIES OF ANY KIND (INCLUDING WITHOUT LIMITATION WARRANTIES OF NON-INFRINGEMENT OF INTELLECTUAL PROPERTY RIGHTS OF ANY THIRD PARTY) WITH RESPECT TO ANY AND ALL INFORMATION GIVEN IN THIS APPLICATION NOTE.

Information

For further information on technology, delivery terms and conditions and prices please contact your nearest Infineon Technologies Office (www.infineon.com).

Warnings

Due to technical requirements components may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies Office.

Infineon Technologies Components may only be used in life-support devices or systems with the express written approval of Infineon Technologies, if a failure of such components can reasonably be expected to cause the failure of that life-support device or system, or to affect the safety or effectiveness of that device or system. Life support devices or systems are intended to be implanted in the human body, or to support and/or maintain and sustain and/or protect human life. If they fail, it is reasonable to assume that the health of the user or other persons may be endangered.

Note: Table of Contents see [page 8](#).

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!

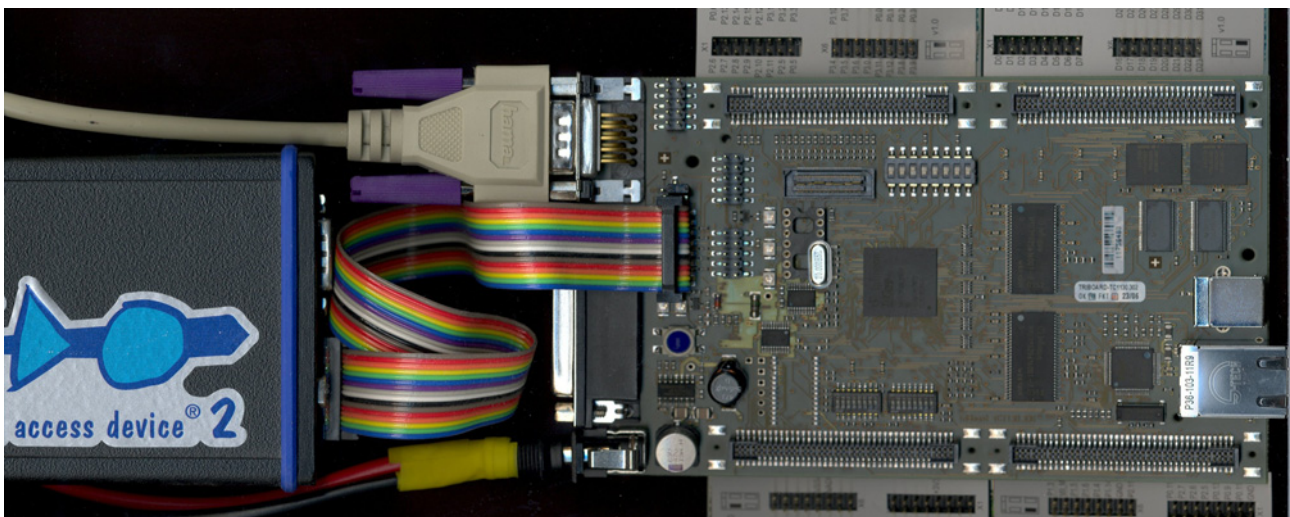




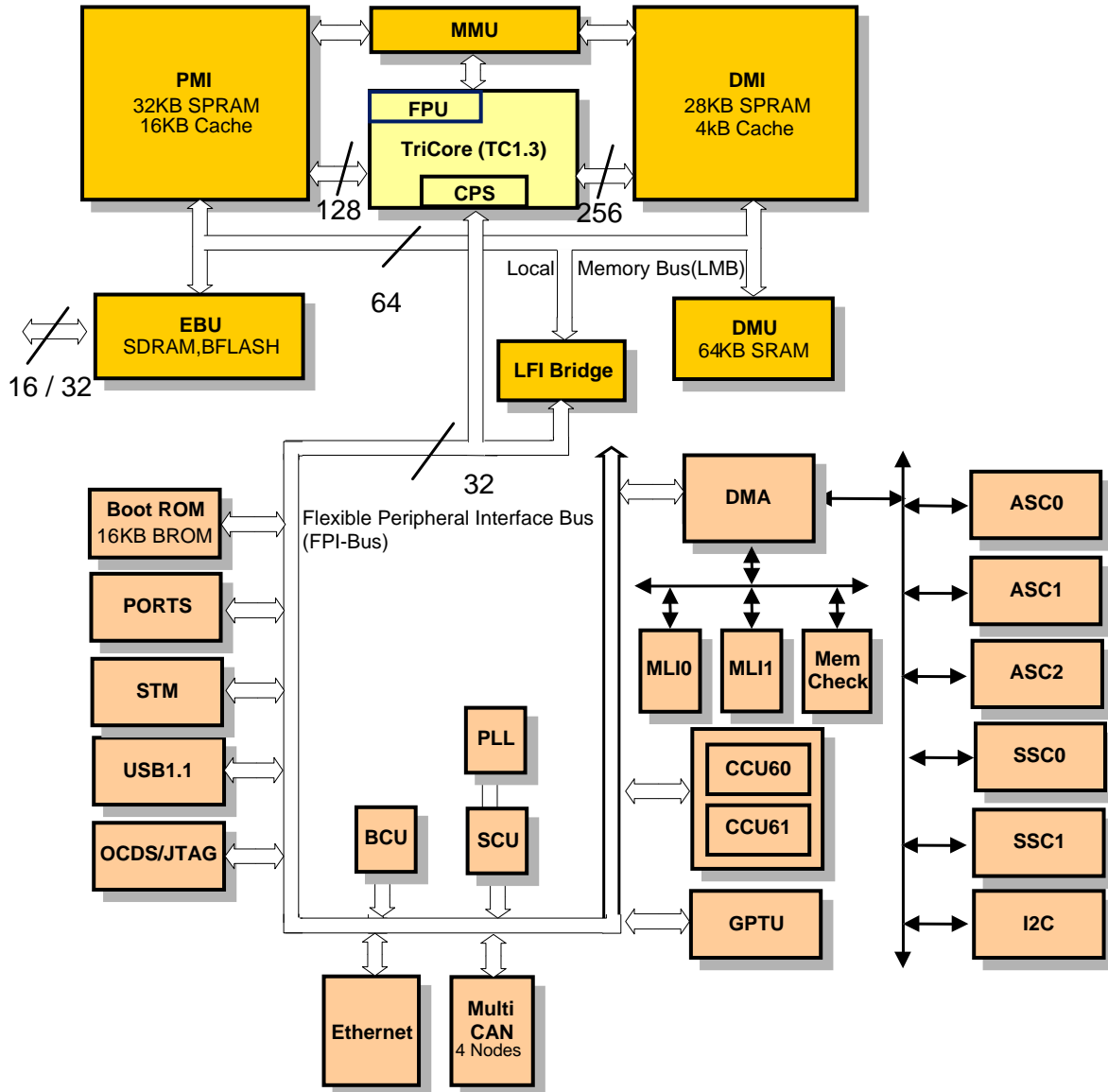
Programming Examples

TC1130

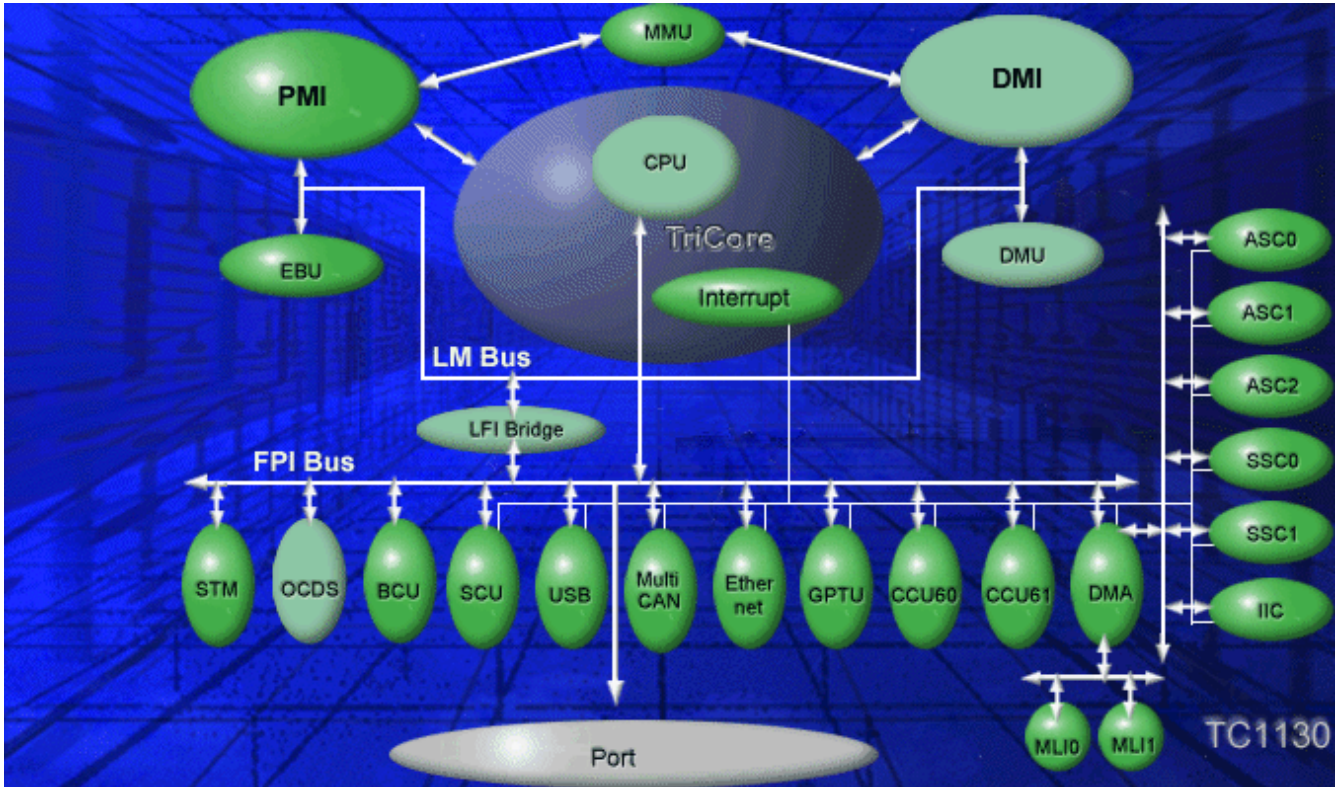
Starter Kit



TC1130 Block Diagram (Source: Product Marketing)

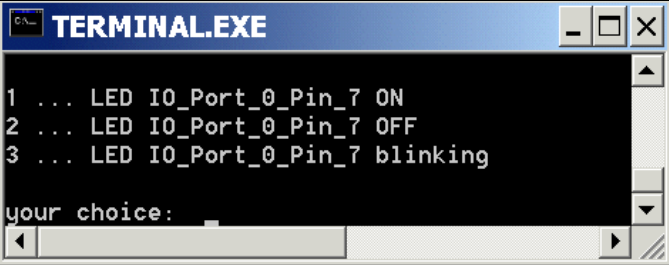





TC1130 Block Diagram (Source: code generator DAVE)



“Cookery-Book“

For your first programming example for the TC1130 Starter-Kit-Board:

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

Additional exercises

7.)	Time – Measurement (Using an oscilloscope / a logic analyzer)
8.)	Time – Measurement [Using the SystemTimer (STM)]

Feedback

9.)	Feedback
-----	--------------------------

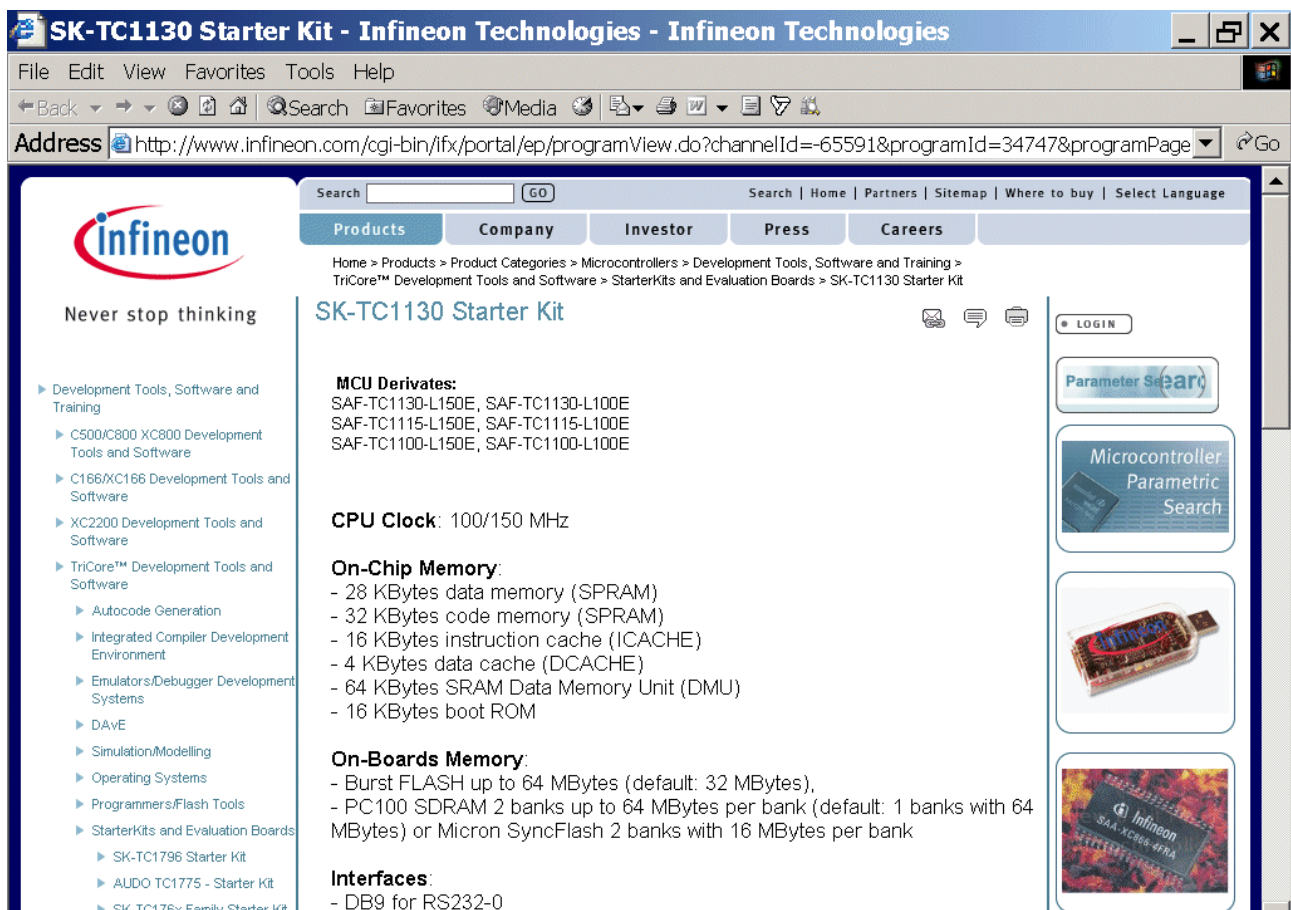
1.) TC1130 Starter Kit Board:



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:



The screenshot shows a web browser window displaying the Infineon website for the SK-TC1130 Starter Kit. The browser title is "SK-TC1130 Starter Kit - Infineon Technologies - Infineon Technologies". The address bar shows the URL: <http://www.infineon.com/cgi-bin/ifx/portal/ep/programView.do?channelId=65591&programId=34747&programPage=...>

The webpage content includes:

- Navigation:** Search, Home, Partners, Sitemap, Where to buy, Select Language. Main menu: Products, Company, Investor, Press, Careers.
- Breadcrumbs:** Home > Products > Product Categories > Microcontrollers > Development Tools, Software and Training > TriCore™ Development Tools and Software > StarterKits and Evaluation Boards > SK-TC1130 Starter Kit
- Product Title:** SK-TC1130 Starter Kit
- MCU Derivates:**
 - SAF-TC1130-L150E, SAF-TC1130-L100E
 - SAF-TC1115-L150E, SAF-TC1115-L100E
 - SAF-TC1100-L150E, SAF-TC1100-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 (DMU)
 - 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
- Left Sidebar:** "Never stop thinking" with a navigation tree including Development Tools, Software and Training; C500/C800 XC800 Development Tools and Software; C166/XC166 Development Tools and Software; XC2200 Development Tools and Software; TriCore™ Development Tools and Software (Autocode Generation, Integrated Compiler Development Environment, Emulators/Debugger Development Systems, DAVe, Simulation/Modelling, Operating Systems, Programmers/Flash Tools); StarterKits and Evaluation Boards (SK-TC1796 Starter Kit, AUDDO TC1775 - Starter Kit, SK-TC176x Family Starter Kit).
- Right Sidebar:** LOGIN button, Parameter Search, Microcontroller Parametric Search, and images of the starter kit components.

- ▶ SK-TC1765 Starter Kit
- ▶ SK-TC1130 Starter Kit
- ▶ SK TC116x-Series Starter Kit
- ▶ Software Partners
- ▶ Software Downloads
- ▶ Training
- ▶ DAVE-Digital Application virtual Engineer
- ▶ DAVE Drive

- BERG10 for RS232-1
- two BERG10 for CAN-0/1 with Transceiver
- BERG16 for OCDS1
- SAMTEC QSH-030-01-F-D-A for OCDS2
- RJ45 Connector with LED's for Ethernet
- USB connector (type B) for USB connections
- DB25 On Board Wiggler for OCDS1
- 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

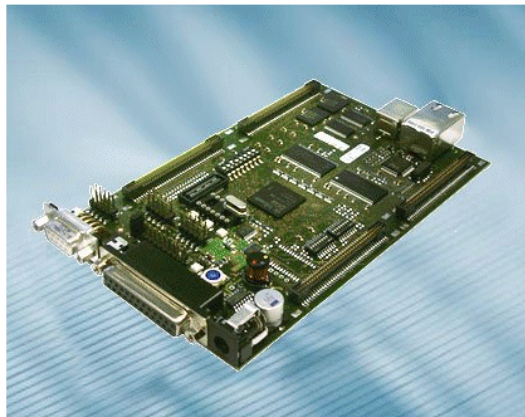
Order Nr.: B158-H8359-X-X-7600

Price*: 400,- EUR

How to order?

[Buy Online](#)

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



▼ Documentation

Description	File Name	Size	Date	Revision
TriBoard TC1130 - Hardware Manual TC1130-300	TriBoardManual-TC1130-V12.pdf	1.6 MB	1 Jul 2005	

©1999 - 2007 Infineon Technologies AG - Usage of this website is subject to our Usage Terms -Imprint -Contact
PRIVACY POLICY



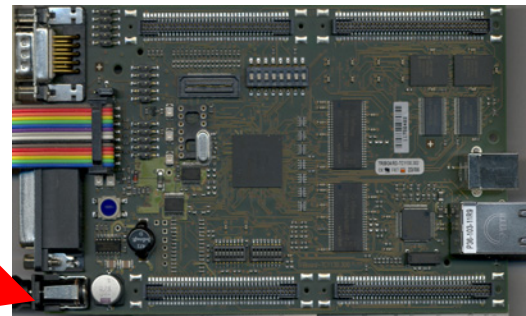
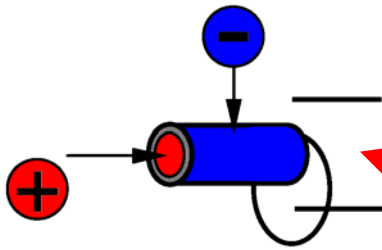
Local intranet

Connecting the TC1130-Board to the Environment:

Connect a Power Supply:

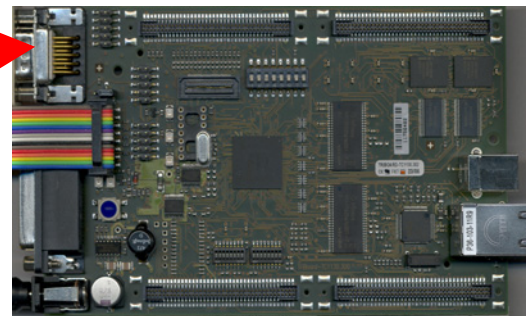
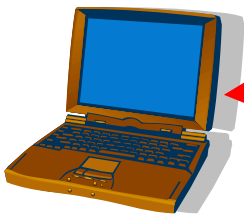
The TC1130 Board requires an external power supply.

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.

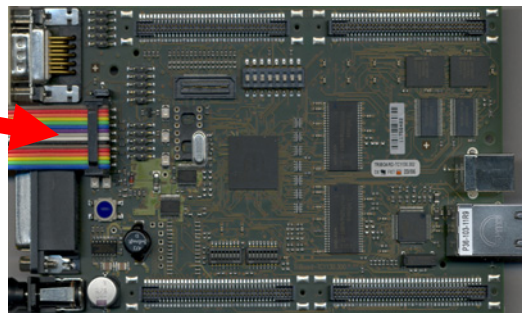


Connect a RS-232 Serial Cable

(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):

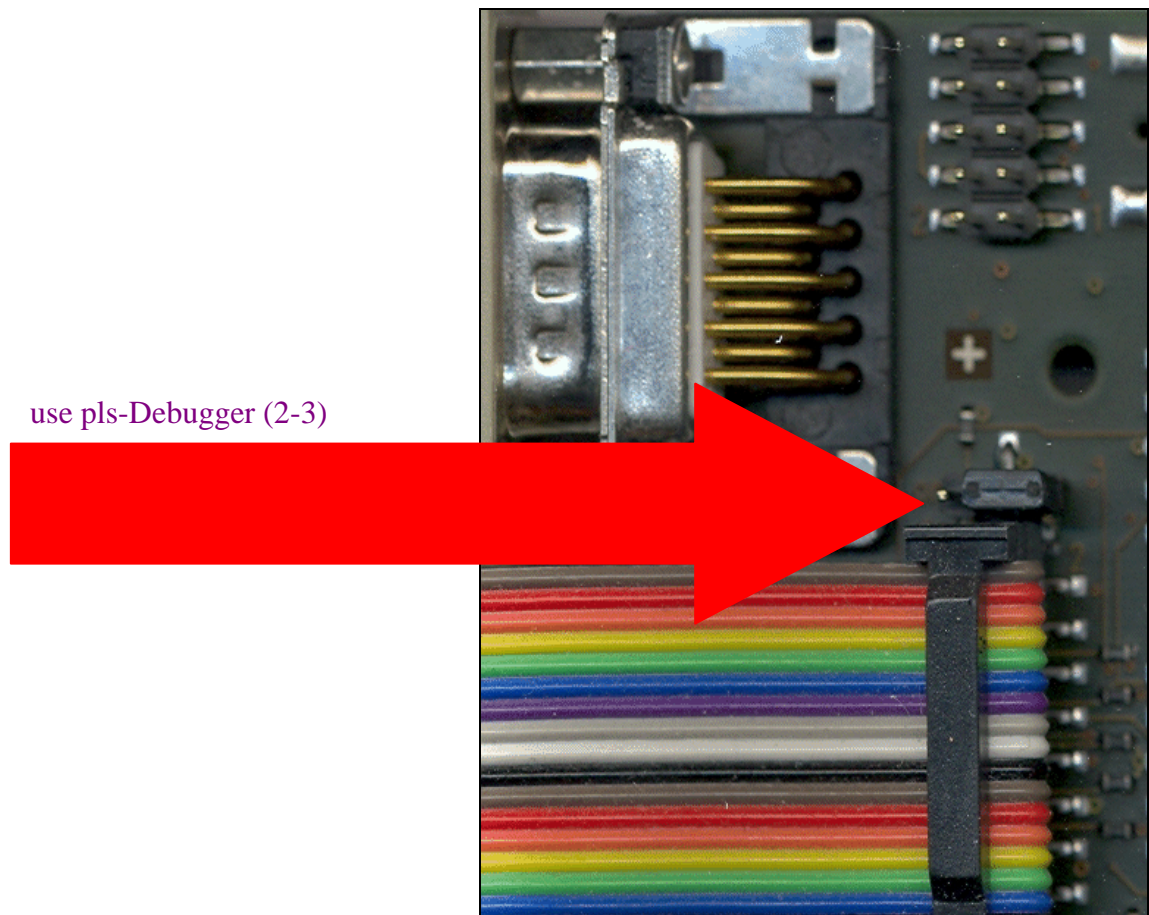
Note:

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**)

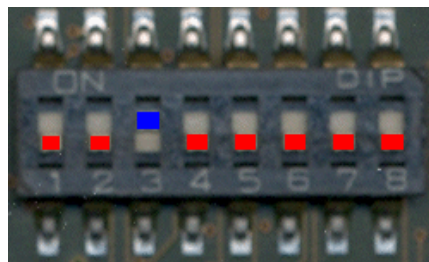
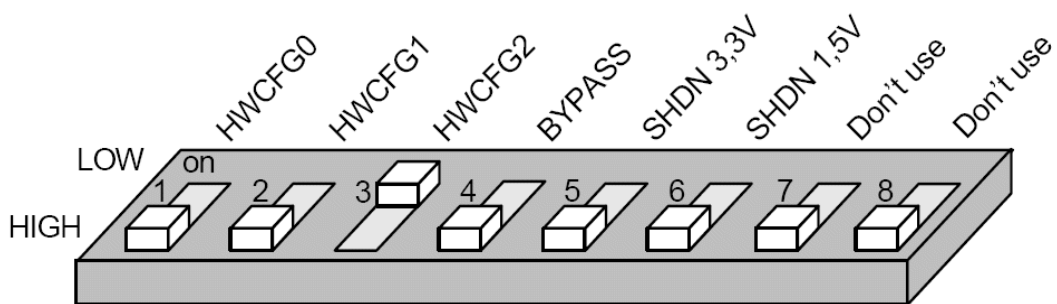
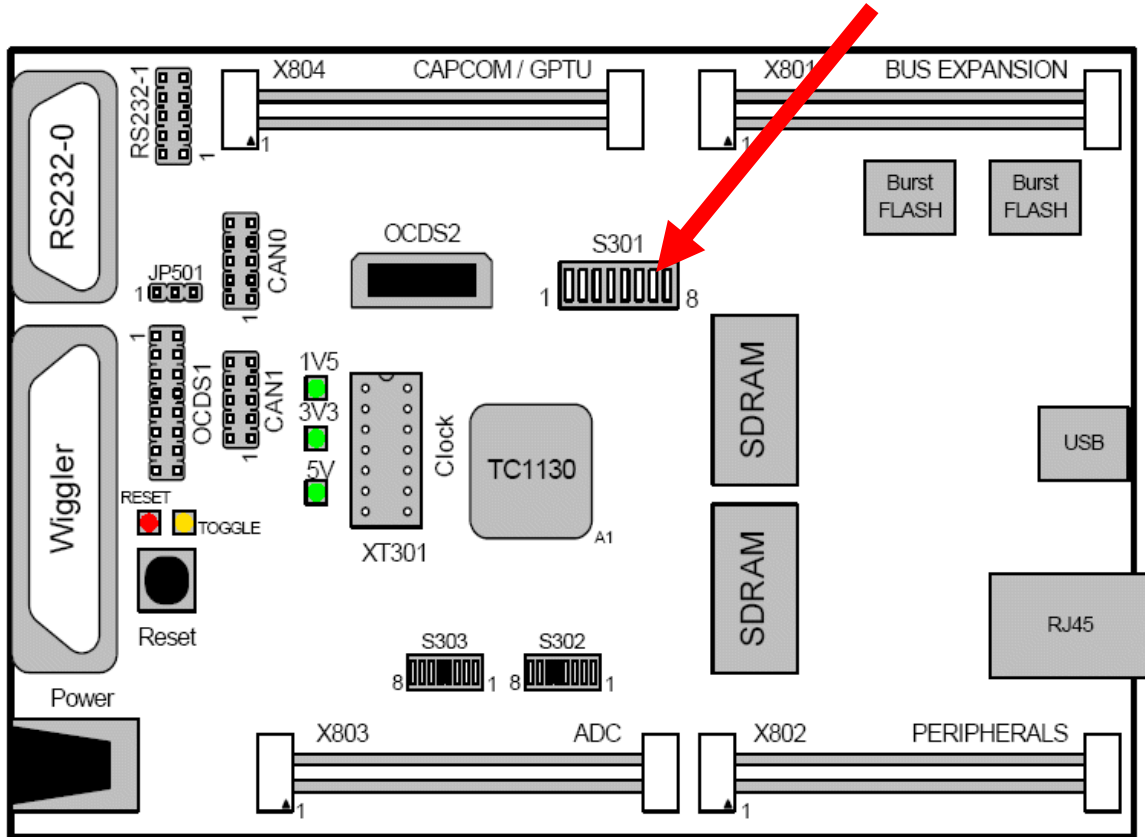


TC1130-Execution-Environment-SPRAM:

Note:

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

Jumper Settings (HW-Configuration DIP-Switch S301)



S301:

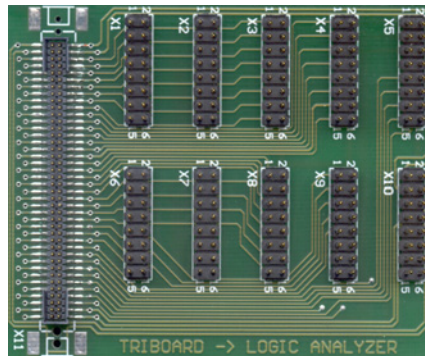
1, 2, 4, 5, 6, 7, 8 : OFF (default)

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)

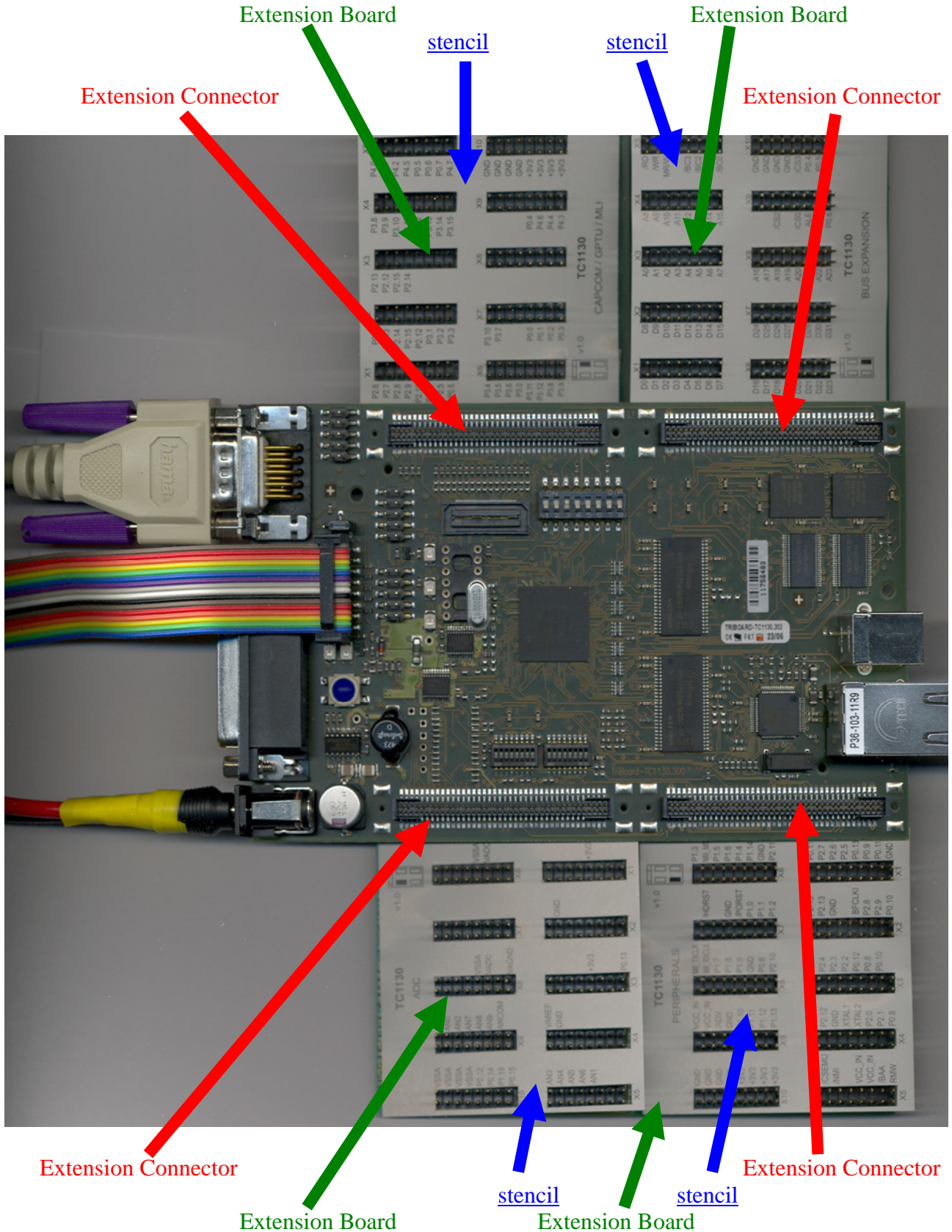
Purpose:

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> the DAvE-mothersystem **setup.exe**

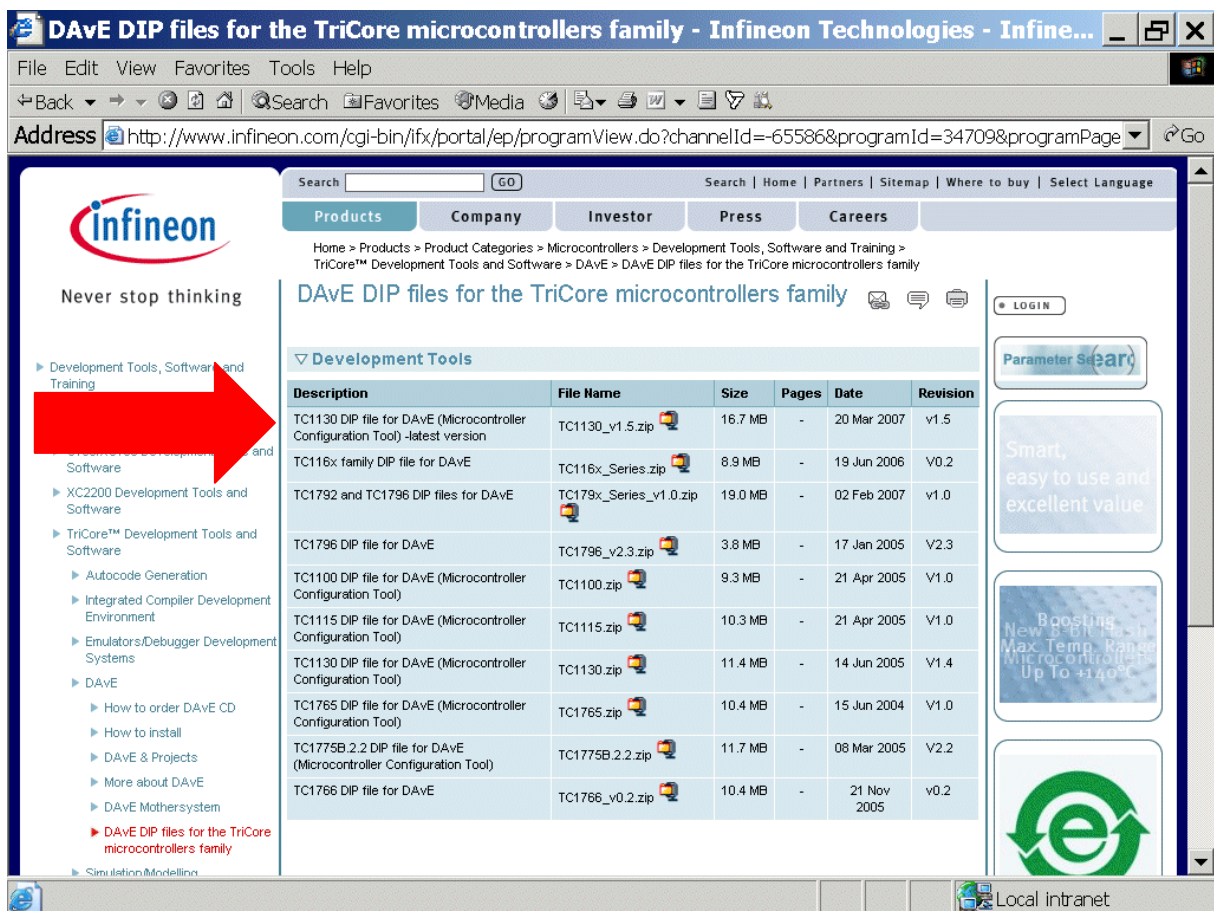
Title	Date	Size
Tool Package		
 DAvE - Mothersystem - latest version	05 Feb 2007	14.8 MB
 DAvE - Mothersystem	04 Jul 2006	15.1 MB

and execute **setup.exe** to install DAvE .

Install the TC1130 microcontroller Update:

1.)

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




The screenshot shows a web browser window displaying the Infineon website. The page title is "DAvE DIP files for the TriCore microcontrollers family - Infineon Technologies - Infine...". The browser address bar shows the URL: <http://www.infineon.com/cgi-bin/ifx/portal/ep/programView.do?channelId=-65586&programId=34709&programPage>. The page content includes a navigation menu, a search bar, and a table of DAvE DIP files. A red arrow points to the file "TC1130_v1.5.zip" in the table.

Description	File Name	Size	Pages	Date	Revision
TC1130 DIP file for DAvE (Microcontroller Configuration Tool) -latest version	TC1130_v1.5.zip	16.7 MB	-	20 Mar 2007	v1.5
TC116x family DIP file for DAvE	TC116x_Series.zip	8.9 MB	-	19 Jun 2006	V0.2
TC1792 and TC1796 DIP files for DAvE	TC179x_Series_v1.0.zip	19.0 MB	-	02 Feb 2007	v1.0
TC1796 DIP file for DAvE	TC1796_v2.3.zip	3.8 MB	-	17 Jan 2005	V2.3
TC1100 DIP file for DAvE (Microcontroller Configuration Tool)	TC1100.zip	9.3 MB	-	21 Apr 2005	V1.0
TC1115 DIP file for DAvE (Microcontroller Configuration Tool)	TC1115.zip	10.3 MB	-	21 Apr 2005	V1.0
TC1130 DIP file for DAvE (Microcontroller Configuration Tool)	TC1130.zip	11.4 MB	-	14 Jun 2005	V1.4
TC1765 DIP file for DAvE (Microcontroller Configuration Tool)	TC1765.zip	10.4 MB	-	15 Jun 2004	V1.0
TC1775B 2.2 DIP file for DAvE (Microcontroller Configuration Tool)	TC1775B_2.2.zip	11.7 MB	-	08 Mar 2005	V2.2
TC1766 DIP file for DAvE	TC1766_v0.2.zip	10.4 MB	-	21 Nov 2005	v0.2

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

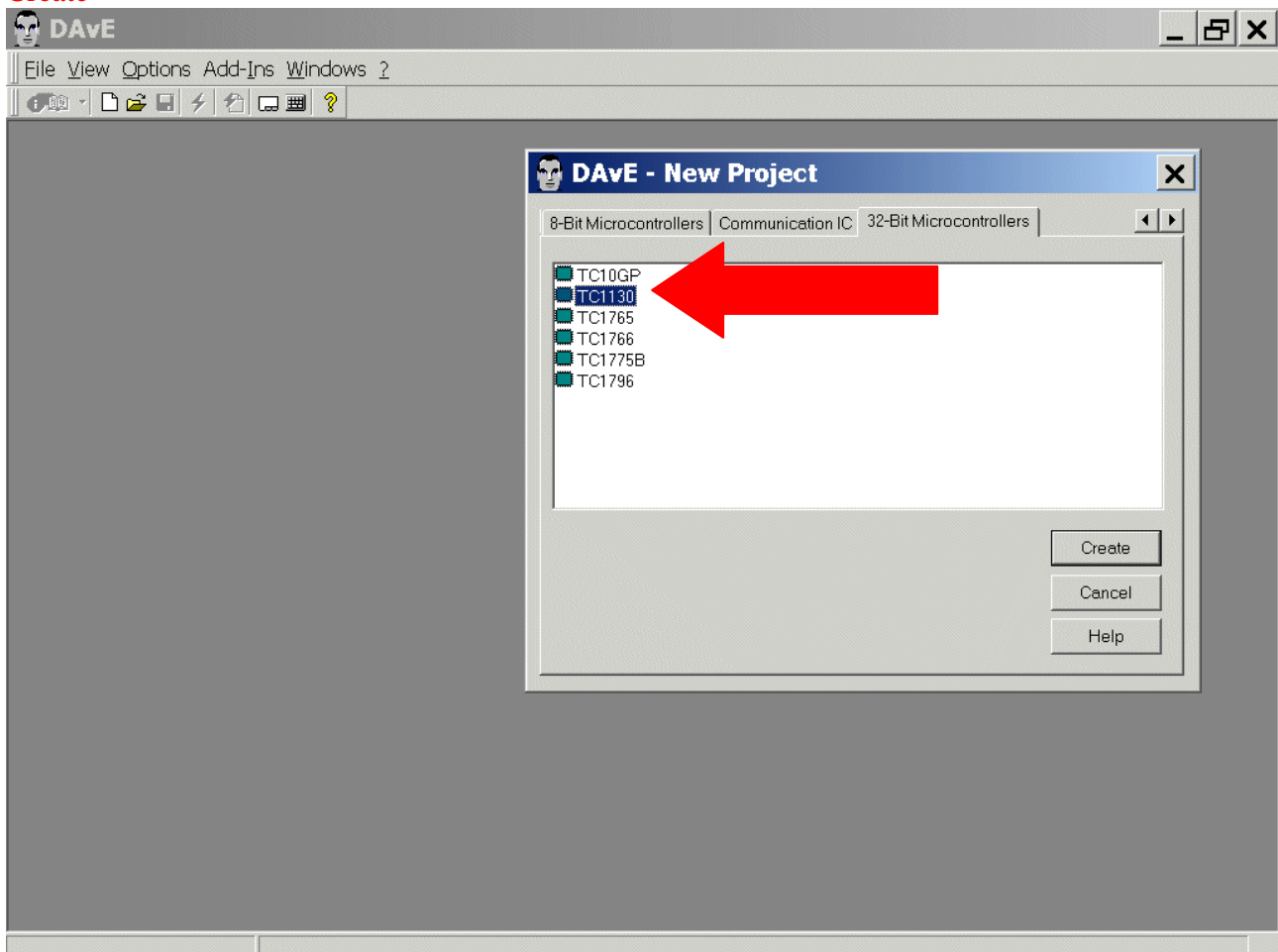
4.) DAVe is now ready to generate code for the TC1130 microcontroller.

3.) DAvE - Microcontroller Initialization after Power-On:



Start the program generator DAvE and select the TC1130 microcontroller:

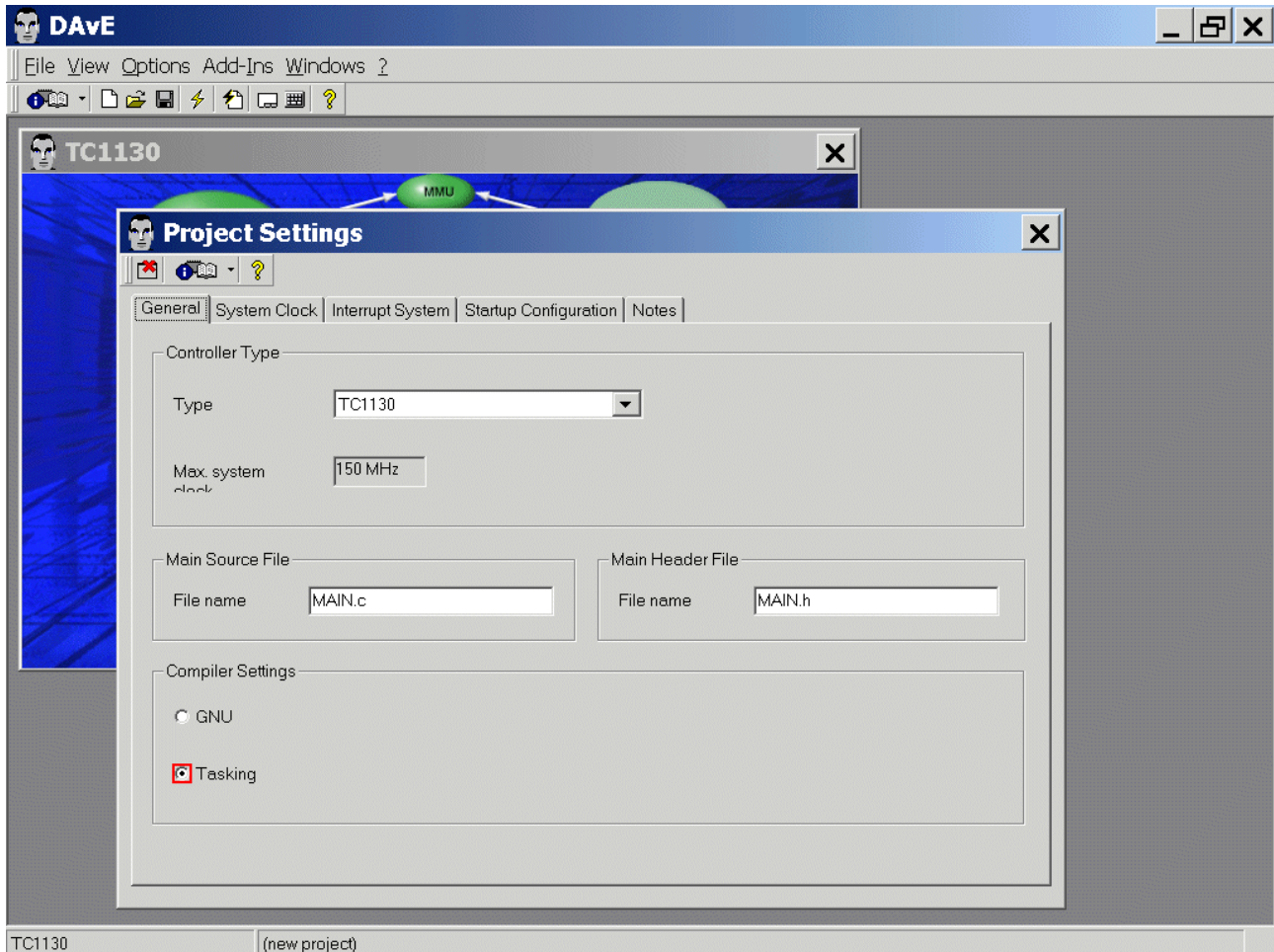
File
New
32-Bit Microcontrollers
TC1130
Create



Choose the Project Settings as you can see in the Screenshots:

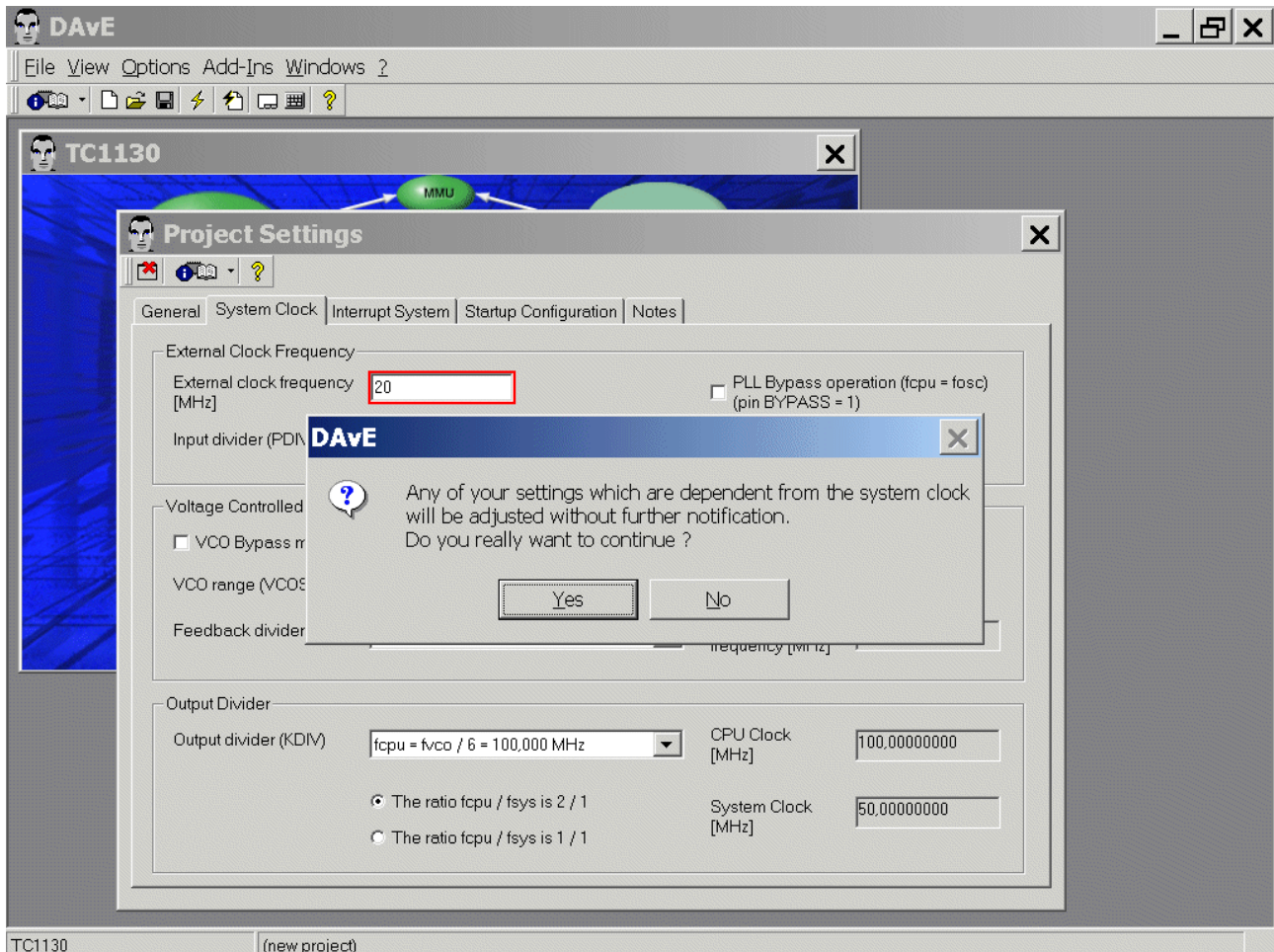
General: Compiler Settings:

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).

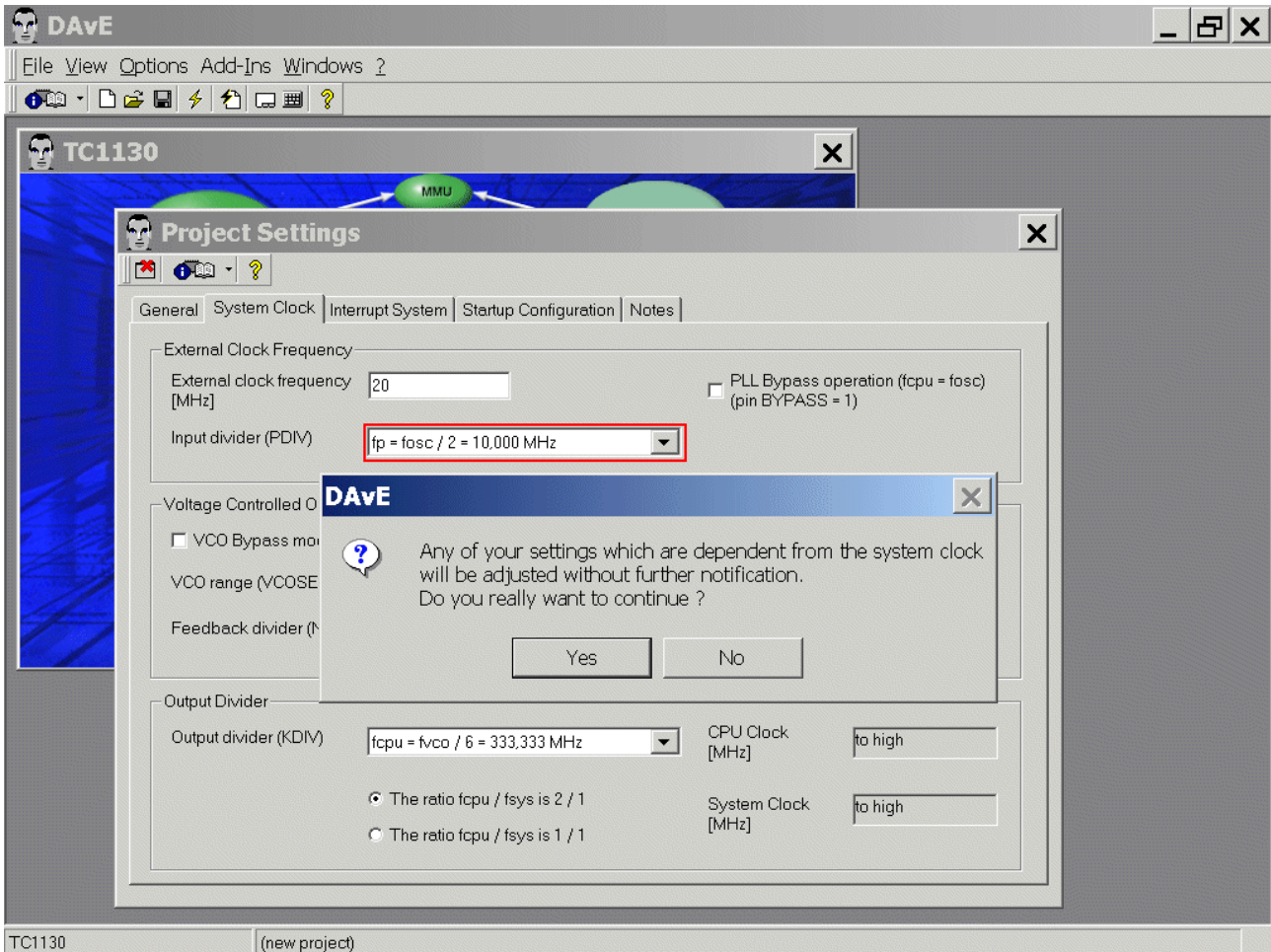


Note:

Validate each alpha numeric entry by pressing ENTER.



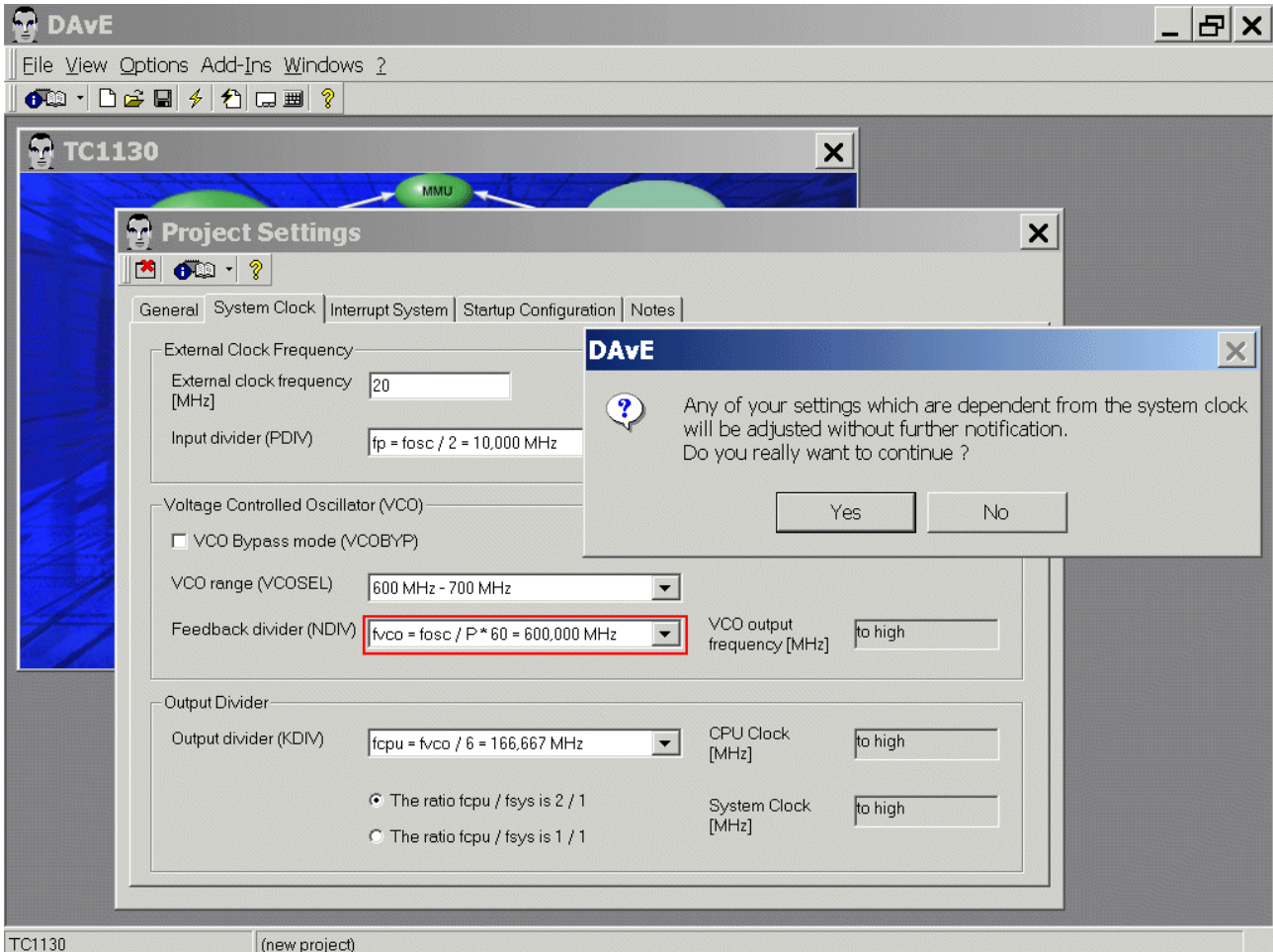
System Clock: External Clock Frequency: Input divider (PDIV) **select** $f_p = f_{osc} / 2 = 10,000$ MHz



Yes

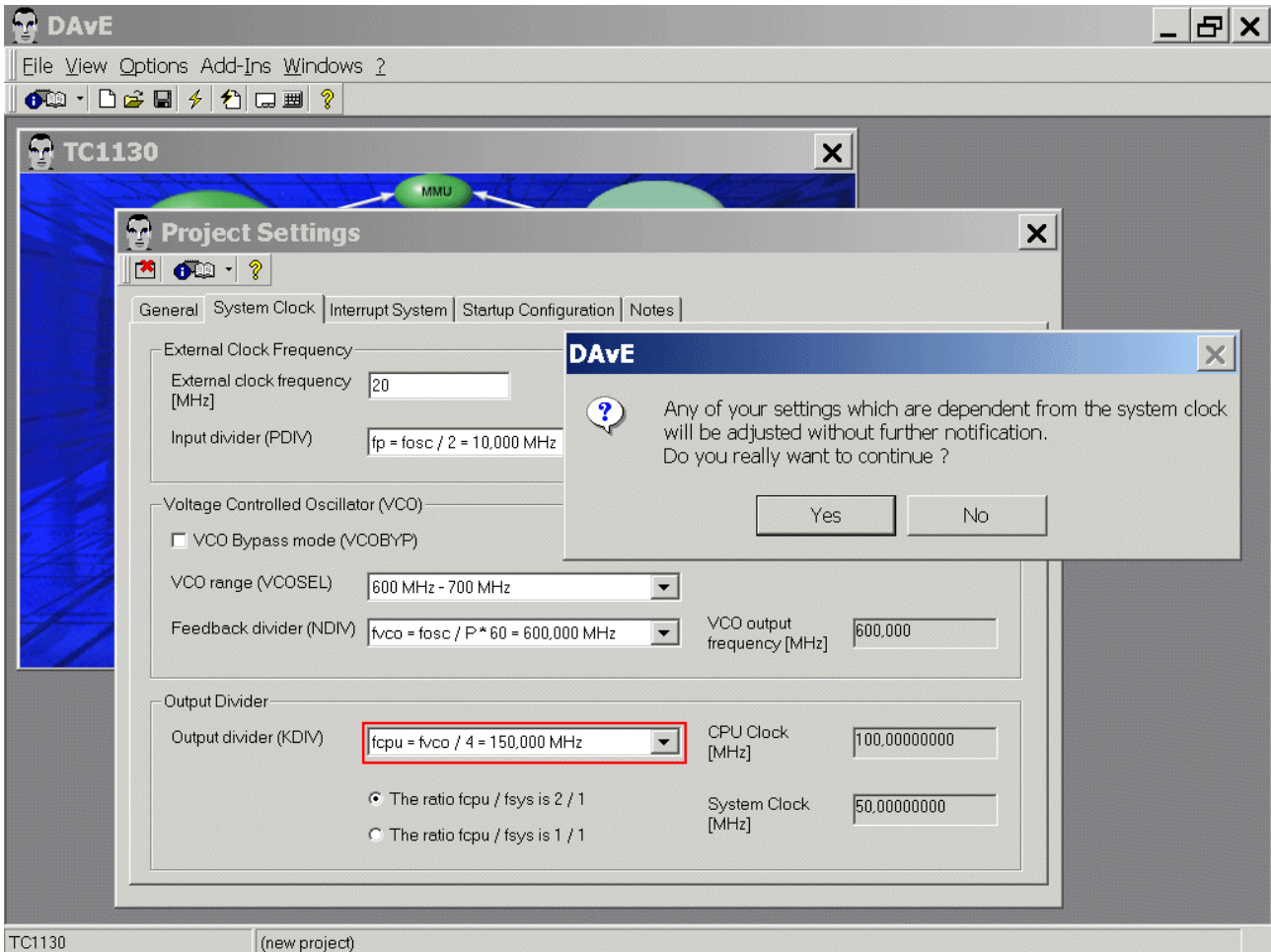
System Clock: Voltage Controlled Oscillator:

Feedback divider (NDIV) **select** $f_{vco} = f_{osc} / P * 60 = 600,000$ 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.

Project Settings

General | System Clock | Interrupt System | Startup Configuration | Notes

External Clock Frequency

External clock frequency [MHz] PLL Bypass operation (fcpu = fosc) (pin BYPASS = 1)

Input divider (PDIV)

Voltage Controlled Oscillator (VCO)

VCO Bypass mode (VCOBYP)

VCO range (VCOSEL)

Feedback divider (NDIV) VCO output frequency [MHz]

Output Divider

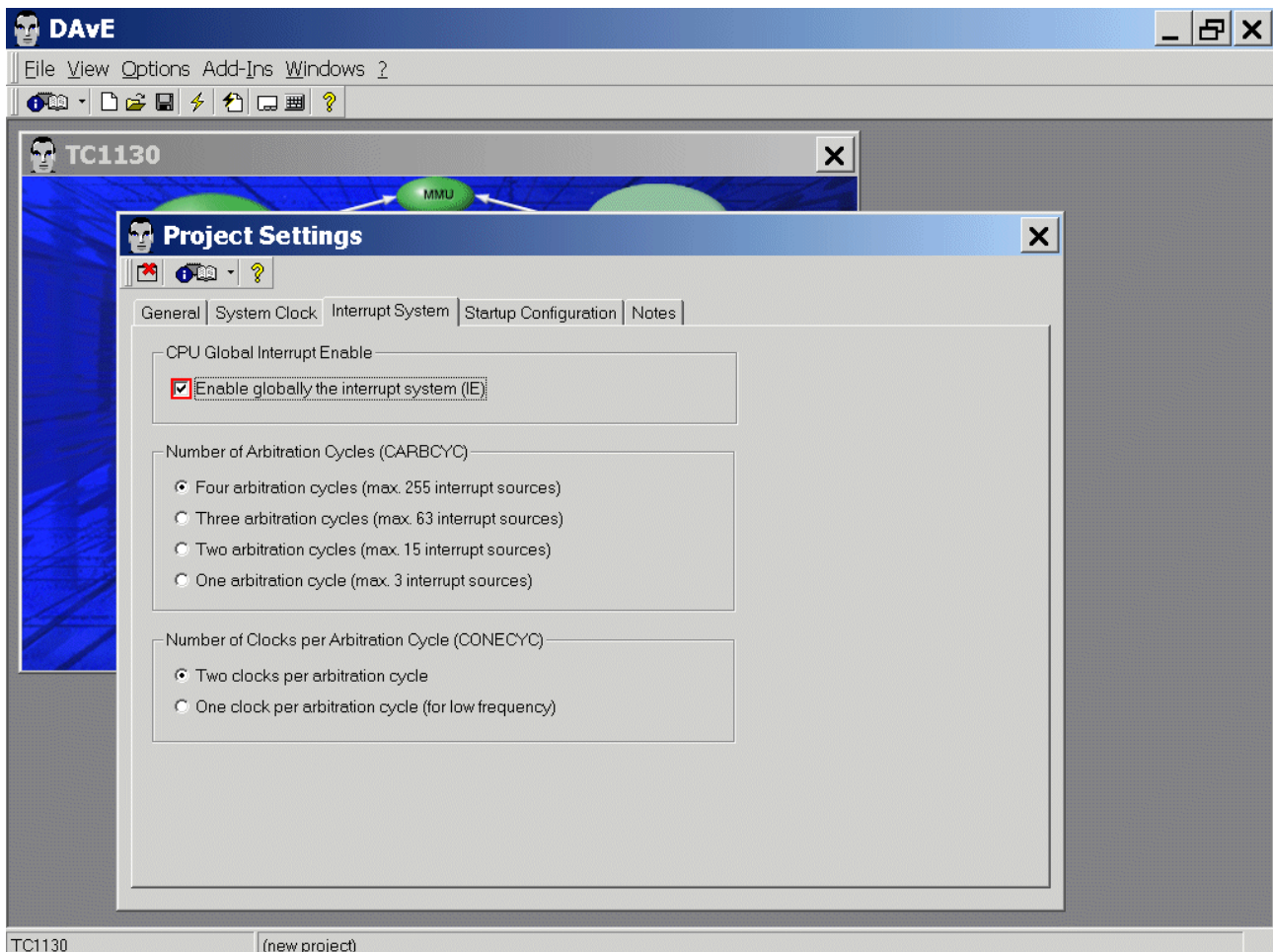
Output divider (KDIV) CPU Clock [MHz]

The ratio fcpu / fsys is 2 / 1

The ratio fcpu / fsys is 1 / 1

System Clock [MHz]

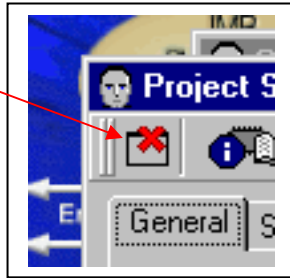
Interrupt System: CPU Global Interrupt Enable: click Enable globally the interrupt system (IE)



Startup Configuration: Hardware Booting Scheme: (do not care !!!)

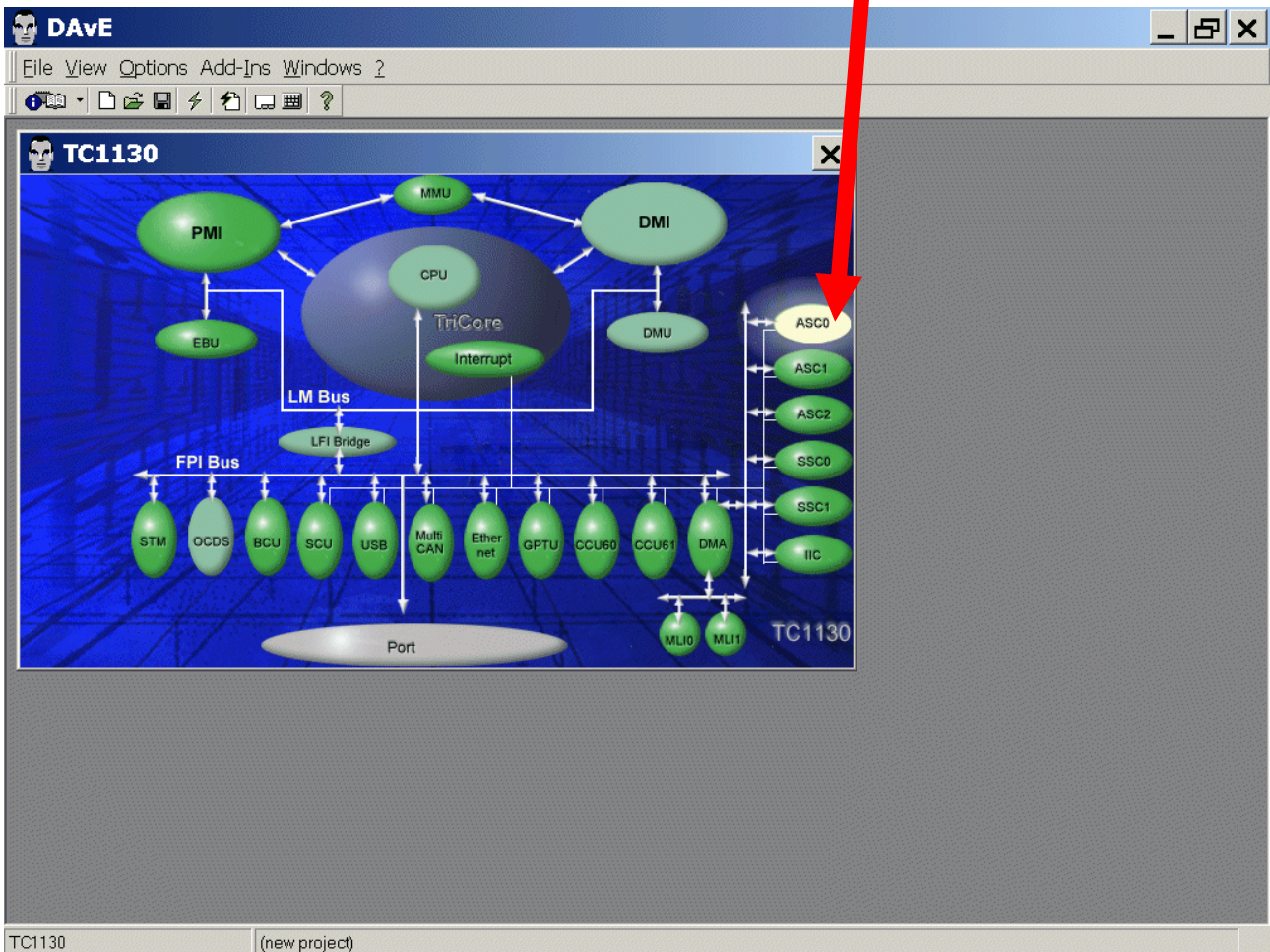
Notes: If you wish, you can insert your comments here.

Exit this dialog now by clicking  the close button:

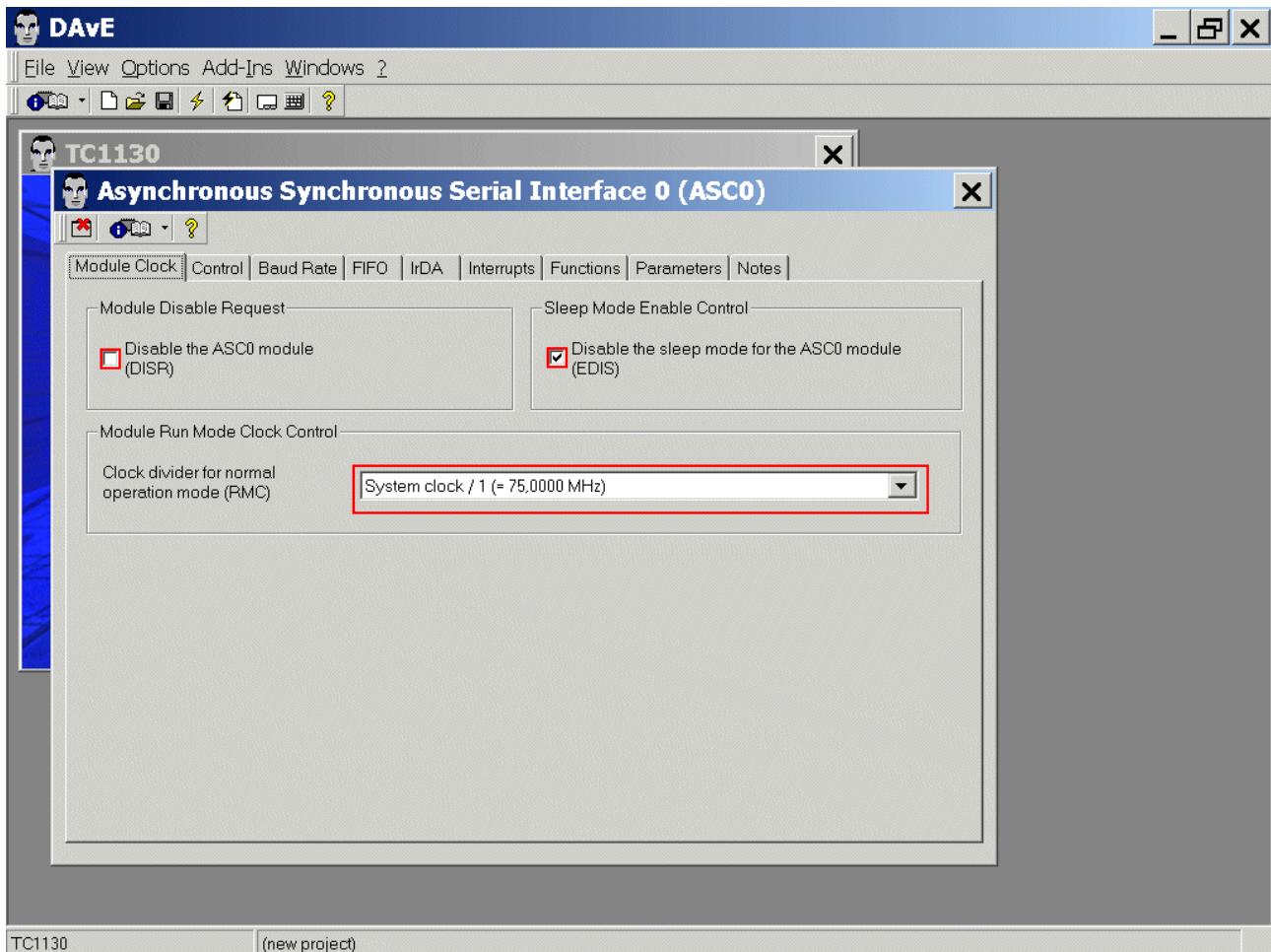


Configuration of the ASC0:

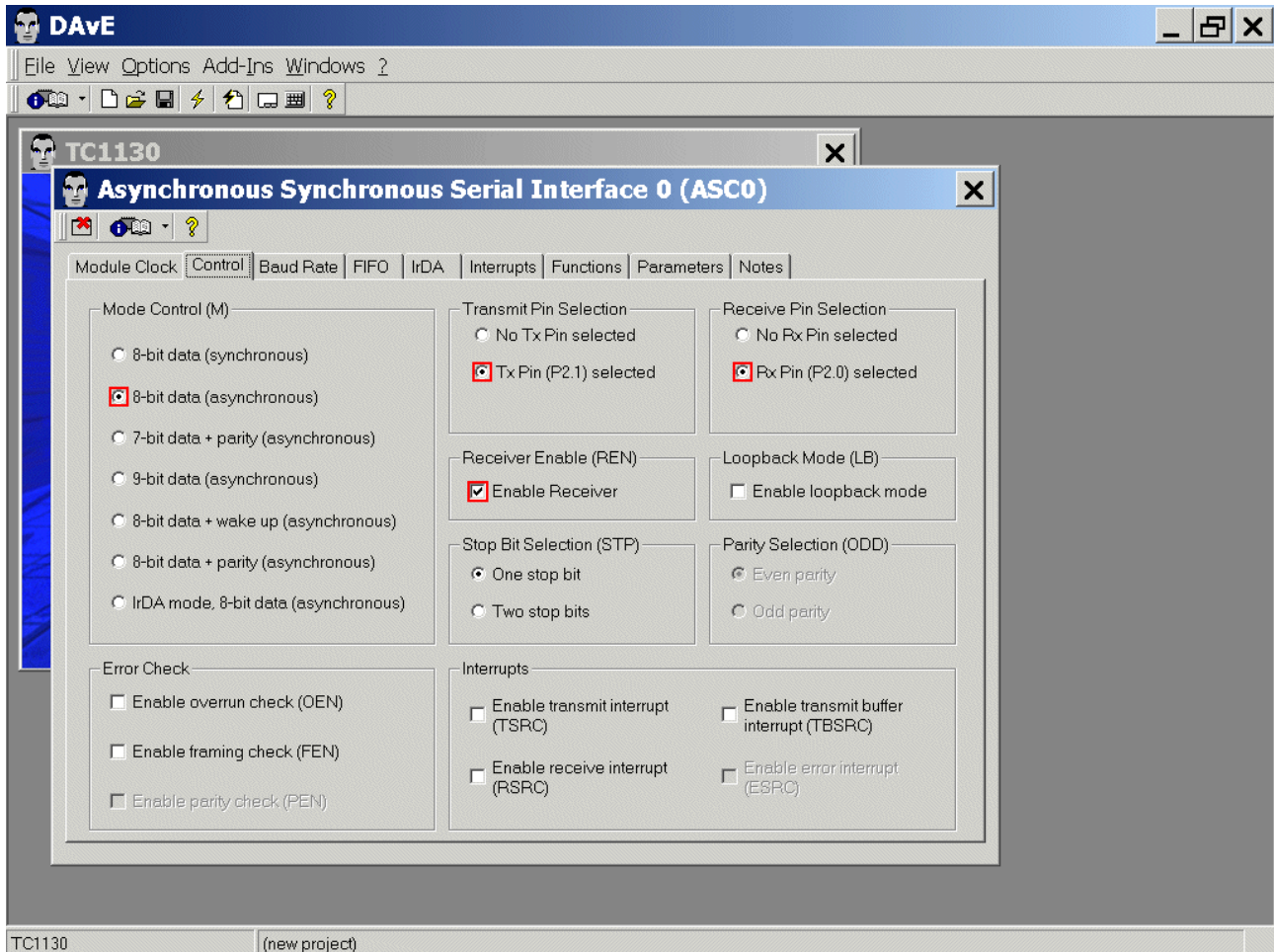
The configuration window can be opened by clicking the specific block/module.



- 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



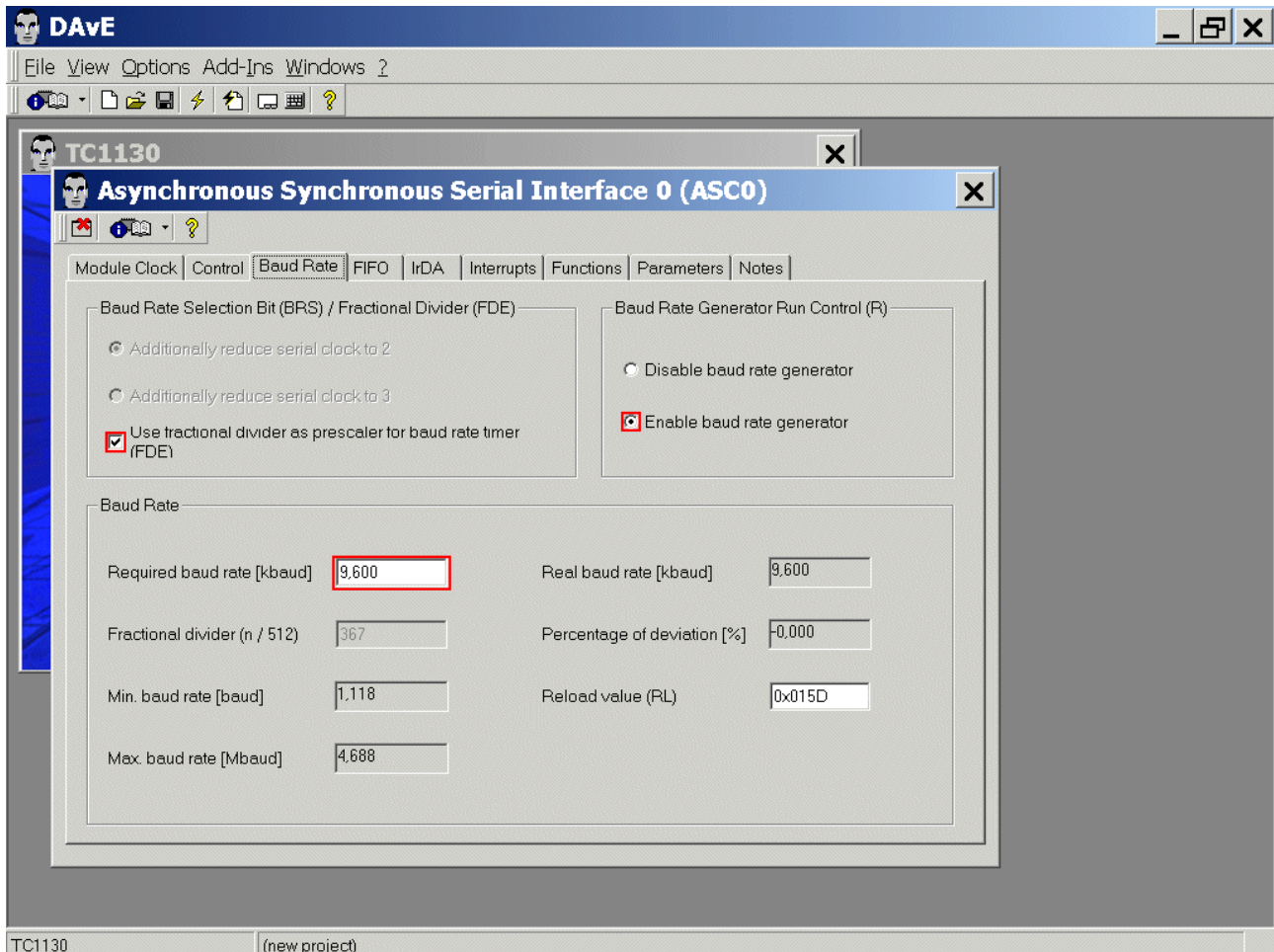
- Control: Mode Control (M): **click** 8-bit data (asynchronous)
- Control: Transmit Pin Selection: **click** TxPin (P2.1) selected
- Control: Receive Pin Selection: **click** RxPin (P2.0) selected
- Control: Receiver Enable: **click** Enable receiver (REN)



Baud Rate: Baud Rate: Required baud rate [kBaund] 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

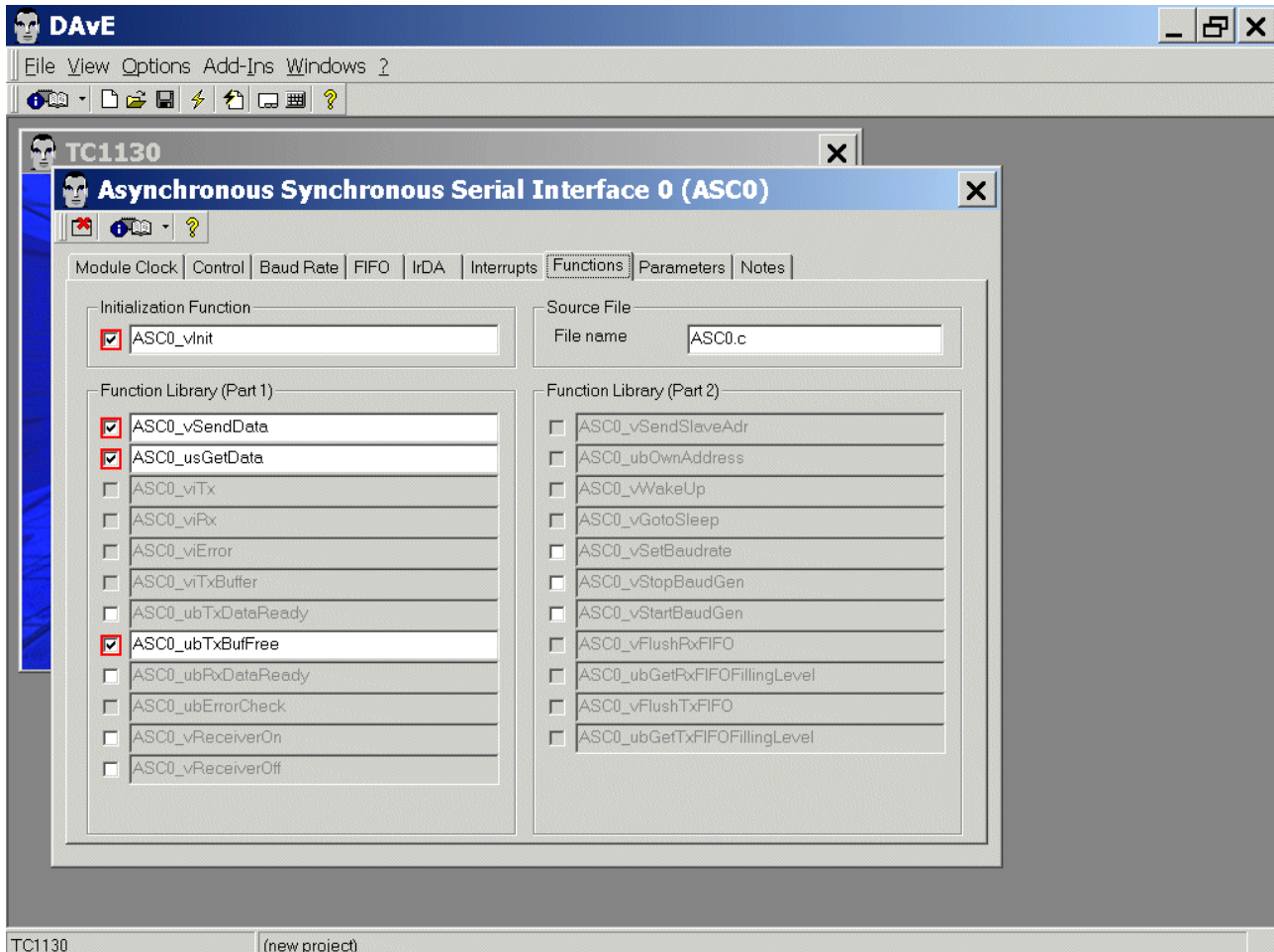


FIFO: (do nothing)

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

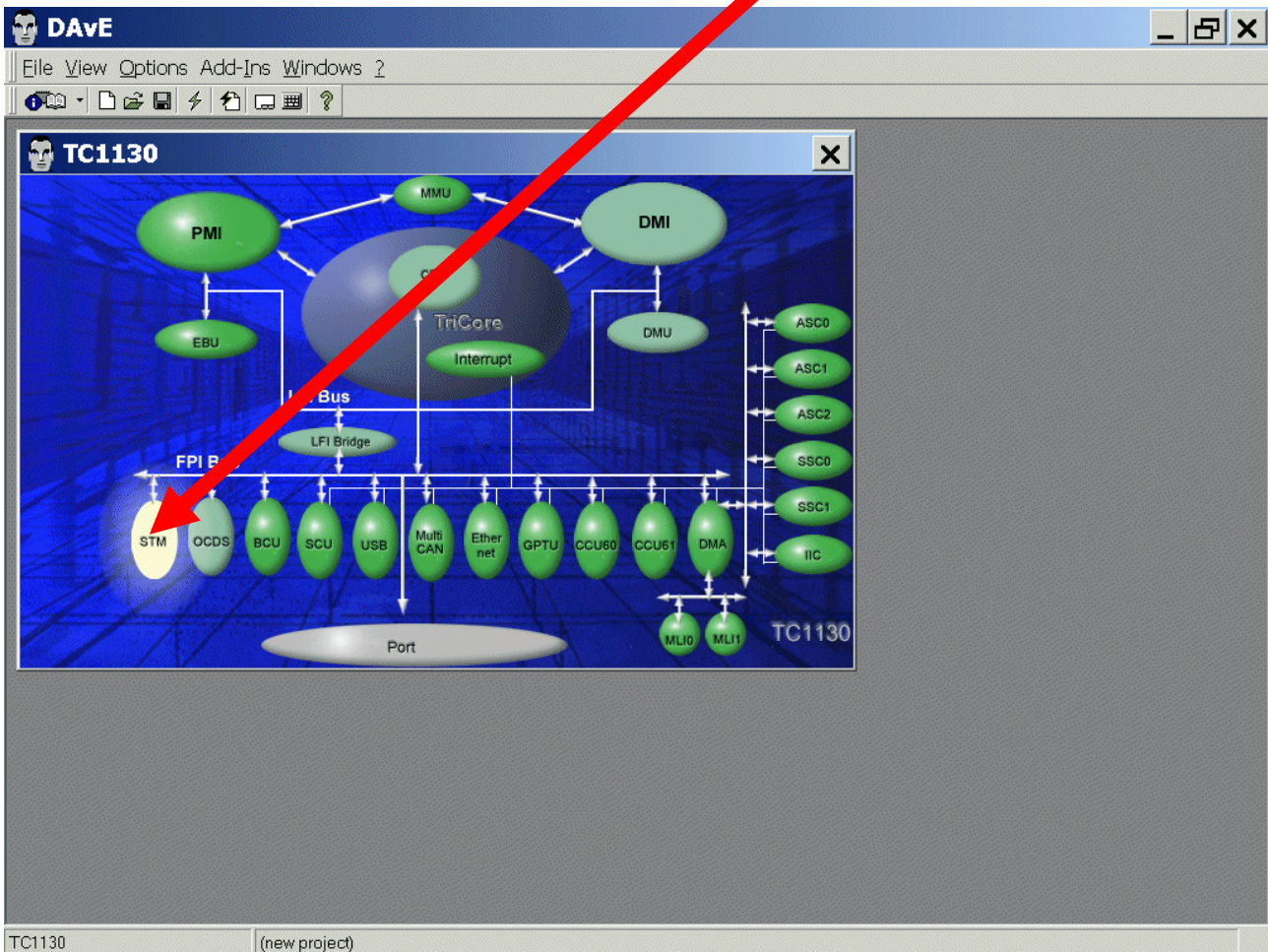


- Parameters: (do nothing)
 Notes: If you wish, you can insert your comments here.

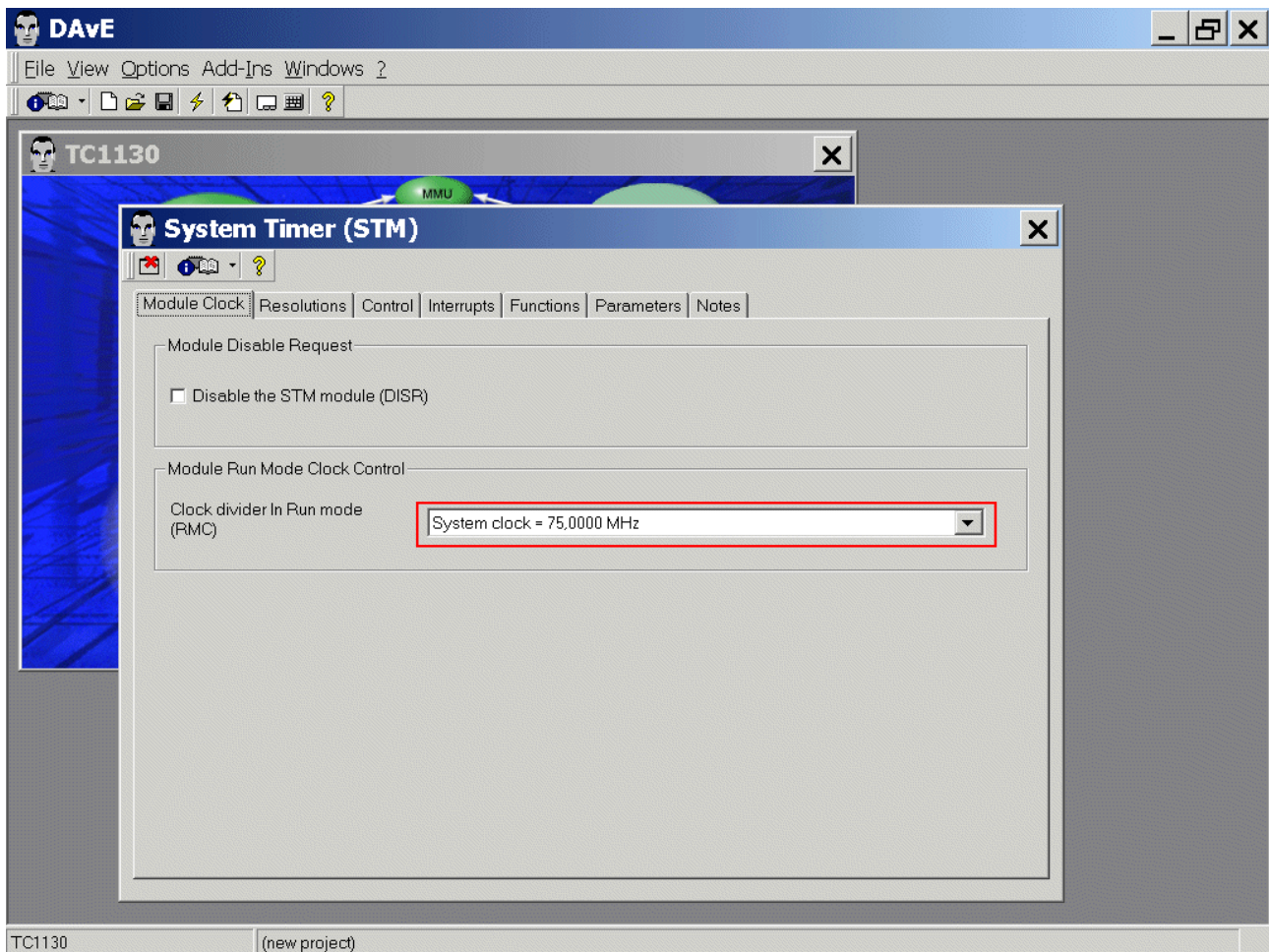
Exit this dialog now by clicking  the close button.

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)



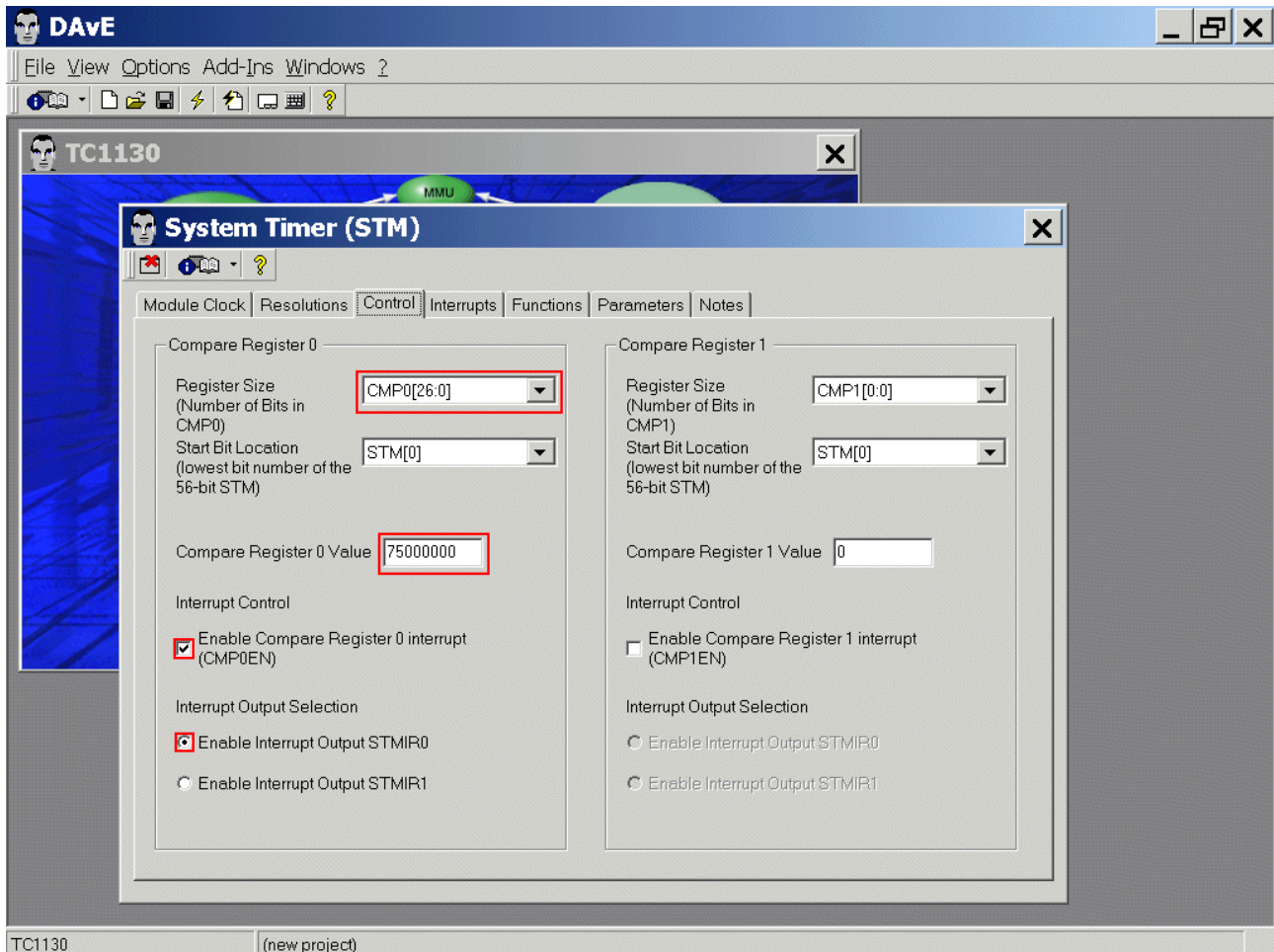
Resolutions: (do nothing)

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:

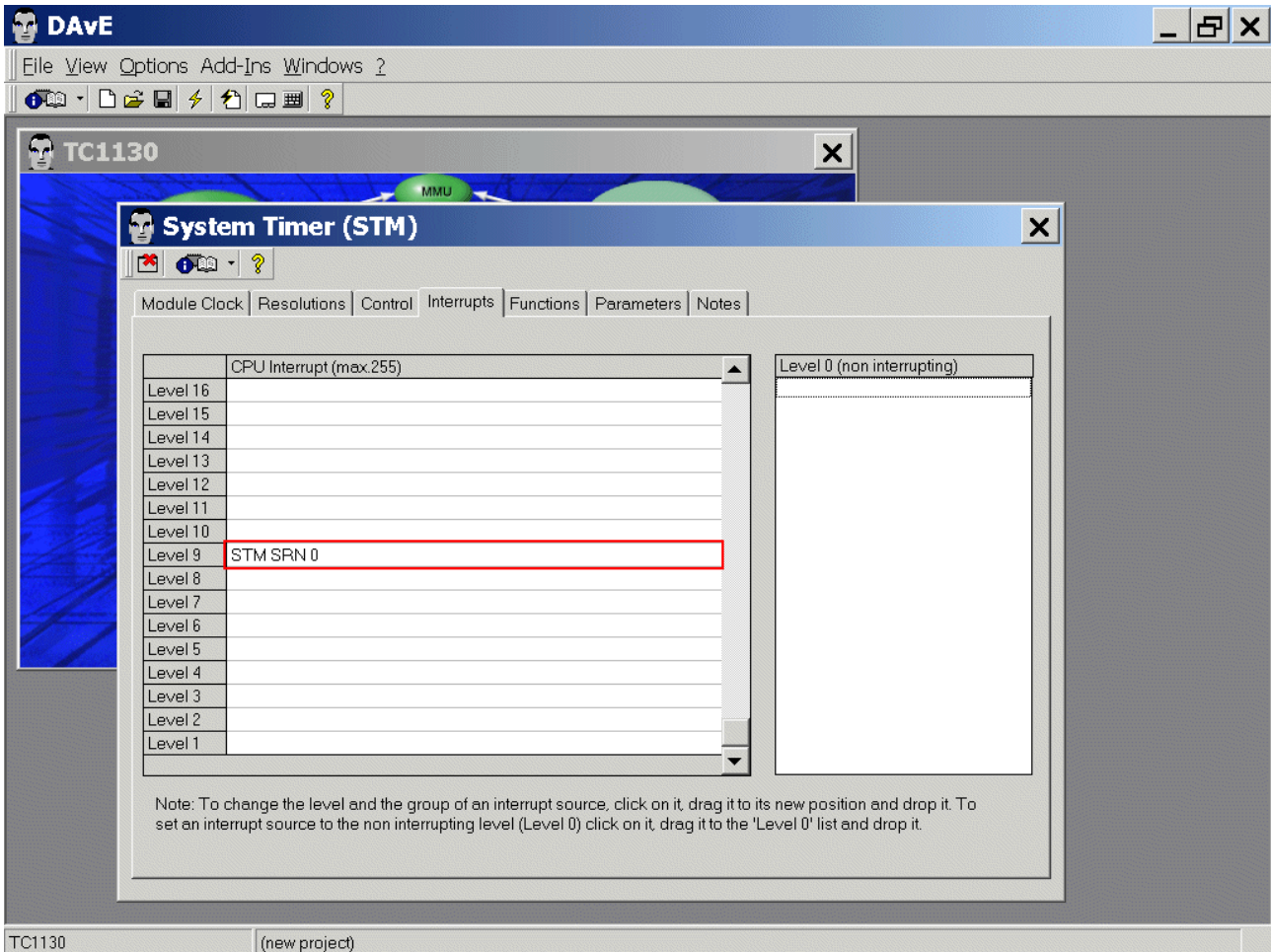
$$75.000.000 * 13,3333 \text{ ns} = 1 \text{ s}$$

There will be a System-Timer-Interrupt every 1 second.

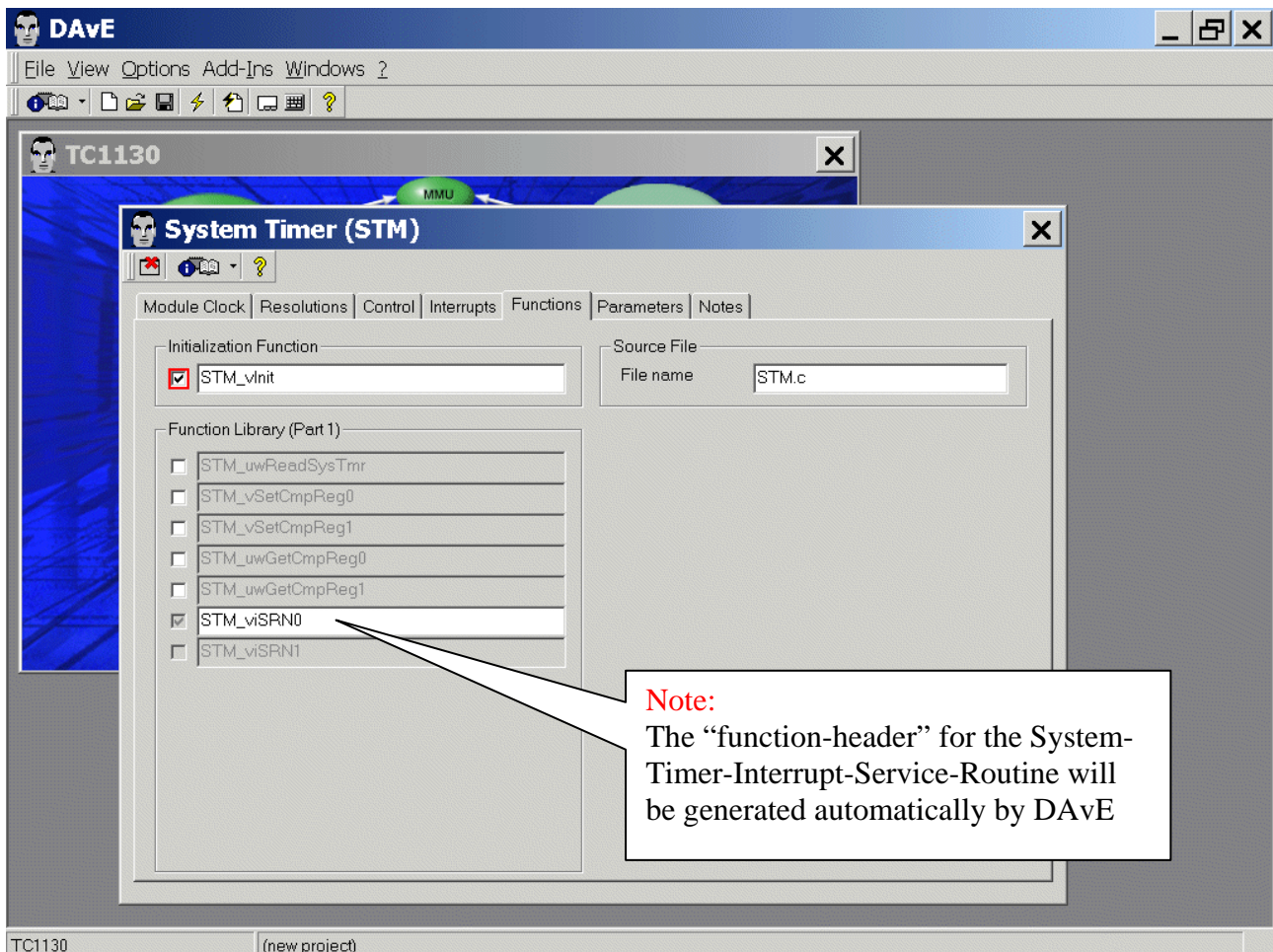
The LED on IO_Port_0_Pin_7 will be blinking at a frequency of 1 second (done in the System-Timer-Interrupt-Service-Routine).



Interrupts: drag and drop STM SRN 0 from Level 0 (non interrupting) to Level 9



Functions: Initialization Function: click ✓ STM_vInit



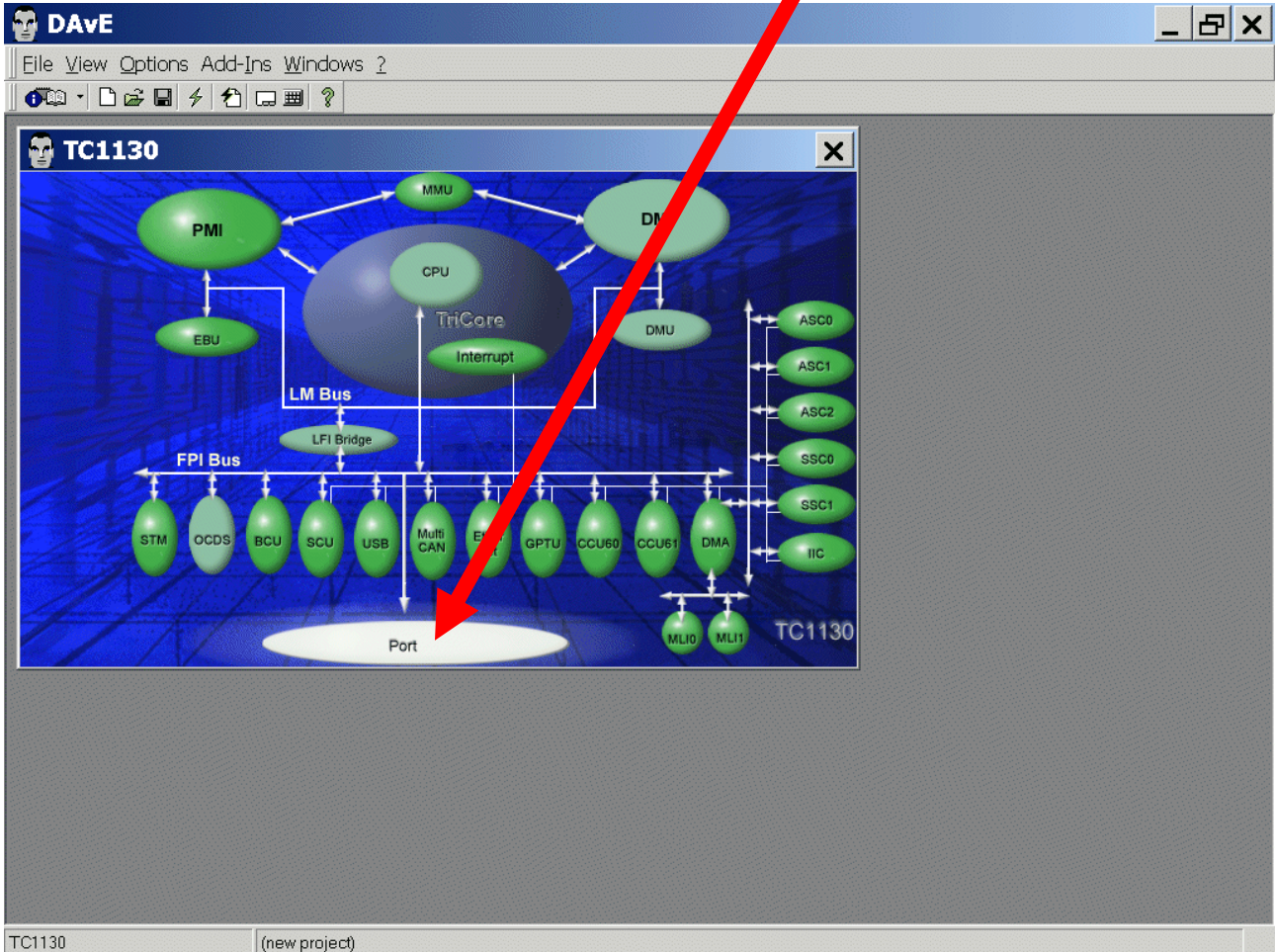
Parameters: (do nothing)

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

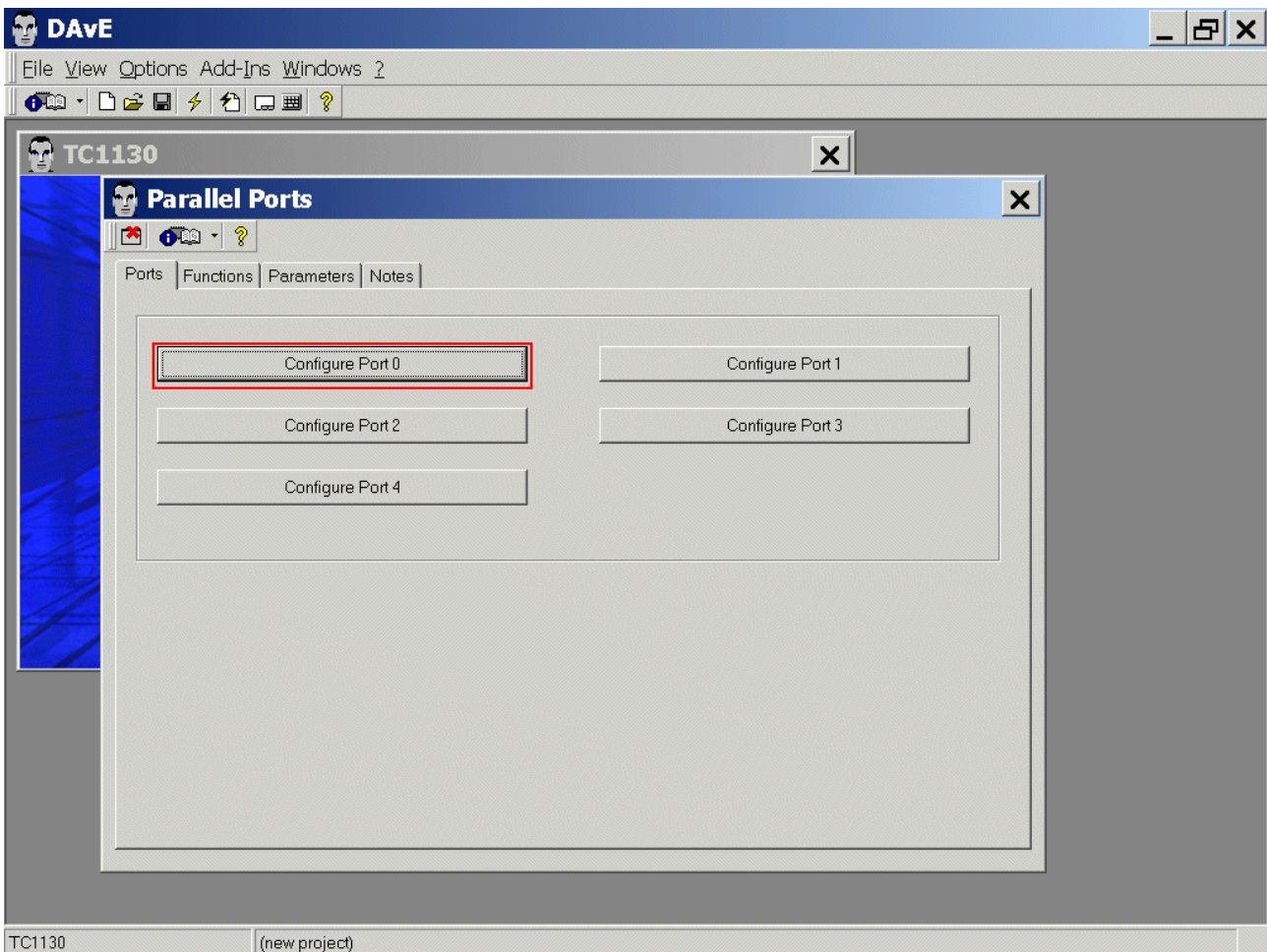
Exit this dialog now by clicking  the close button.

Port Configuration:

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

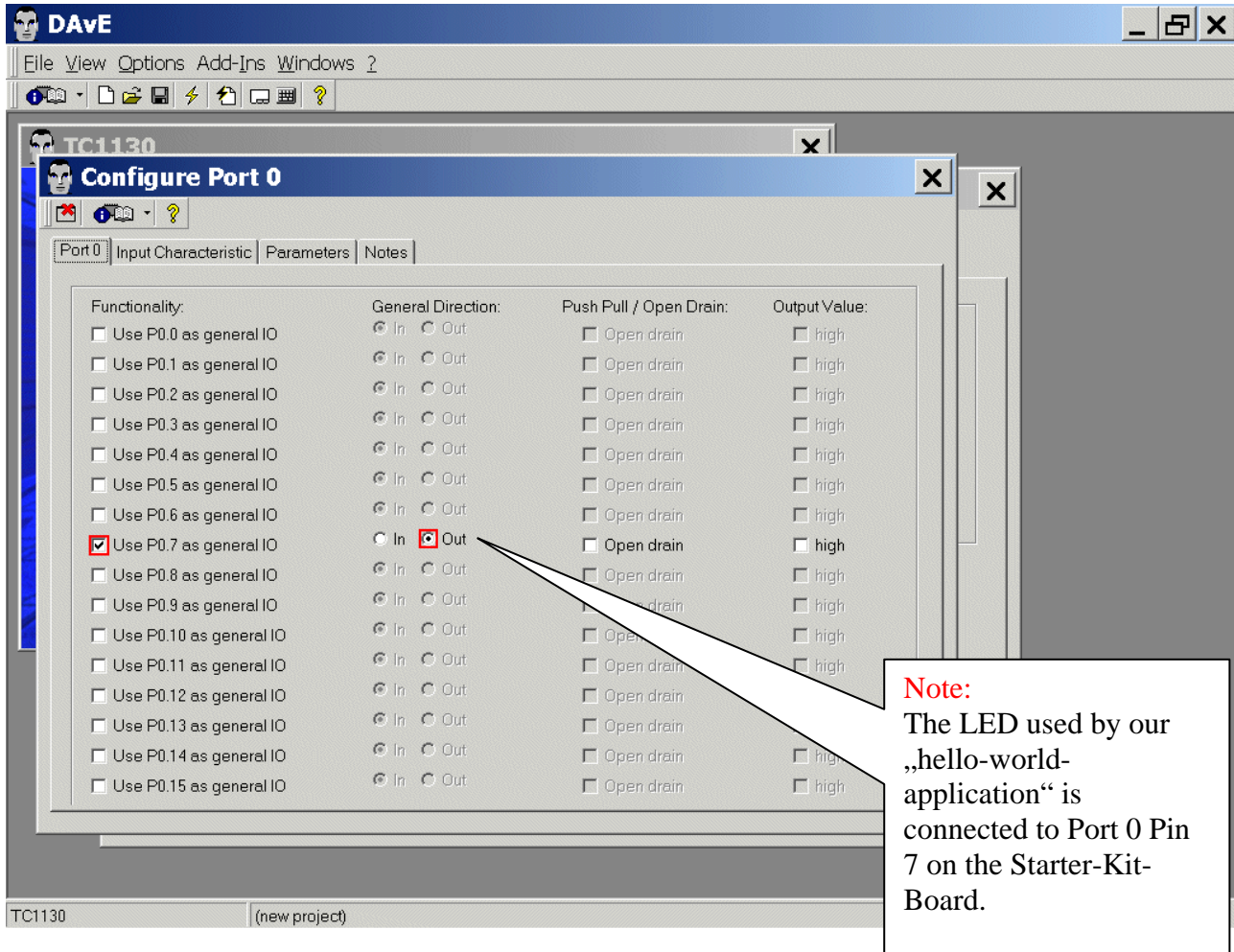


Ports: **click** Configure Port 0



Ports: Configure Port 0

Port 0: **Functionality:** click ✓ Use P0.7 as general IO, **General Direction:** click ⊙ Out



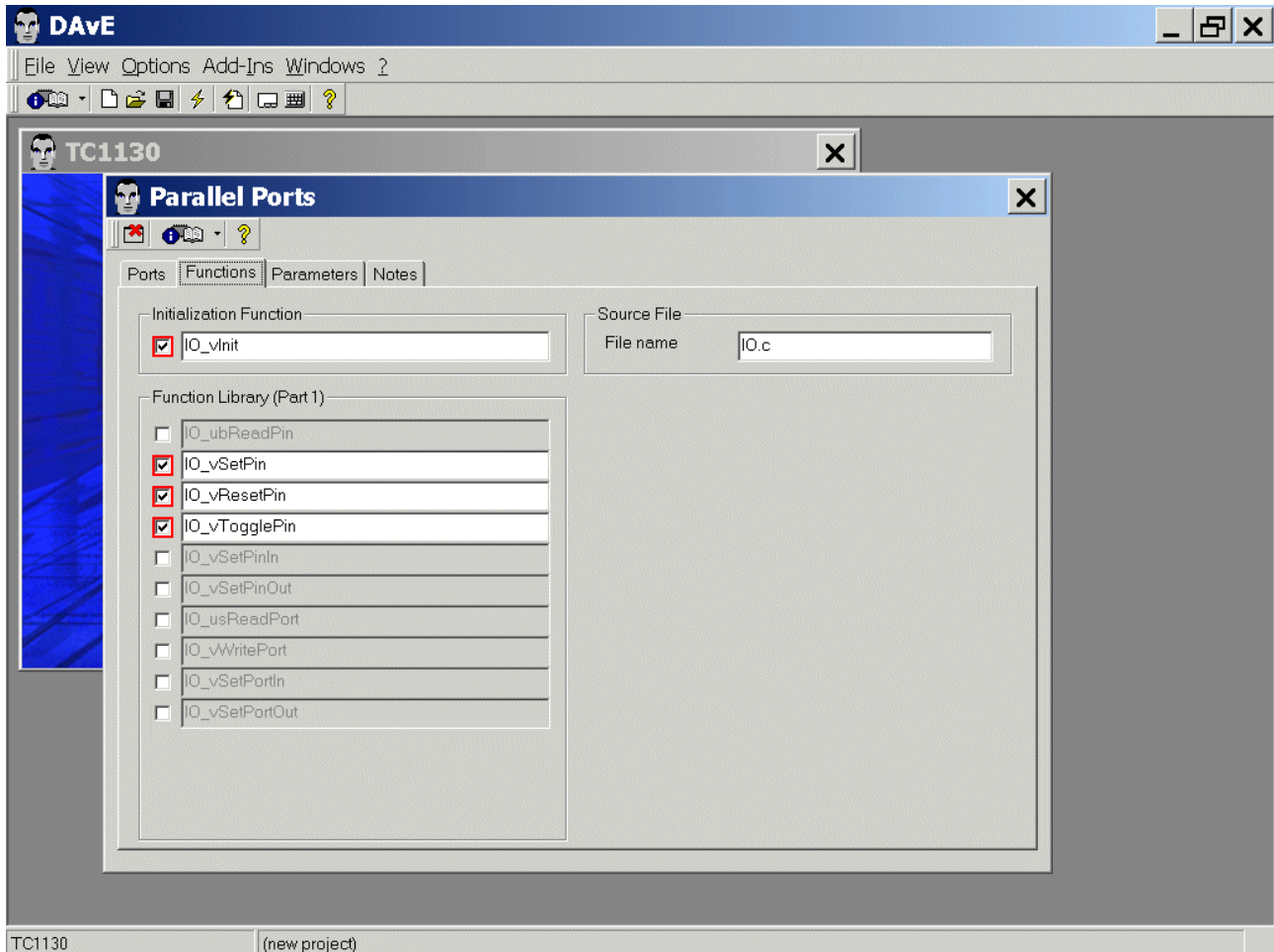
Input Characteristic: (do nothing)

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)

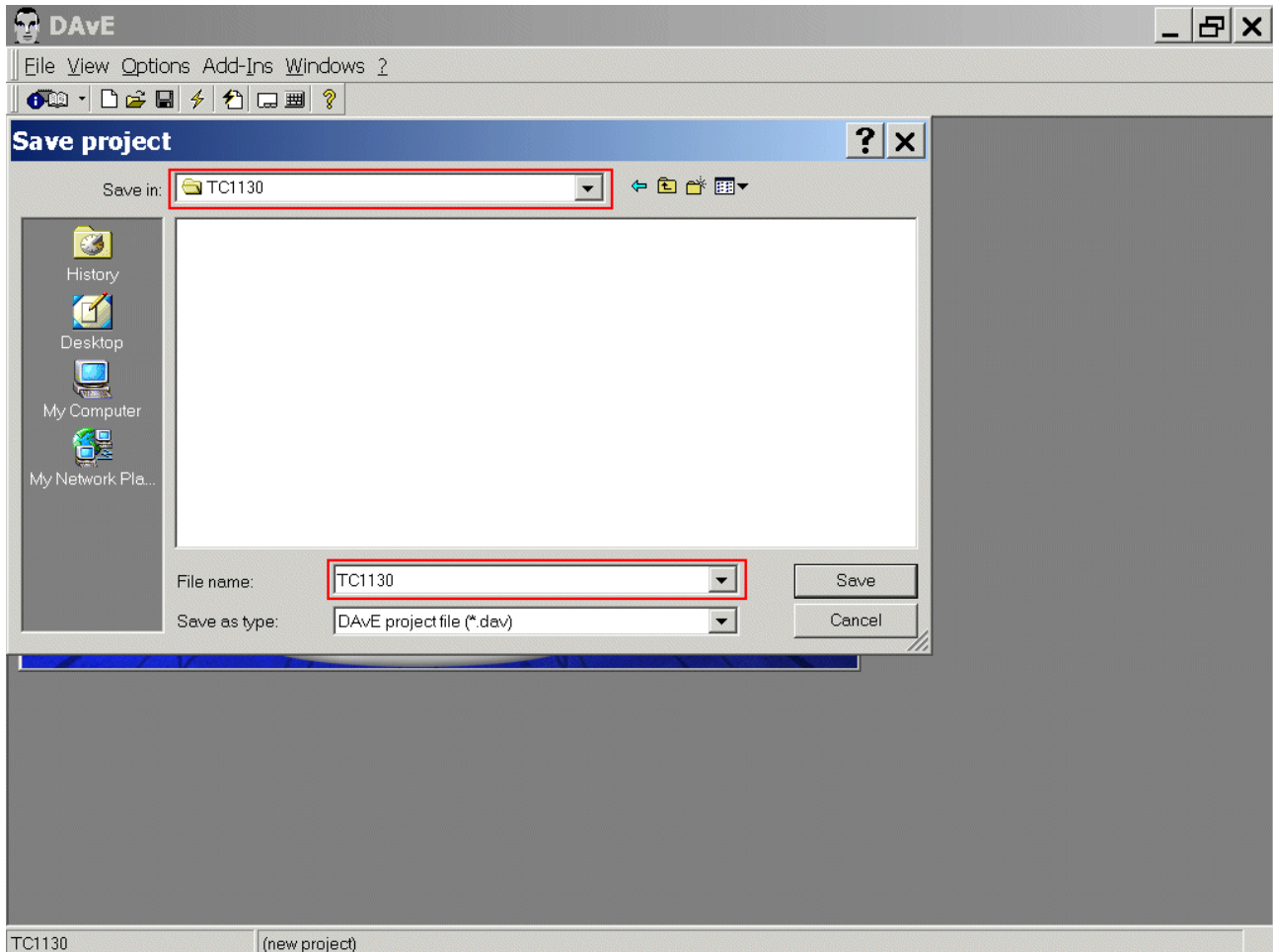
Notes: If you wish, you can insert your comments here.

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



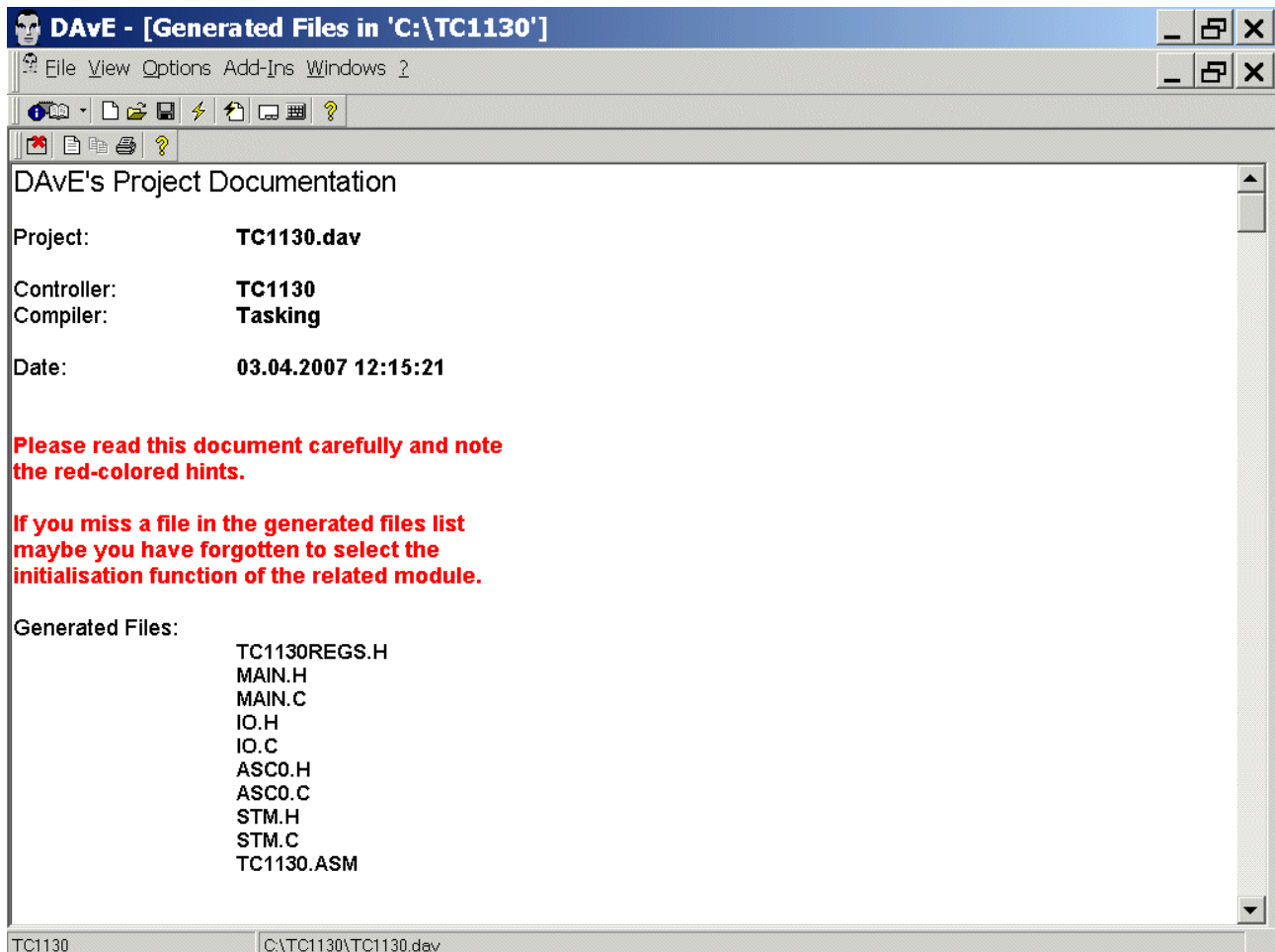
Save

Generate Code:

<p>File Generate Code</p>	<p>or click </p>
---	---



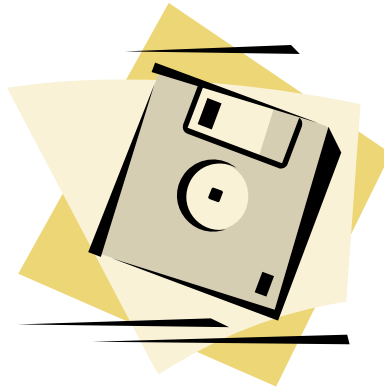
DAvE will show you all the files he has generated (File Viewer opens automatically).



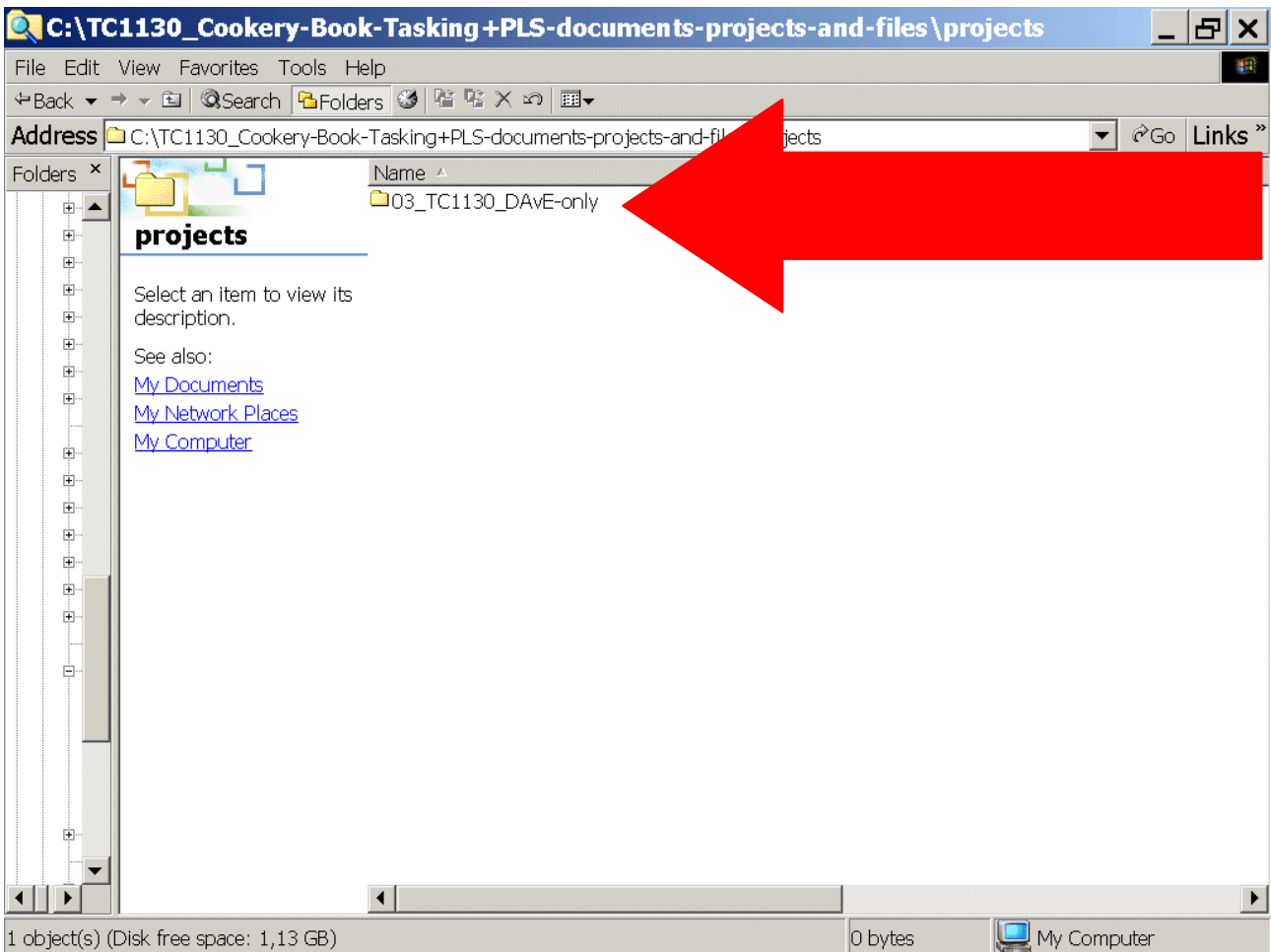
File
Exit

Save changes?

click Yes



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

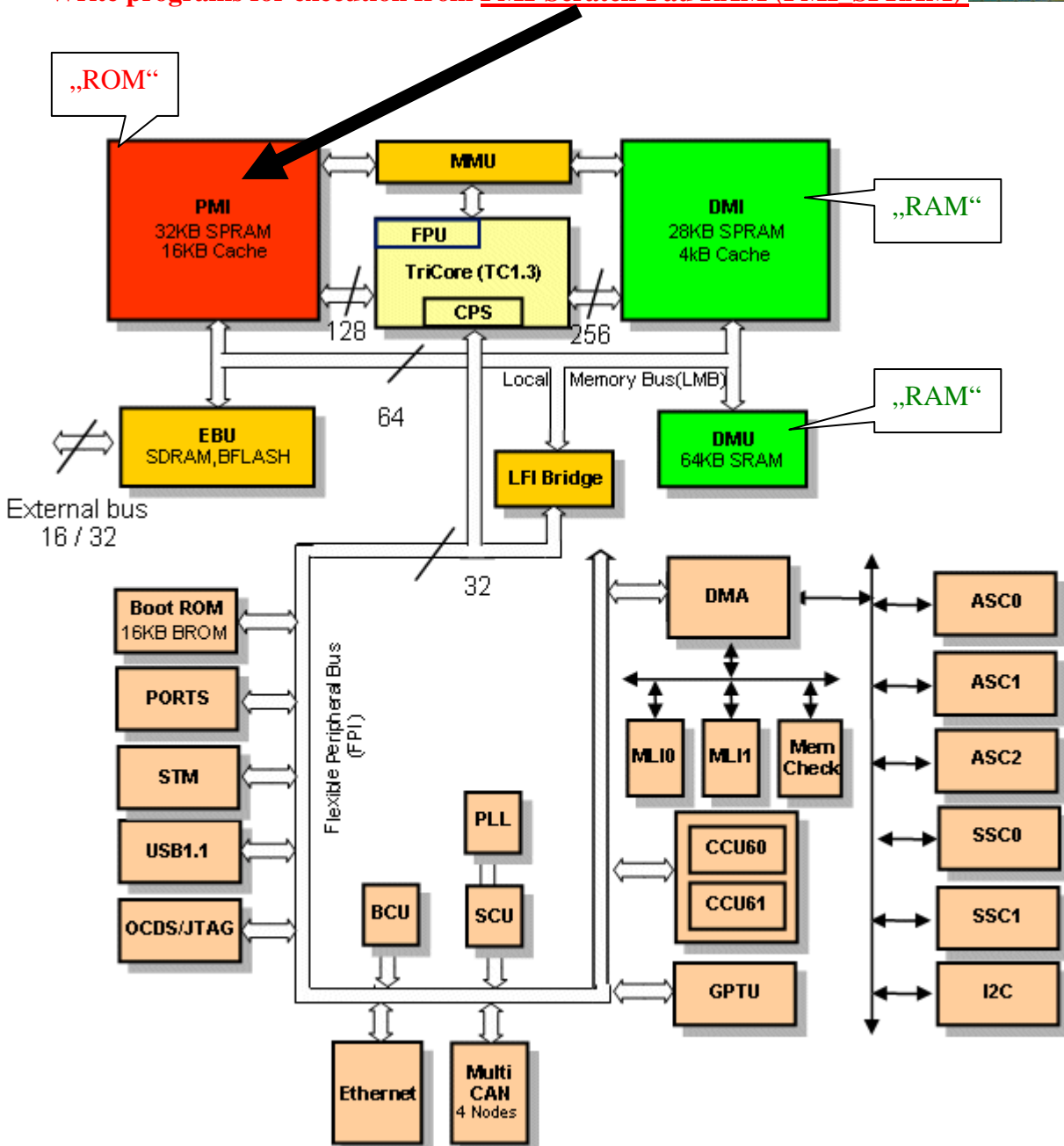


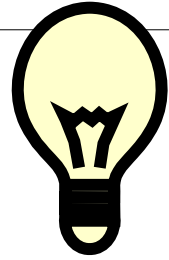
4.) Using of the TASKING - EDE Development Tools:

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

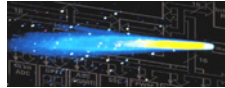


Write programs for execution from PMI-Scratch-Pad RAM (PMI_SPRAM)





Note:



“ROM”:

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



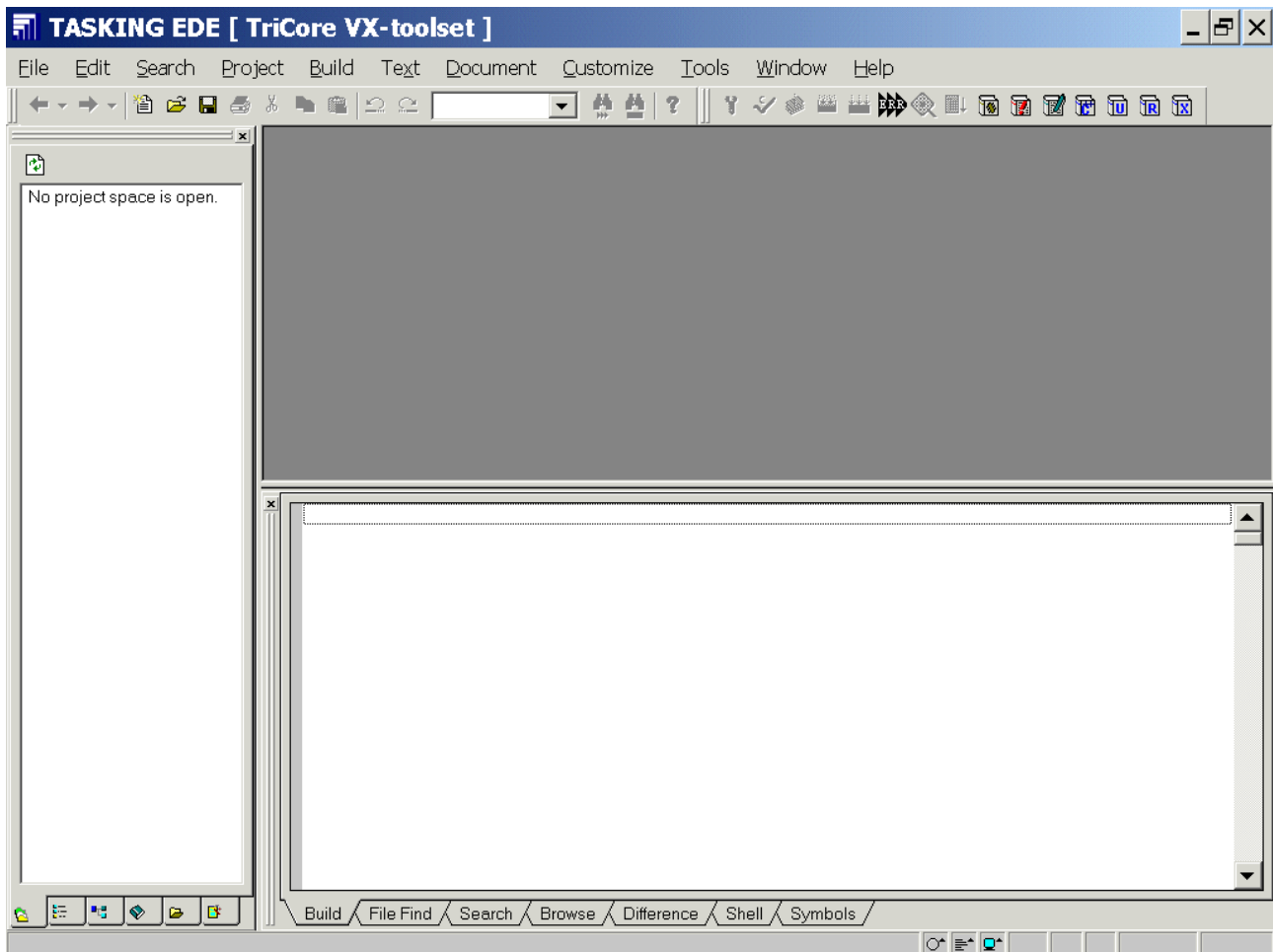
“RAM”:

Variables are located in “RAM-space”.

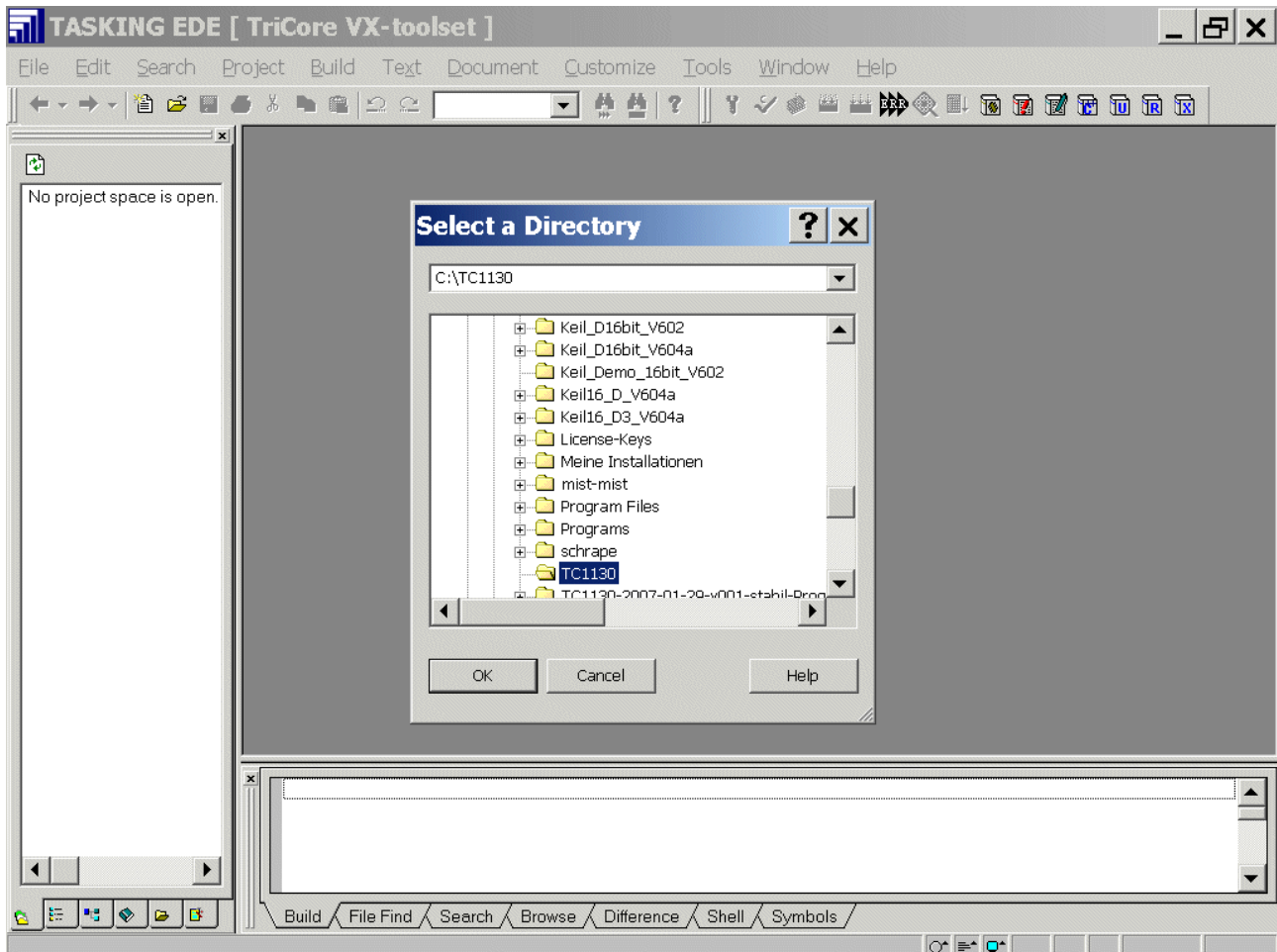
Install the Tasking Development Tools TriCore v2.3r1

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

If you see an open project – close it: **File – Close Project Space**

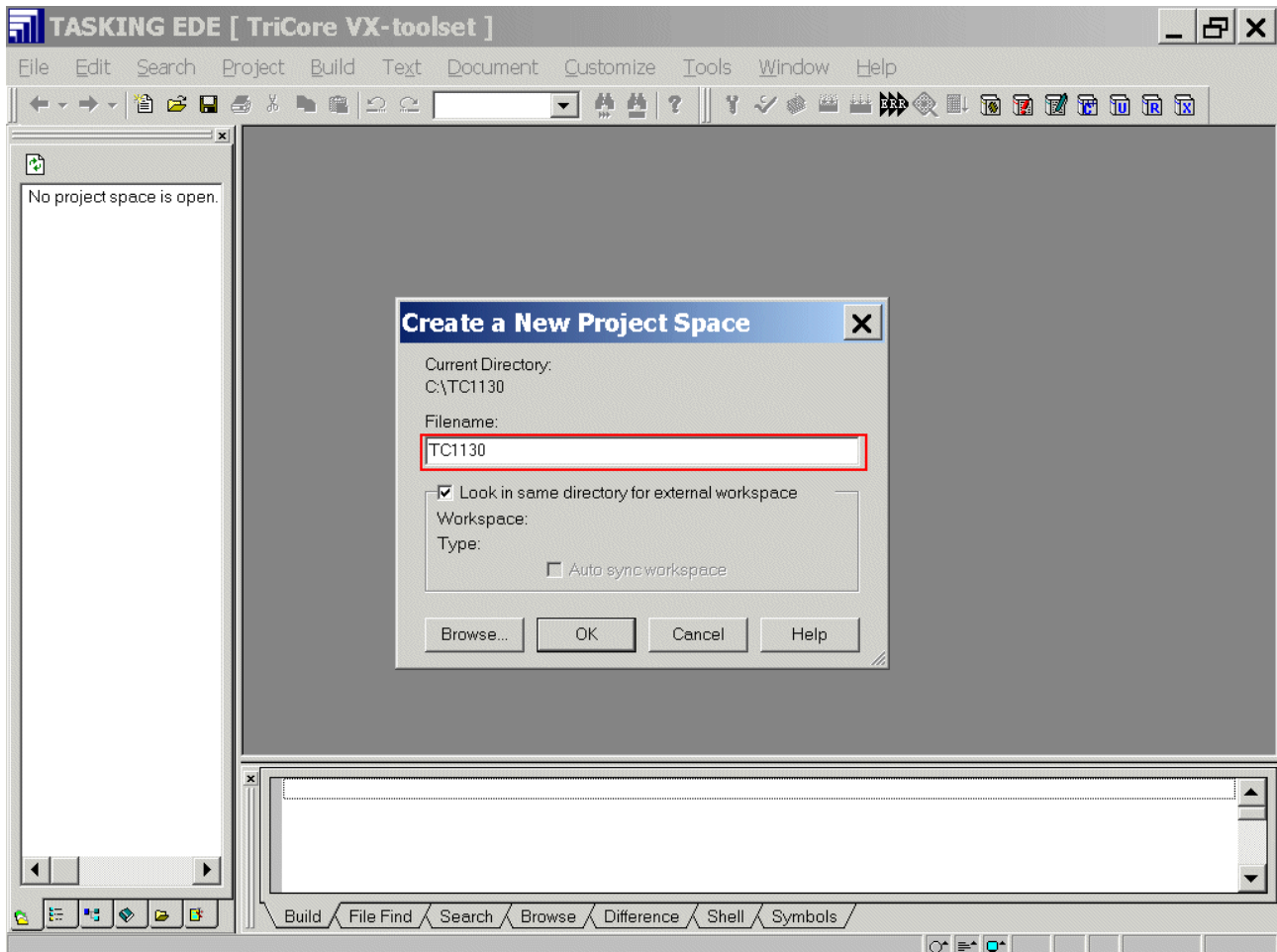


File
Change Directory
Choose C:\TC1130



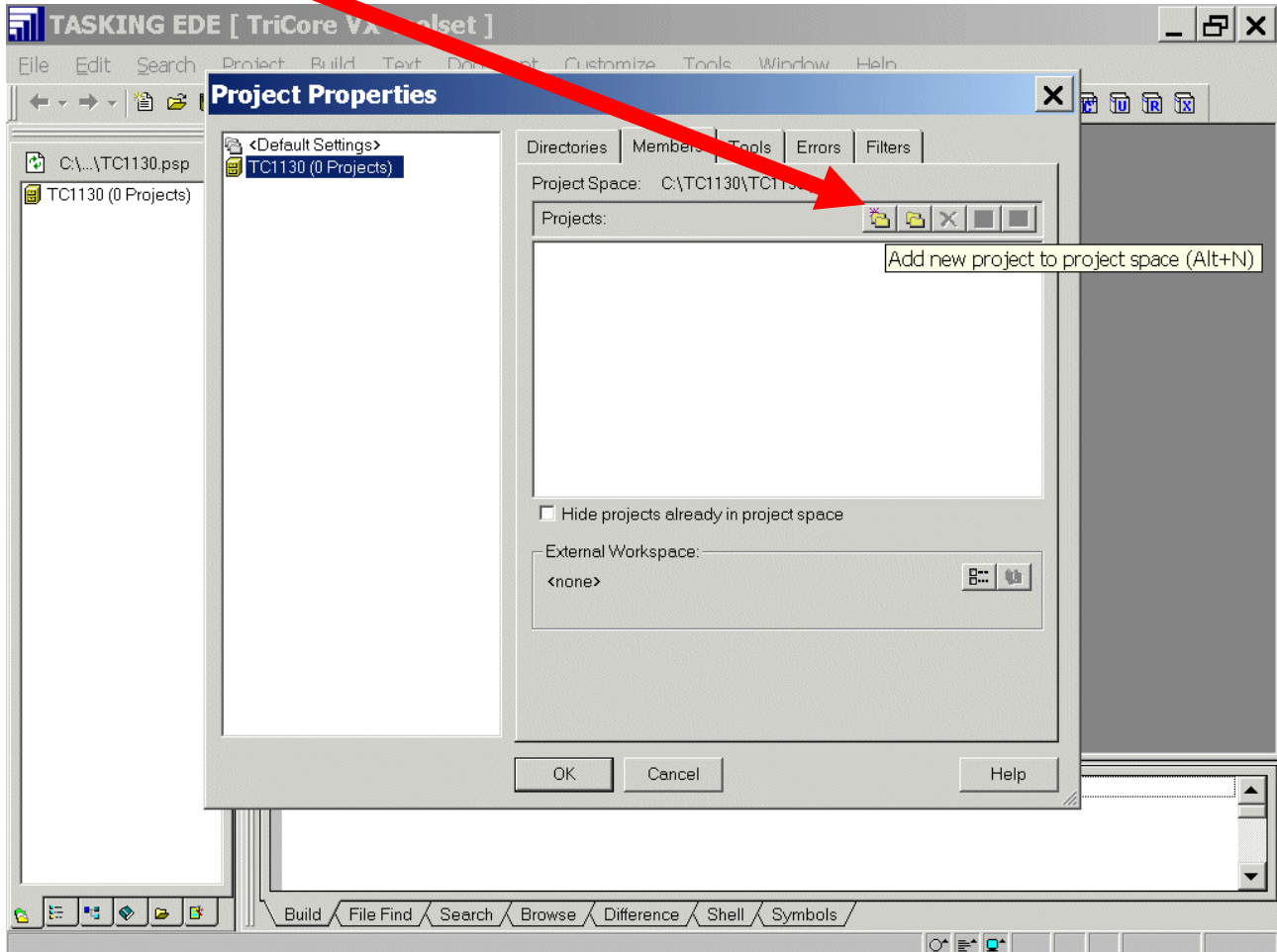
OK

File
New Project Space
Insert TC1130

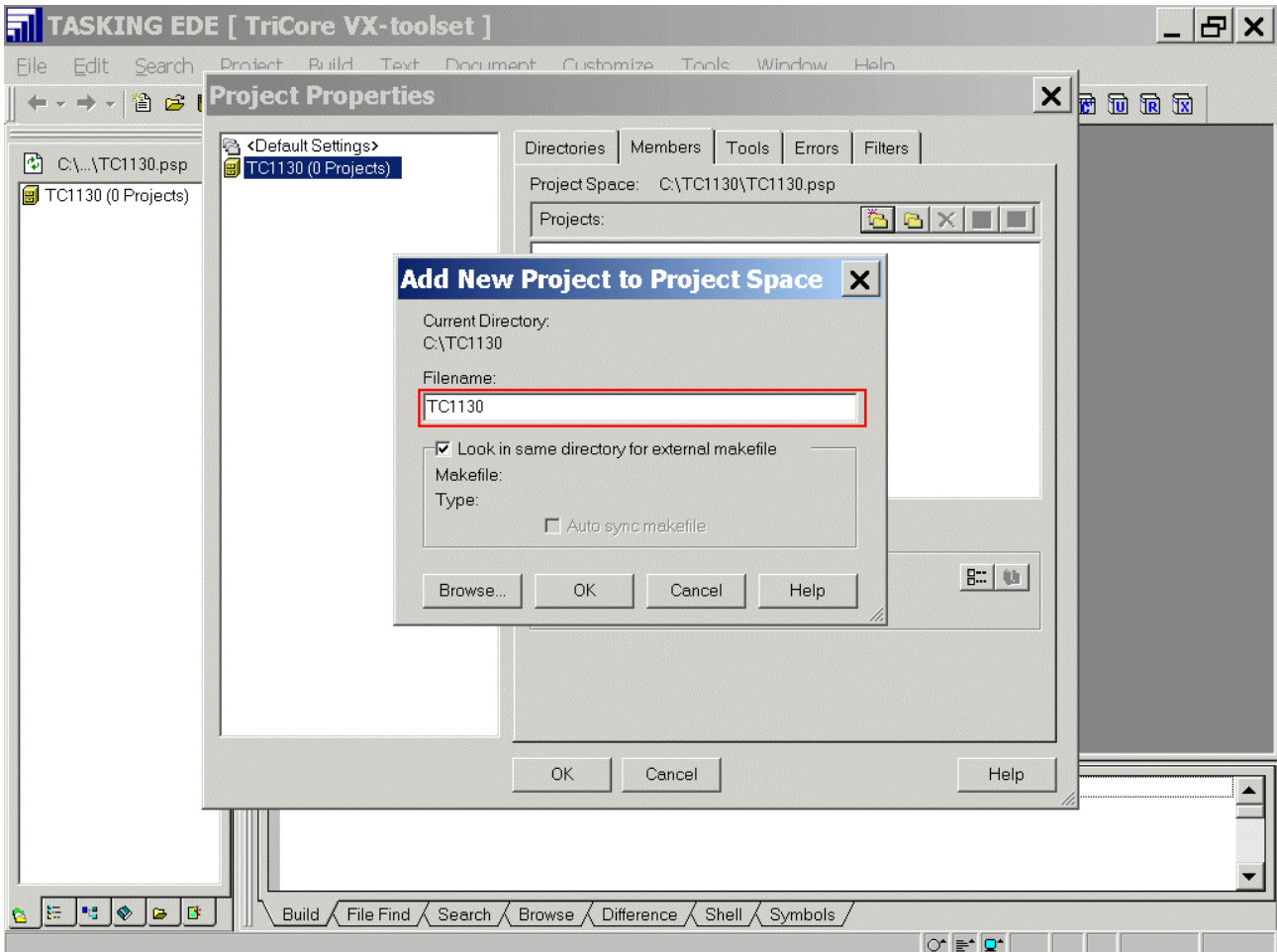


OK

Click: "Add new project to project space"

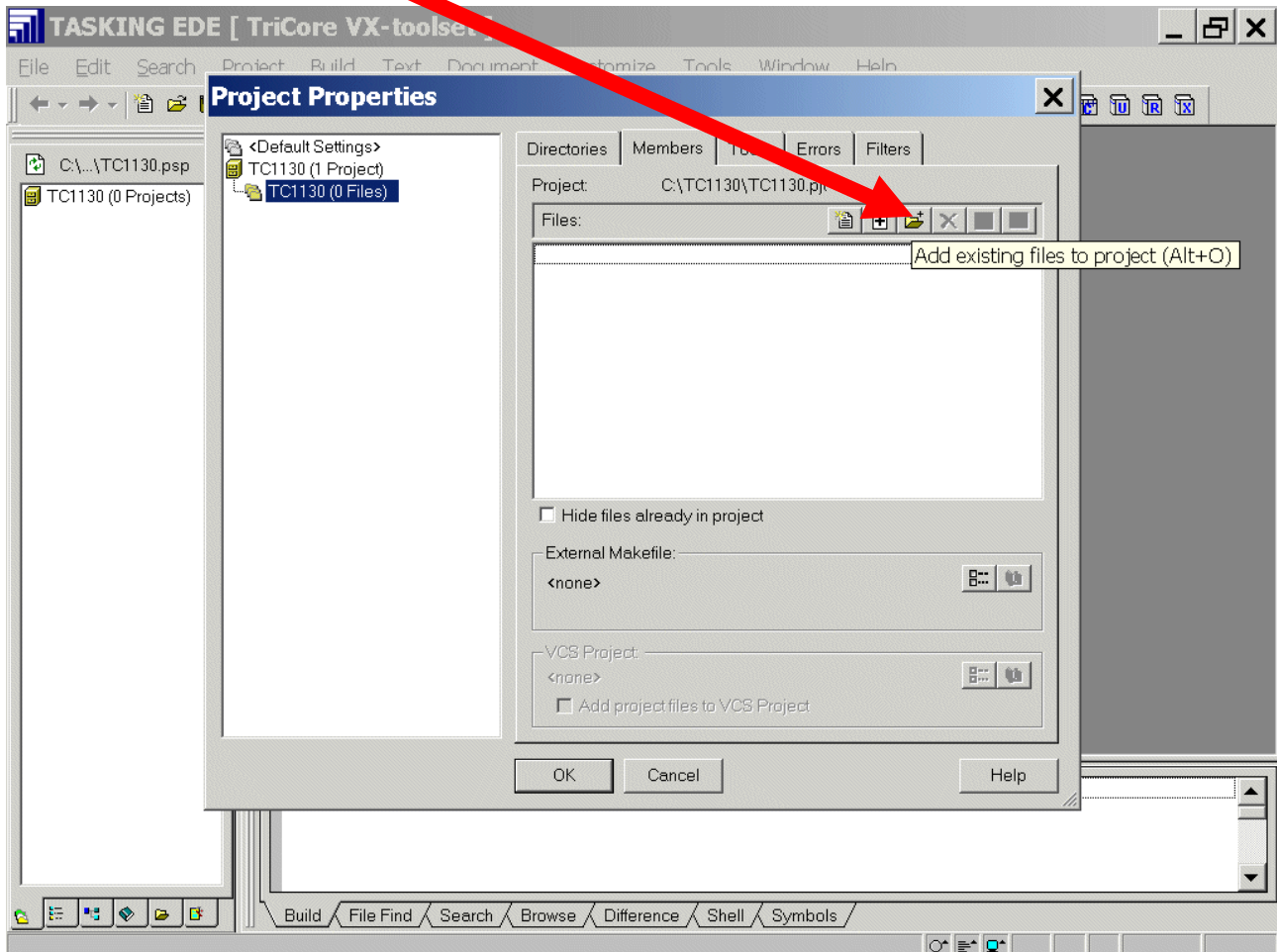


Insert TC1130

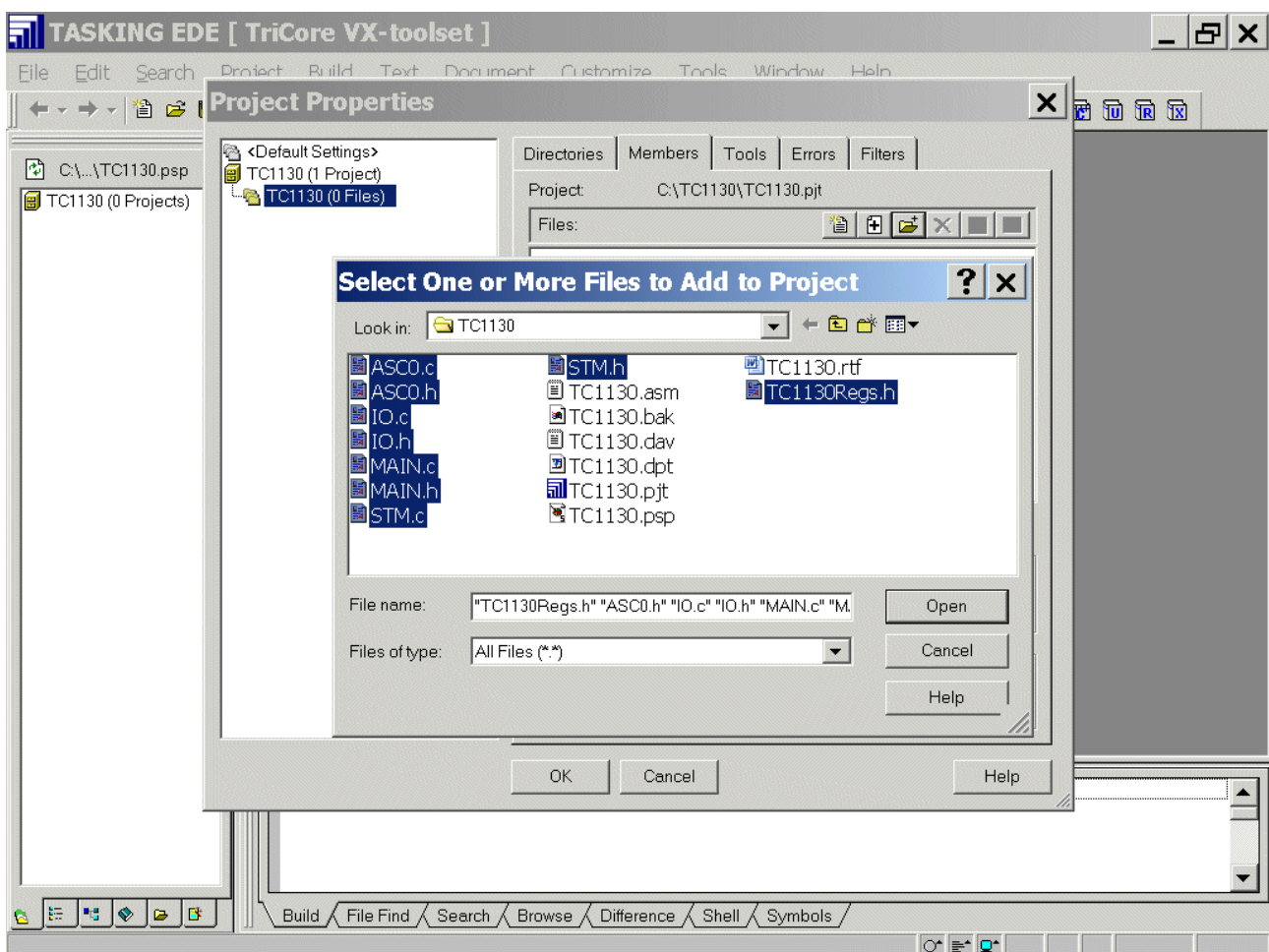


OK

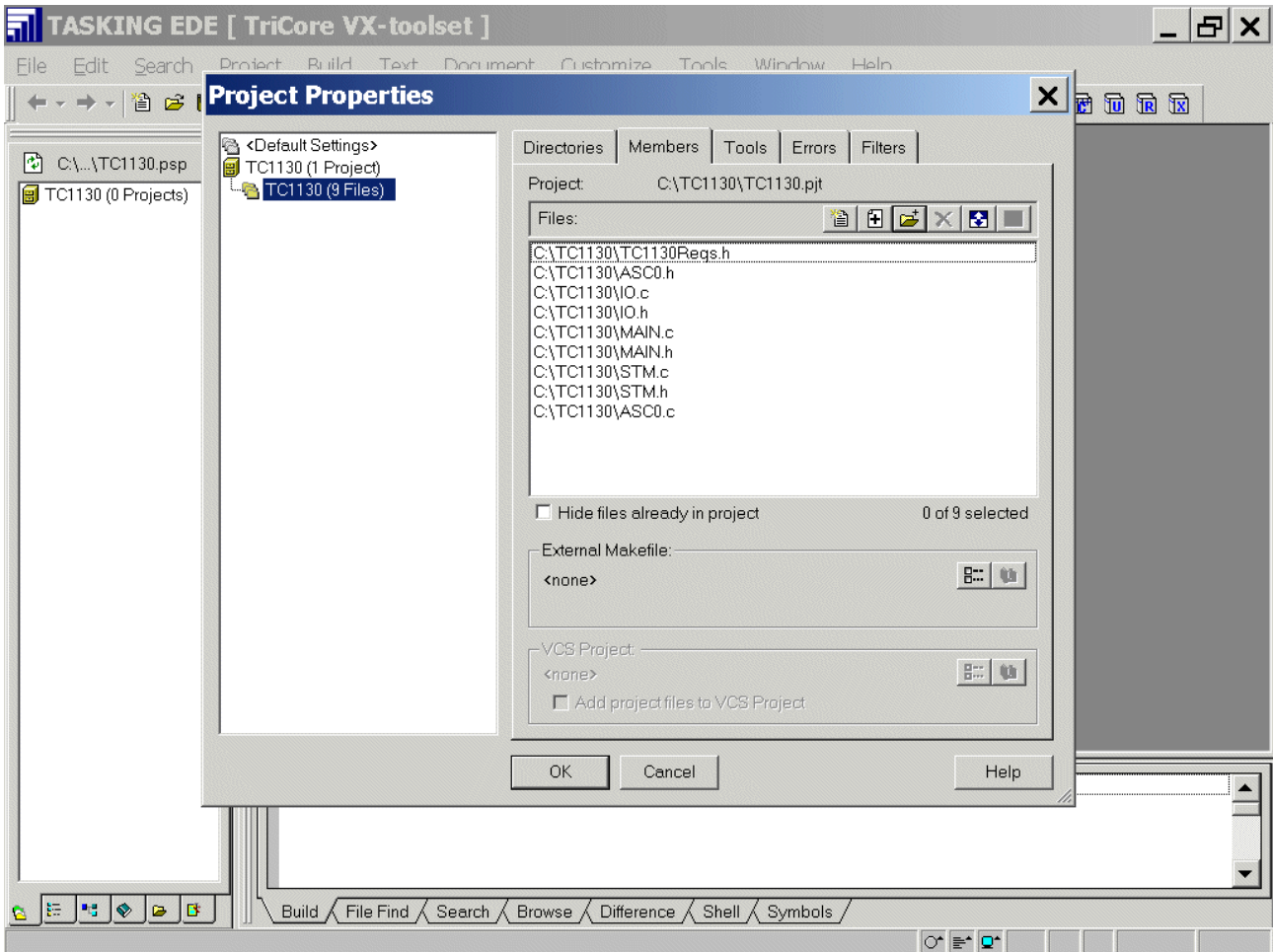
Click: "Add existing files to project"



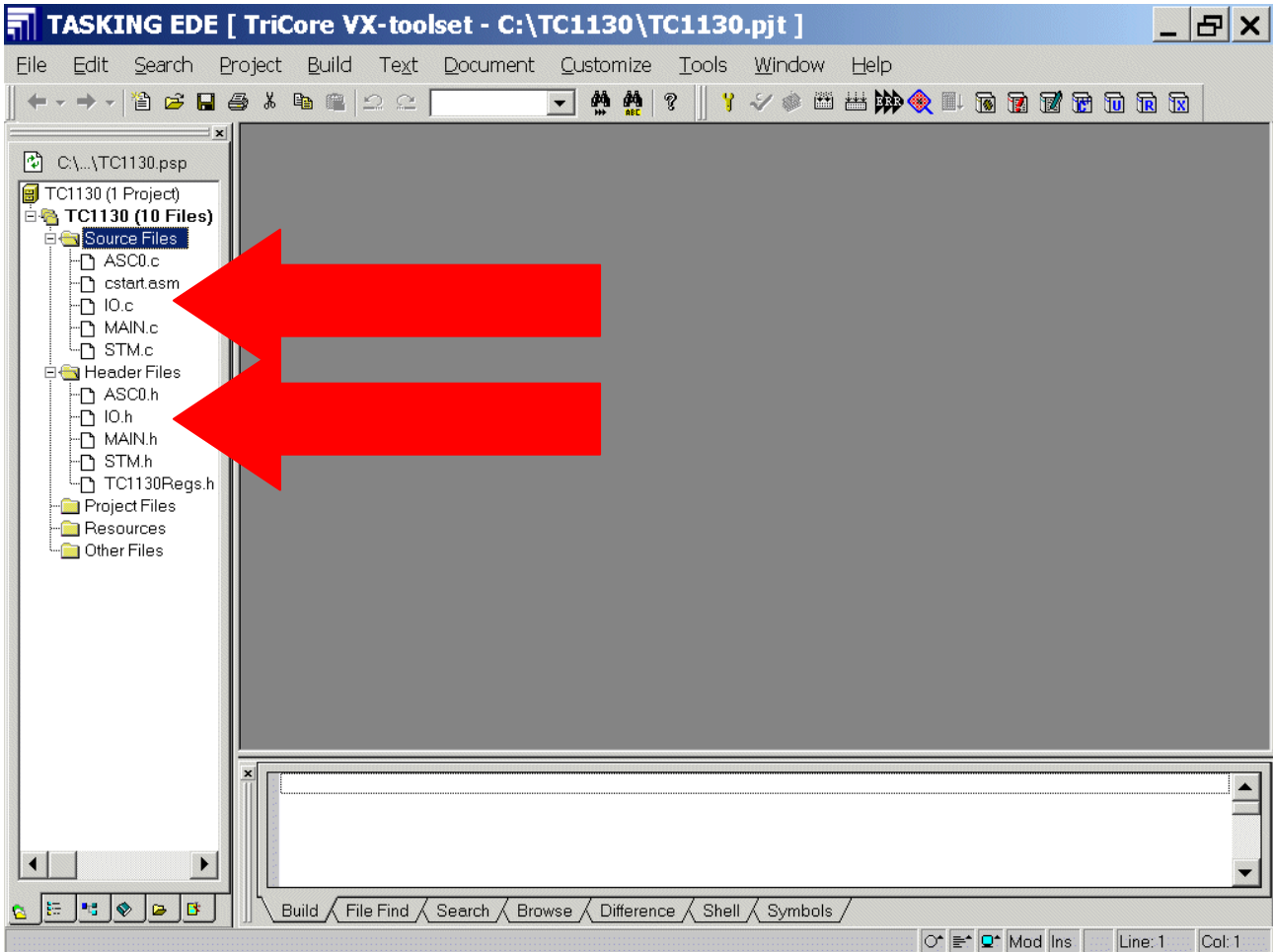
- Select ASC0.c
- Select ASC0.h
- Select IO.c
- Select IO.h
- Select MAIN.c
- Select MAIN.h
- Select STM.c
- Select STM.h
- Select TC1130Regs.h



Open



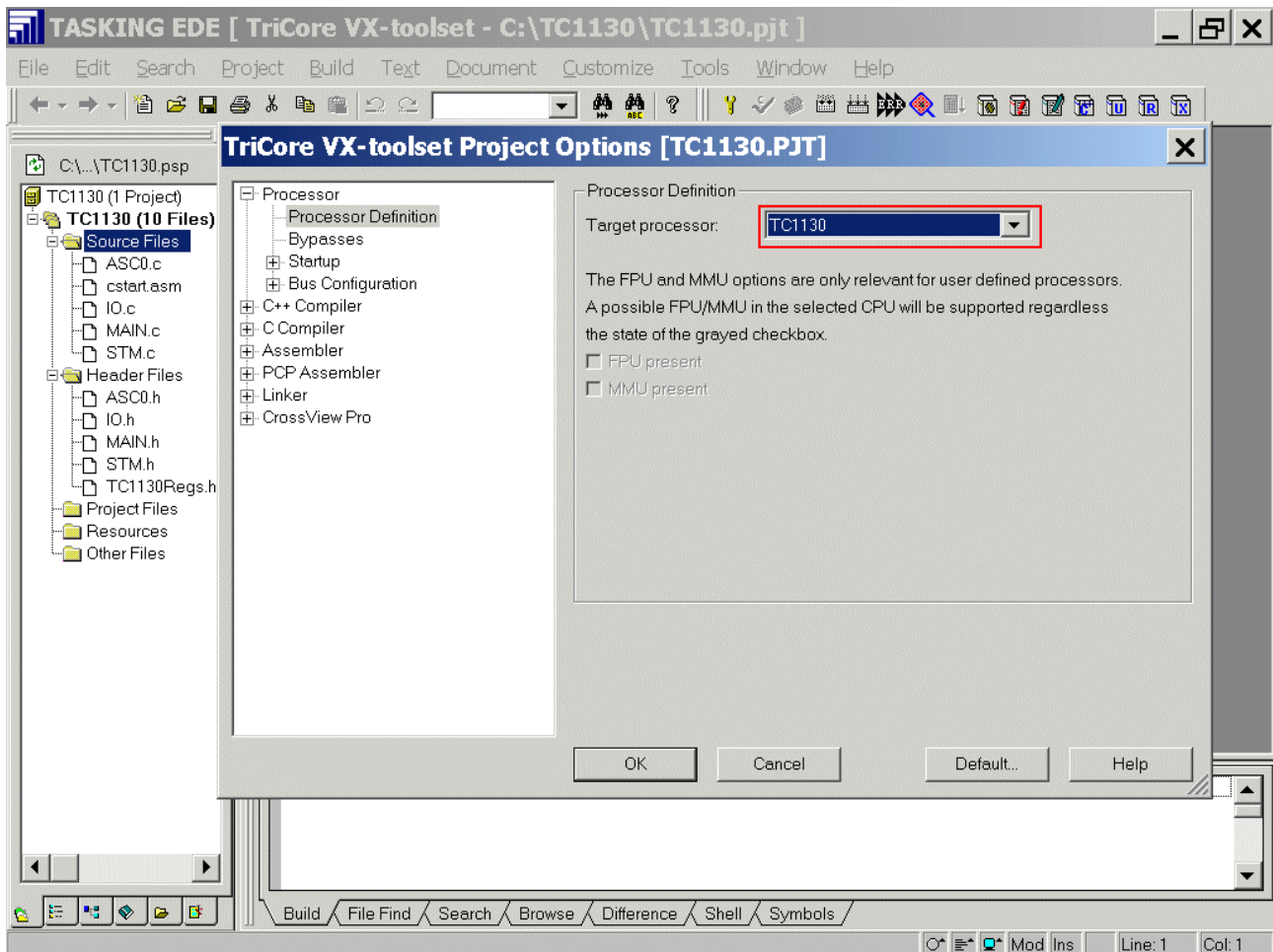
OK



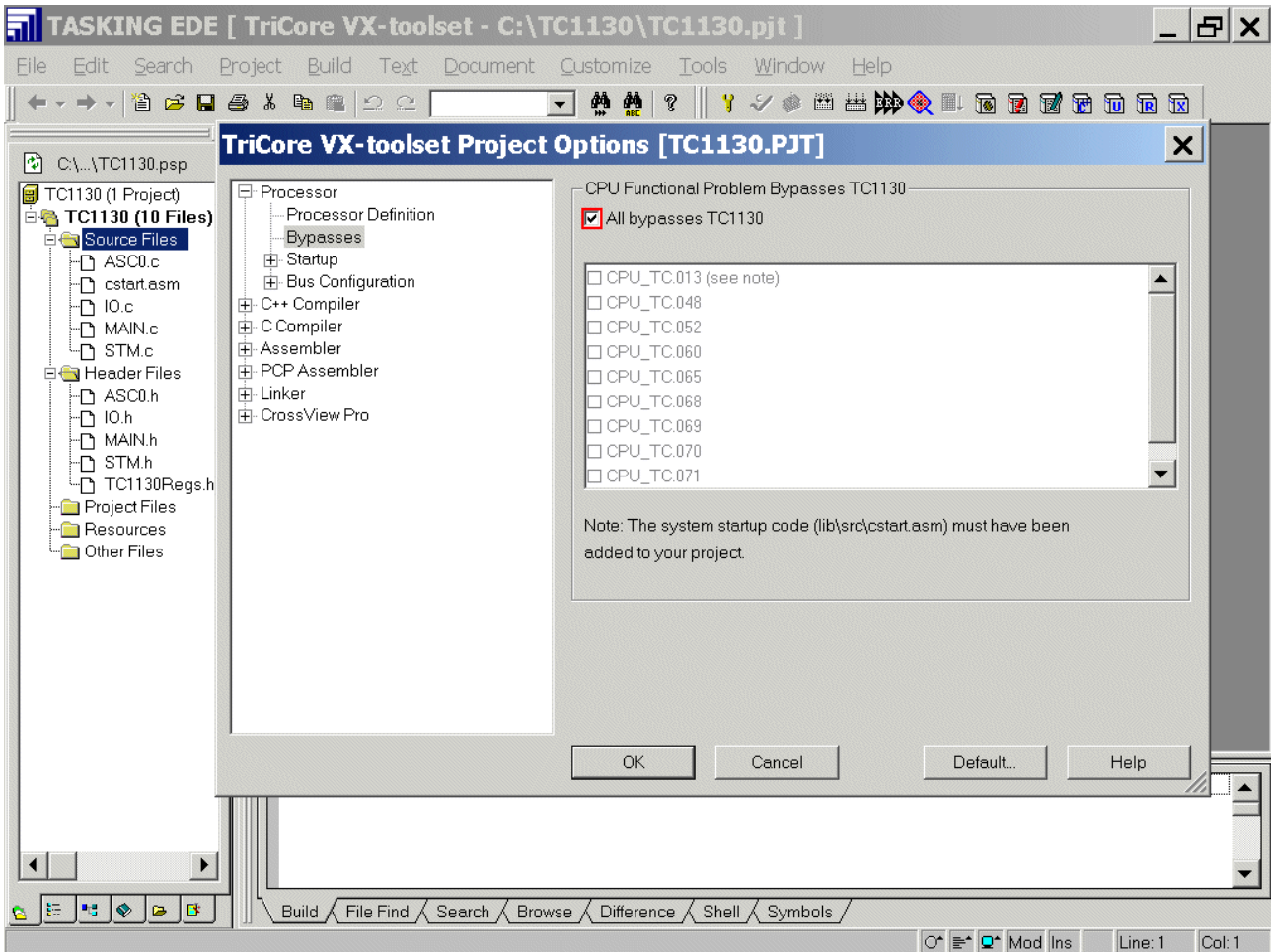
Configure Compiler, Assembler, Linker, Locater and Build – Control:

Project – Project Options

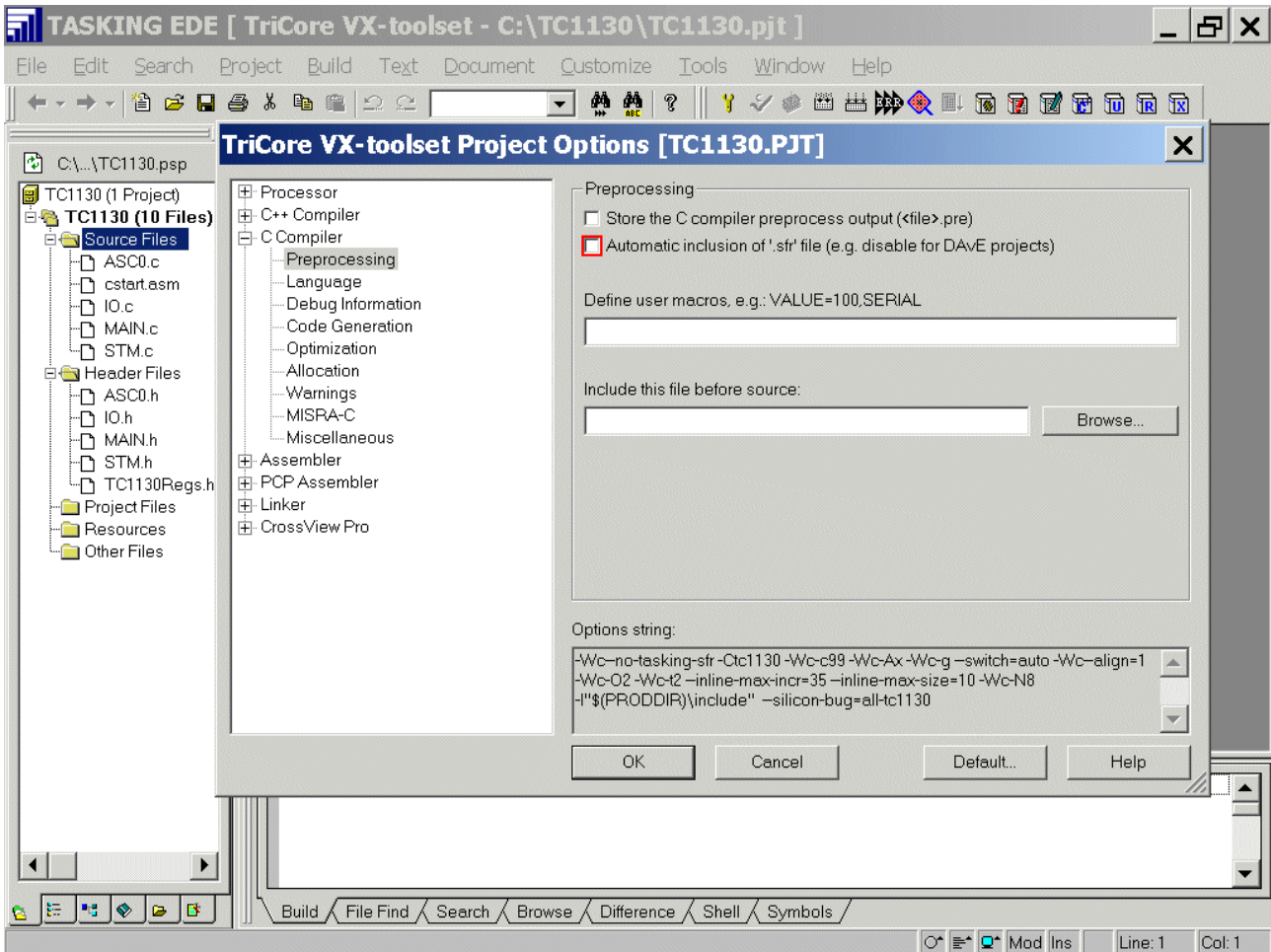
Processor: Processor Definition: Target processor: select TC1130



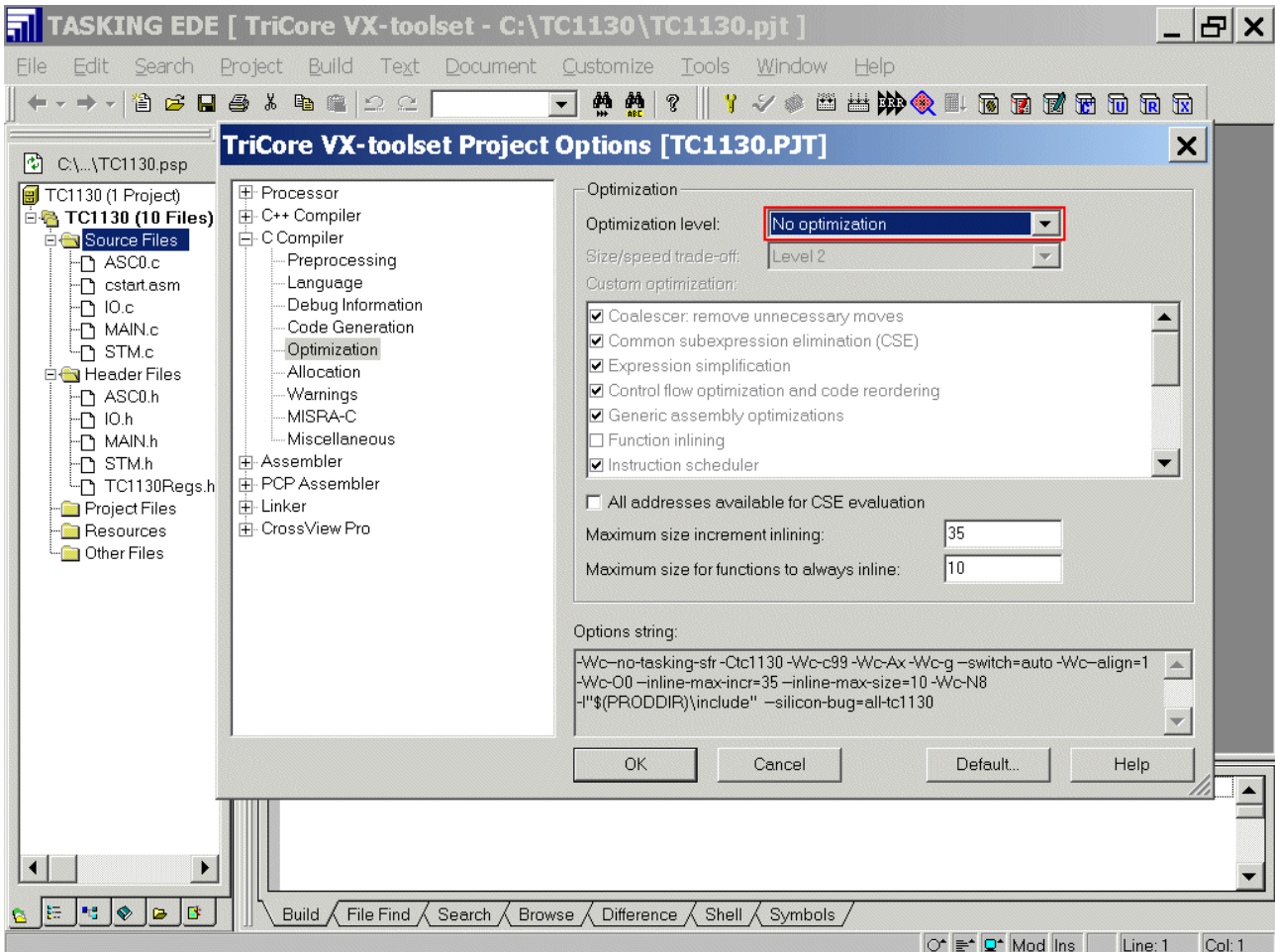
Processor: Bypasses: CPU Functional Problem Bypasses: **click** ✓ All bypasses TC1130



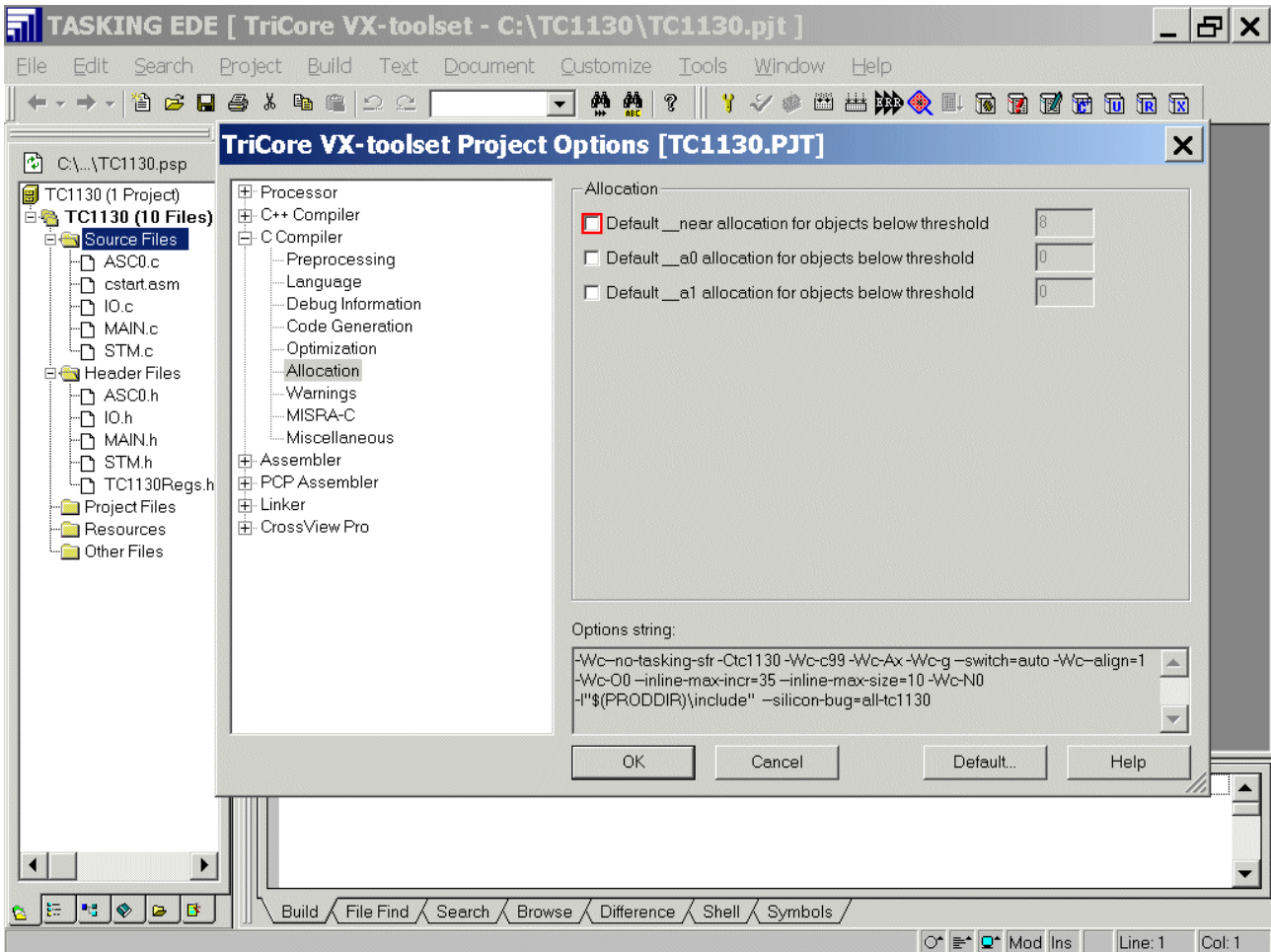
C Compiler: Preprocessing: [click to deactivate](#) Automatic inclusion of .sfr file



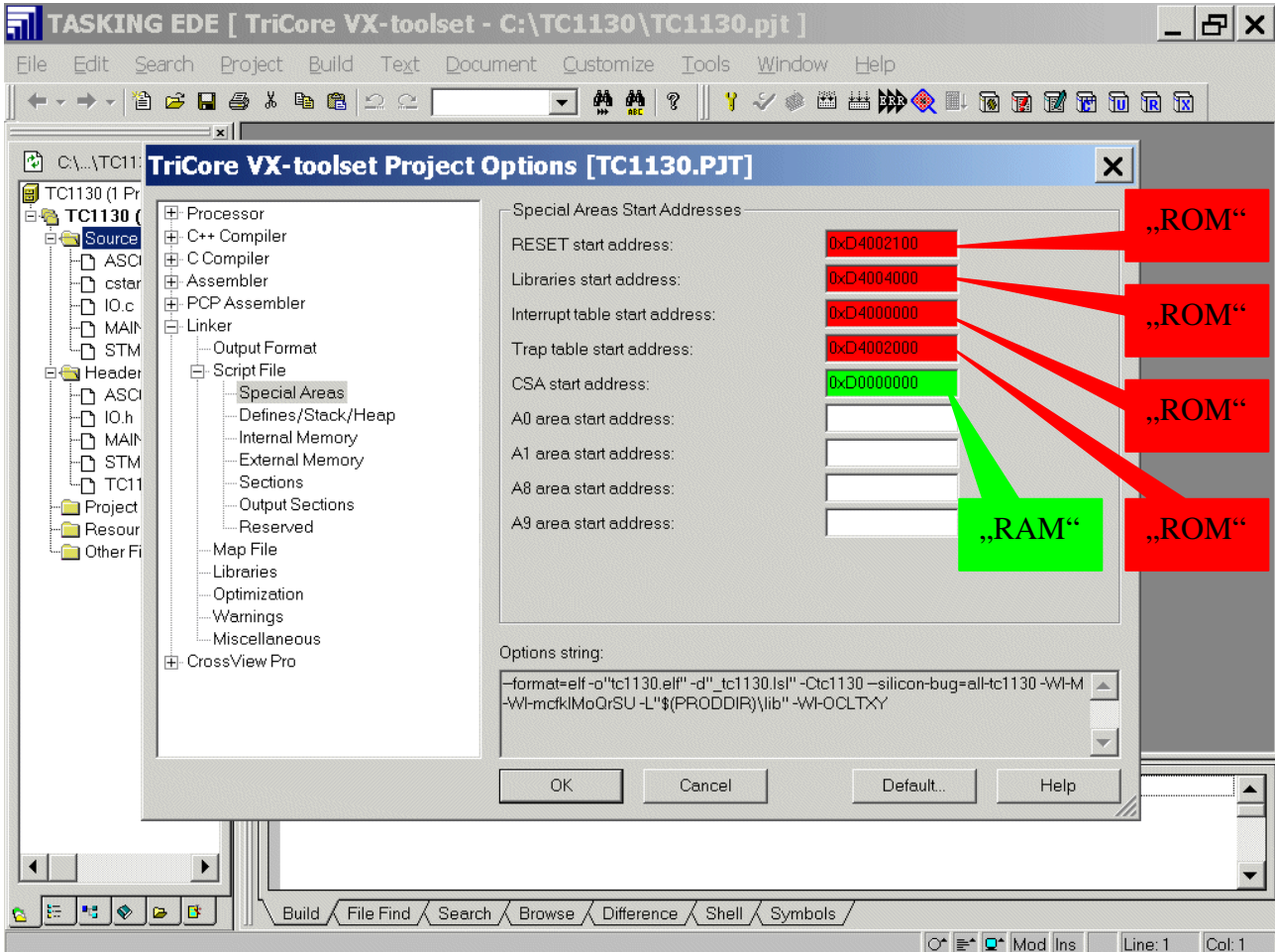
C Compiler: Optimization: Optimization level: select No optimization



C Compiler: Allocation: **click to deactivate** Default __near allocation for objects below threshold



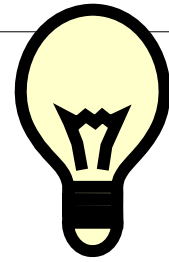
- Linker: Script File: Special Areas: RESET start address: insert 0xD4002100 (PMI_SPRAM)
- Linker: Script File: Special Areas: Libraries start address: insert 0xD4004000 (PMI_SPRAM)
- Linker: Script File: Special Areas: Interrupt table start address: insert 0xD4000000 (PMI_SPRAM)
- Linker: Script File: Special Areas: Trap table start address: insert 0xD4002000 (PMI_SPRAM)
- Linker: Script File: Special Areas: CSA start address: insert/check 0xD0000000 (DMI_LDRAM)



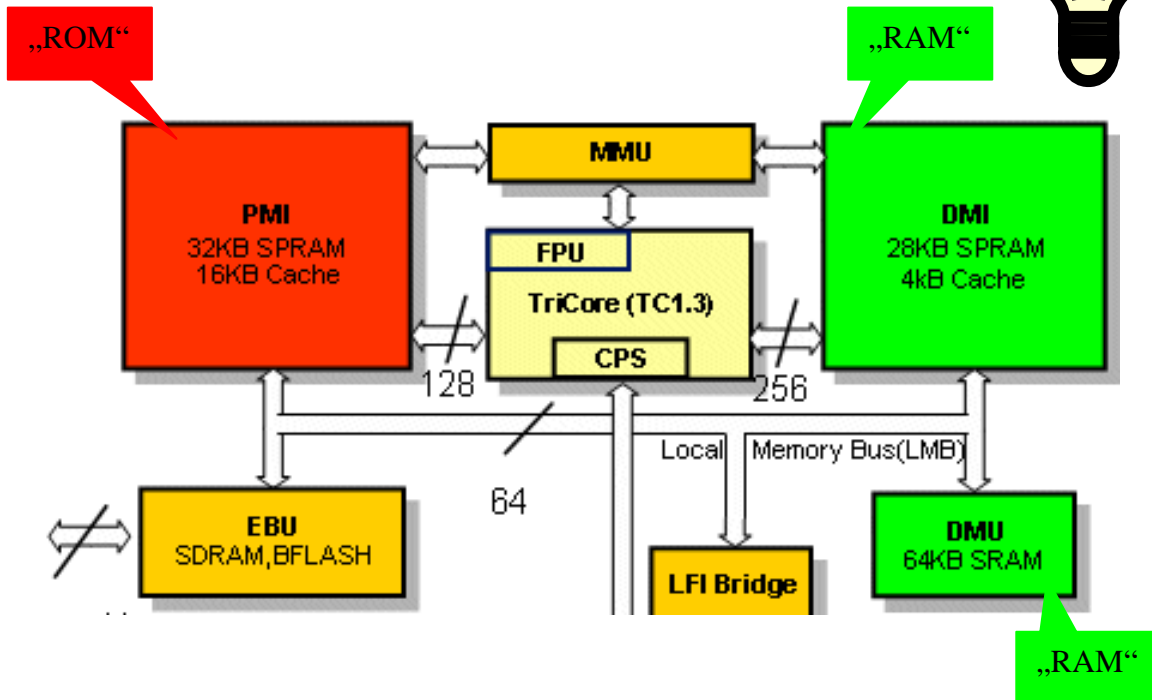
Note:

DMI Local Data RAM = DMI_LDRAM = "DMI_SPRAM"





Additional Information:



Adobe Reader - [tc1130_um_v1.3_2004_11_sys.pdf]

File Edit View Document Tools Window Help

TC1130 (Vol. 1 of 2)
System Units

Memory Map of On-Chip Local Memories

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

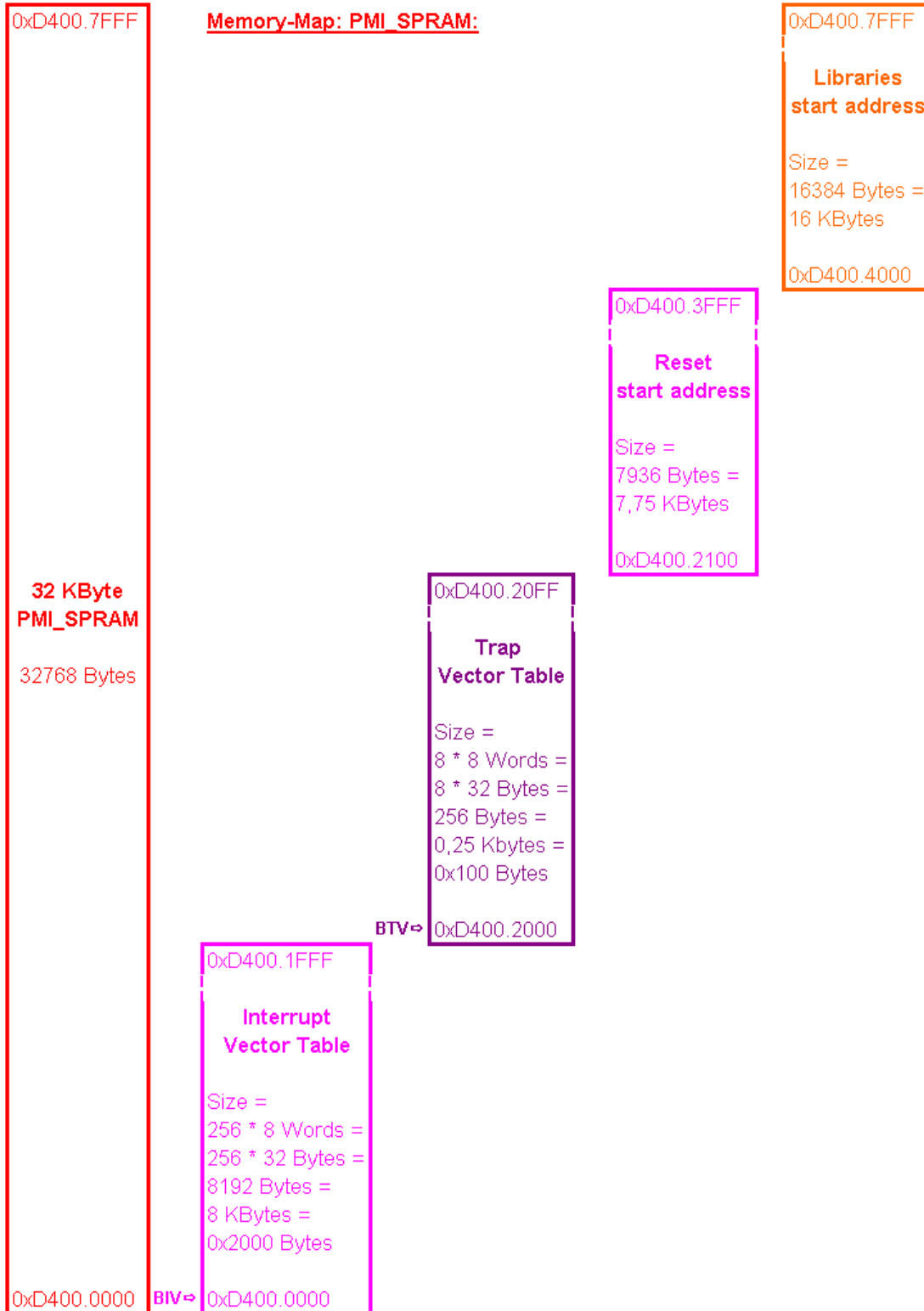
Seg ment	Address Range	Size	Description	DMI Acc.	PMI Acc.	
12	C000 0000 _H - C000 FFFF _H	64 KB	DMU	via LMB	via LMB	o a c h e d
	C001 0000 _H - CFFF FFFF _H	≈ 256 MB	Reserved			
13	D000 0000 _H - D000 6FFF _H	28 KB	DMI Local Data RAM (LDRAM)	DMI local	via LMB	n o n
	D000 7000 _H - D3FF FFFF _H	≈ 64 MB	Reserved			- c a c h e
	D400 0000 _H - D400 7FFF _H	32 KB	PMI Local Code Scratch Pad RAM (SPRAM)	via LMB	PMI local	
	D400 8000 _H - D7FF FFFF _H	≈ 64 MB	Reserved			

8,15 x 11,58 in

243 of 979



Additional Information:





Additional Information:

Interrupt Vector Table:

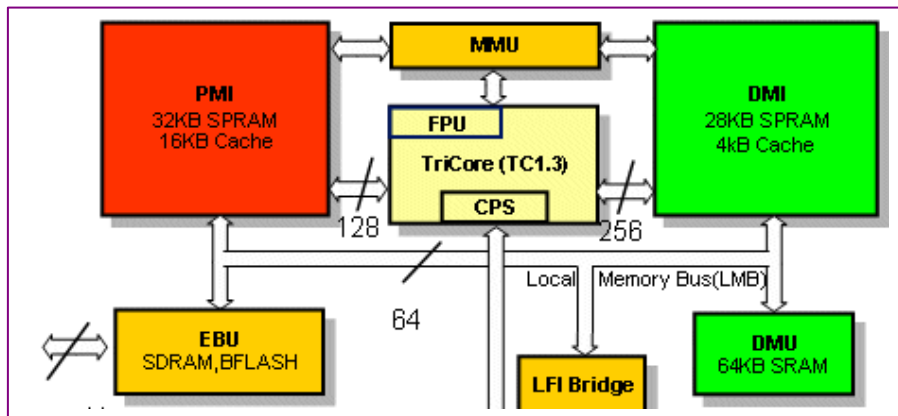
For the full range of 256 interrupt entries an alignment to an 8 KBytes boundary is required.

$$\text{BTV} = 0xD400.0000 = 3.556.769.792 / 8.192 = 434.176 \quad \checkmark$$

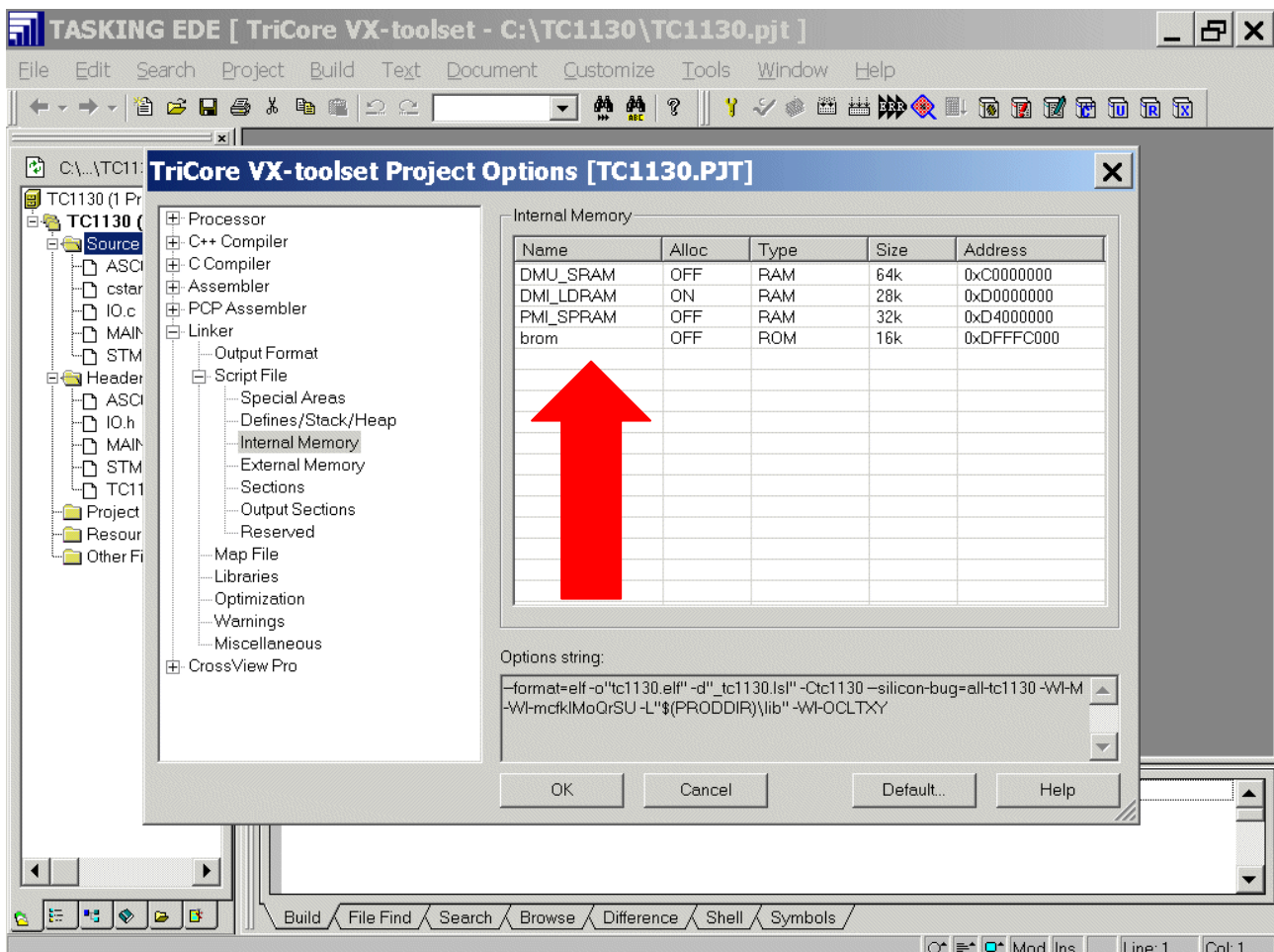
Trap Vector Table:

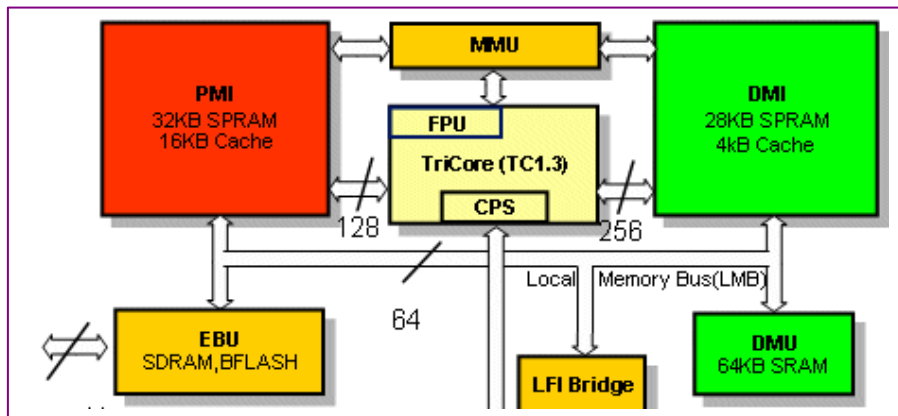
There are eight different trap classes, resulting in Trap Classes from 0 to 7. The contents of BTV should therefore be at least set to a 256 Byte boundary (8 Trap Classes * 8 word spacing).

$$\text{BIV} = 0xD400.2000 = 3.556.777.984 / 256 = 13.893.664 \quad \checkmark$$

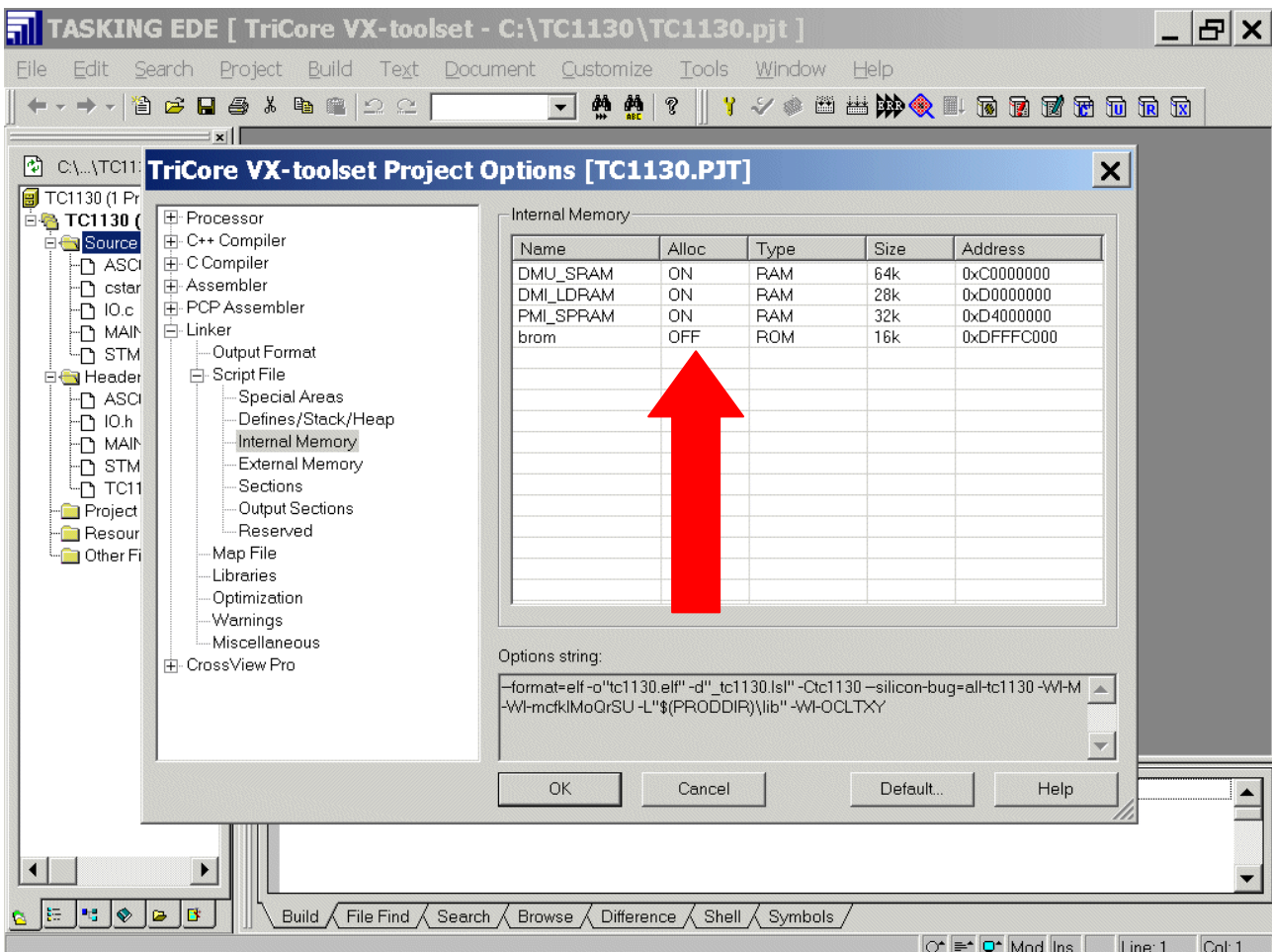


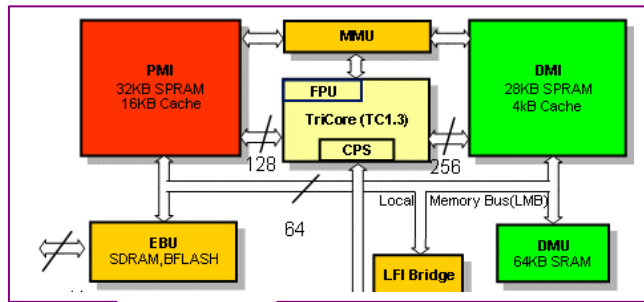
Linker: Script File: Internal Memory: change from lmbrom to DMU_SRAM
 Linker: Script File: Internal Memory: change from ldram to DMI_LDRAM
 Linker: Script File: Internal Memory: change from spram to PMI_SPRAM





Linker: Script File: Internal Memory: DMU_SRAM: Alloc: **select ON!!!**
 Linker: Script File: Internal Memory: PMI_SDRAM: Alloc: **select ON!!!**





Linker: Script File: Internal Memory: PMI_SPRAM: Type: select ROM



The screenshot shows the TASKING EDE IDE interface. The main window displays the 'TriCore VX-toolset Project Options [TC1130.PJT]' dialog box. The 'Internal Memory' section contains the following table:

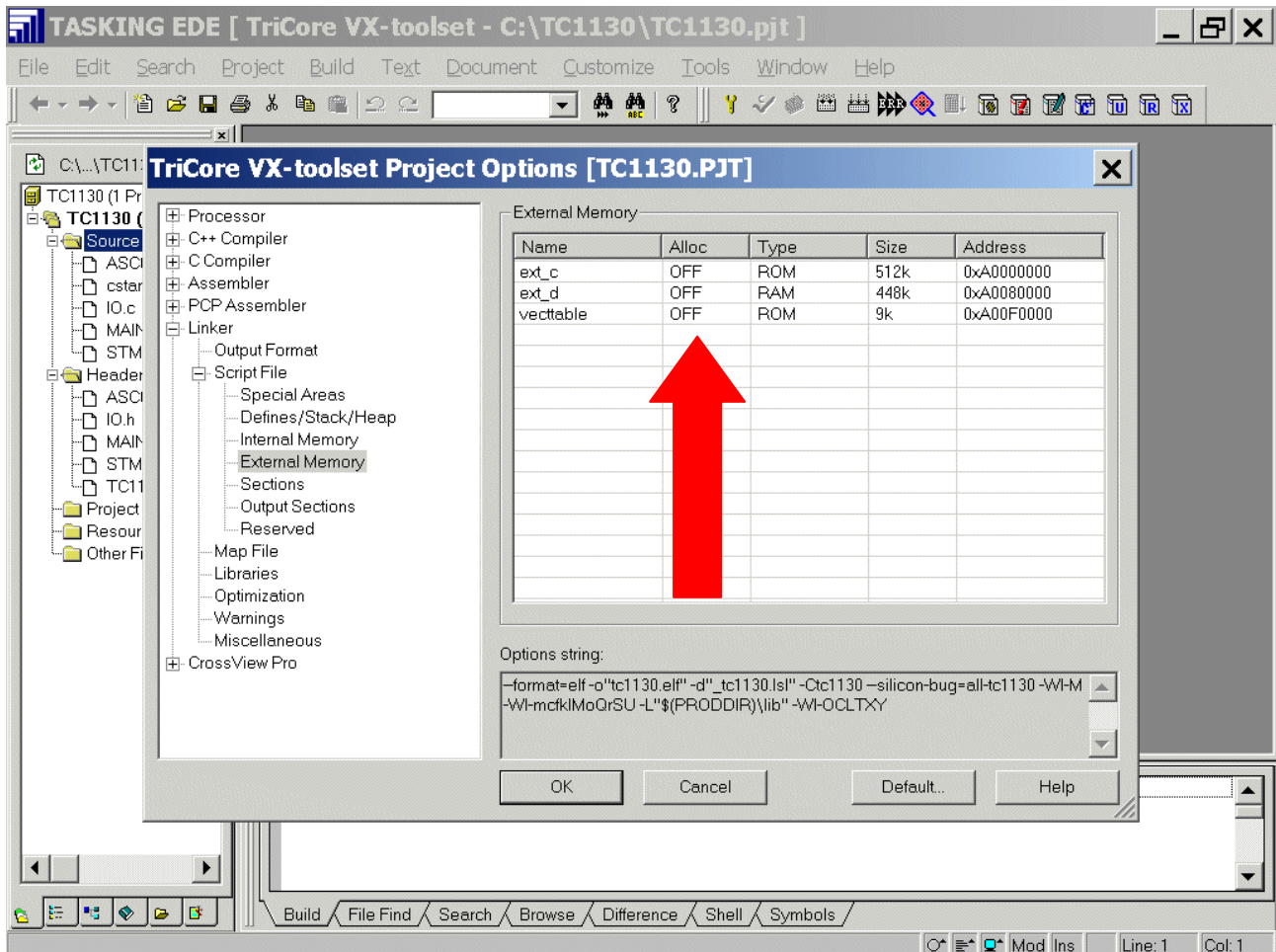
Name	Alloc	Type	Size	Address
DMU_SPRAM	ON	RAM	64k	0xC0000000
DMI_LDRAM	ON	RAM	28k	0xD0000000
PMI_SPRAM	ON	ROM	32k	0xD4000000
brom	OFF	ROM	16k	0xDFFFC000

The 'Script File' option in the left-hand tree is selected, and the 'PMI_SPRAM' row in the table is highlighted. Red arrows point to these elements. The 'Options string' field at the bottom contains linker options for format, output directory, and memory layout.

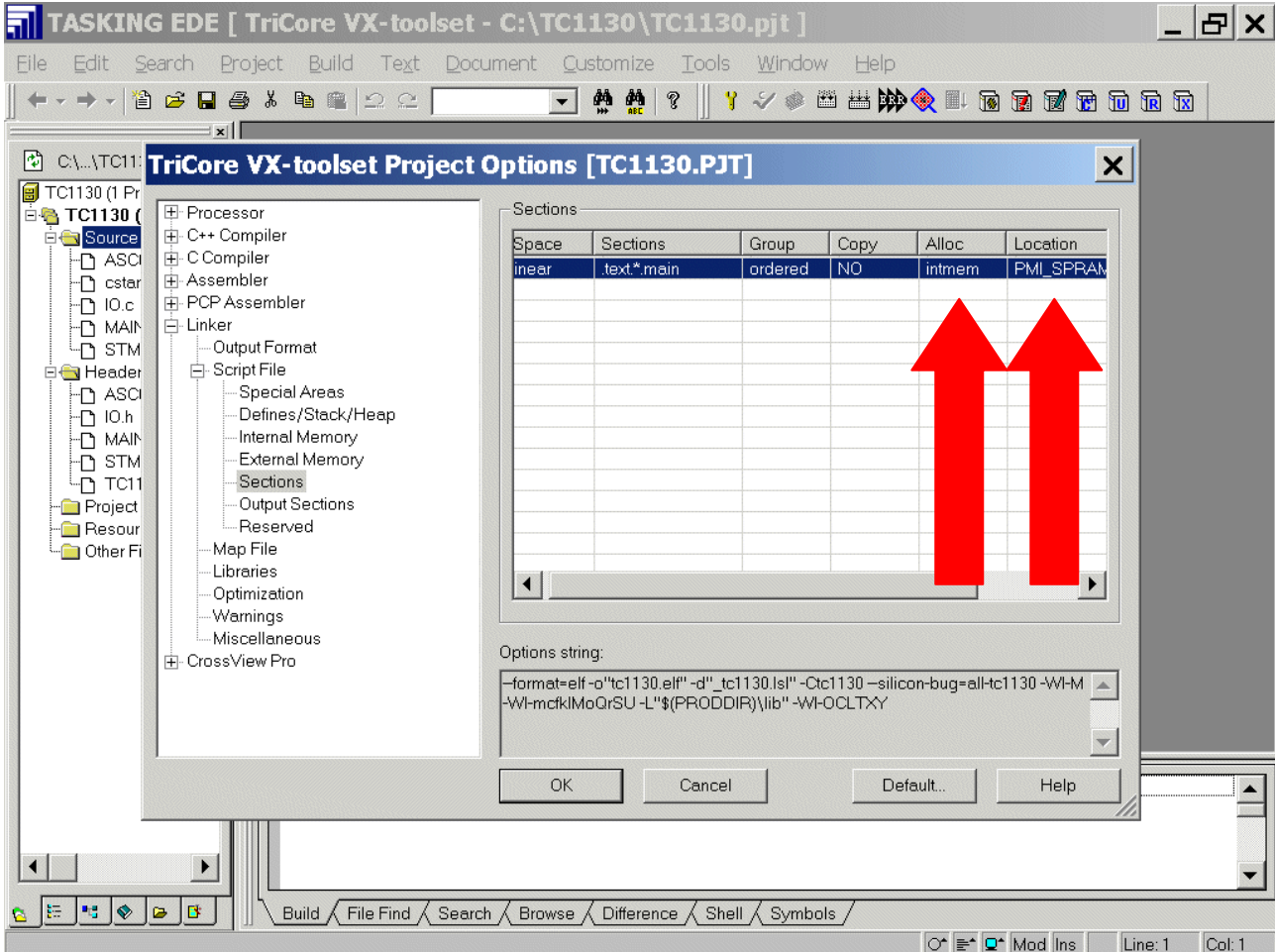


(Configuration of this dialog-window is now finished)

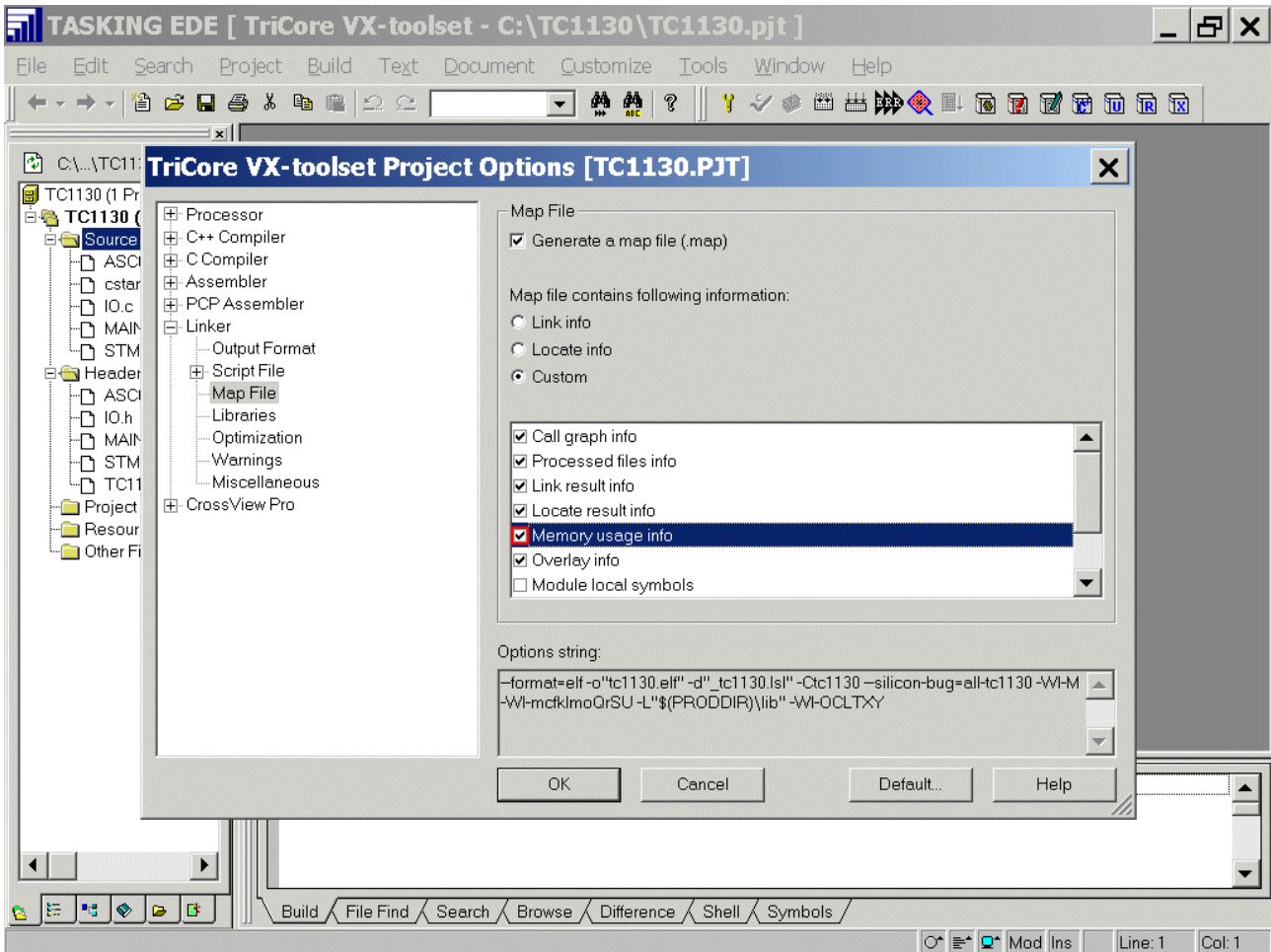
Linker: Script File: External Memory: ext_c: Alloc: select OFF
 Linker: Script File: External Memory: ext_d: Alloc: select OFF
 Linker: Script File: External Memory: vectable: Alloc: select OFF



Linker: Script File: Sections: linear: Alloc: select intmem
 Linker: Script File: Sections: linear: Location: insert PMI_SPRAM

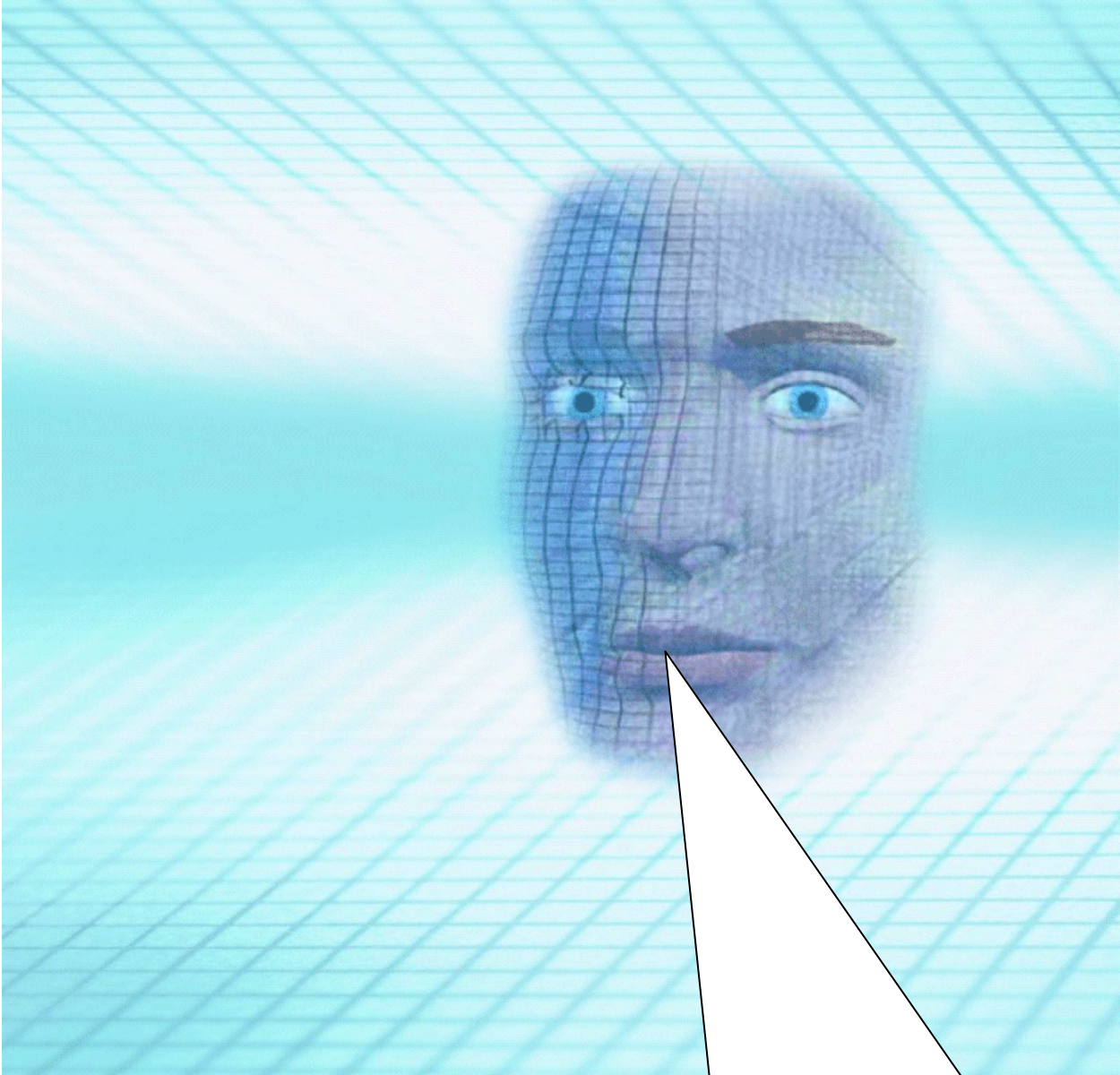


Linker: Map File: click ✓ Memory usage info



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):

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

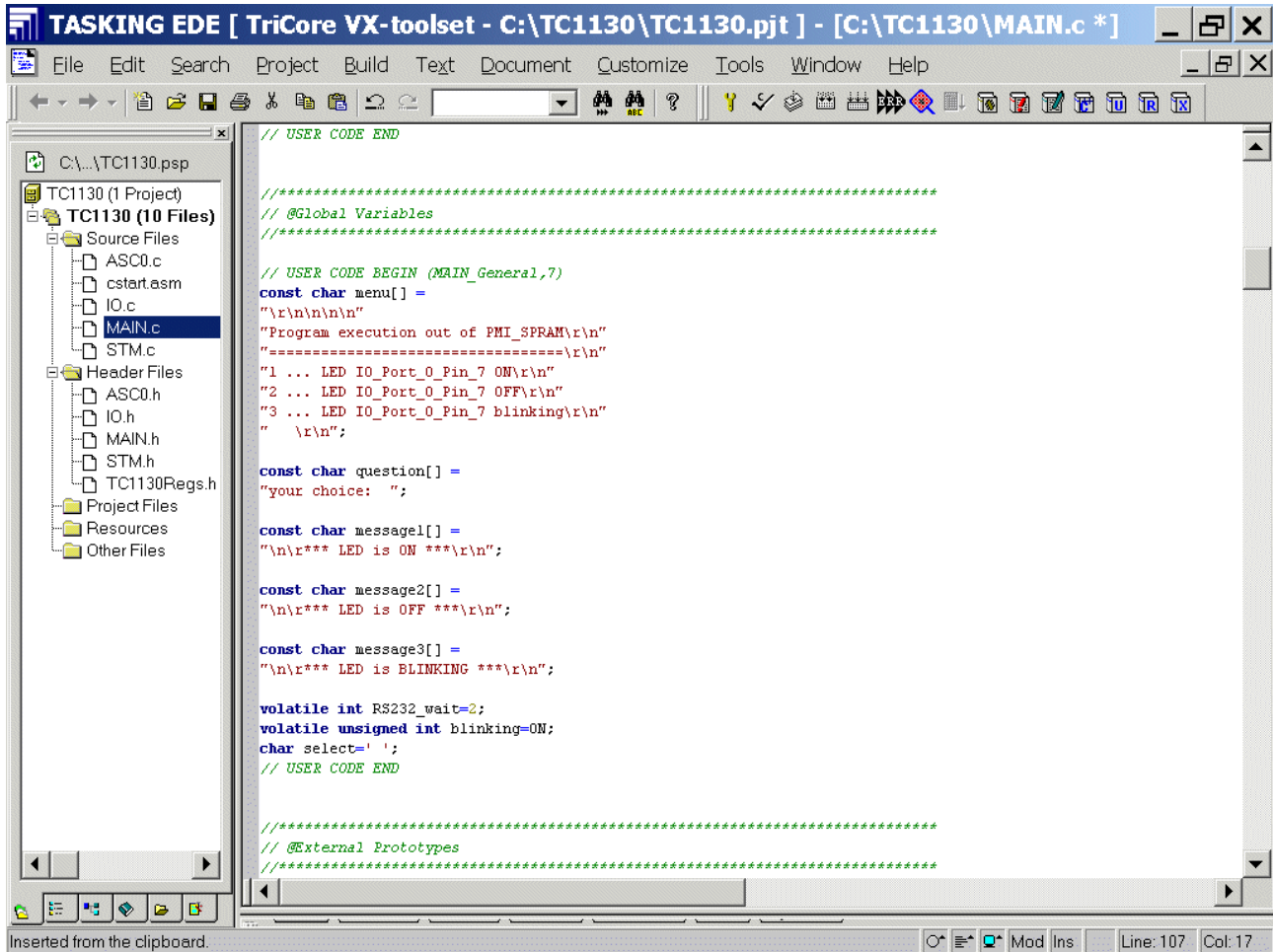
const char question[] =
"your choice: ";

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

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

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

volatile int RS232_wait=2;
volatile unsigned int blinking=ON;
char select=' ';
```



```

TASKING EDE [ TriCore VX-toolset - C:\TC1130\TC1130.pjt ] - [C:\TC1130\MAIN.c *]
File Edit Search Project Build Text Document Customize Tools Window Help
C:\...TC1130.psp
TC1130 (1 Project)
  TC1130 (10 Files)
    Source Files
      ASC0.c
      cstart.asm
      IO.c
      MAIN.c
      STM.c
    Header Files
      ASC0.h
      IO.h
      MAIN.h
      STM.h
      TC1130Regs.h
    Project Files
    Resources
    Other Files

// USER CODE END

//*****
// @Global Variables
//*****

// USER CODE BEGIN (MAIN_General,7)
const char menu[] =
"\r\n\r\n\r\n"
"Program execution out of PHI_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: ";

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

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

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

volatile int RS232_wait=2;
volatile unsigned int blinking=0N;
char select=' ';
// USER CODE END

//*****
// @External Prototypes
//*****

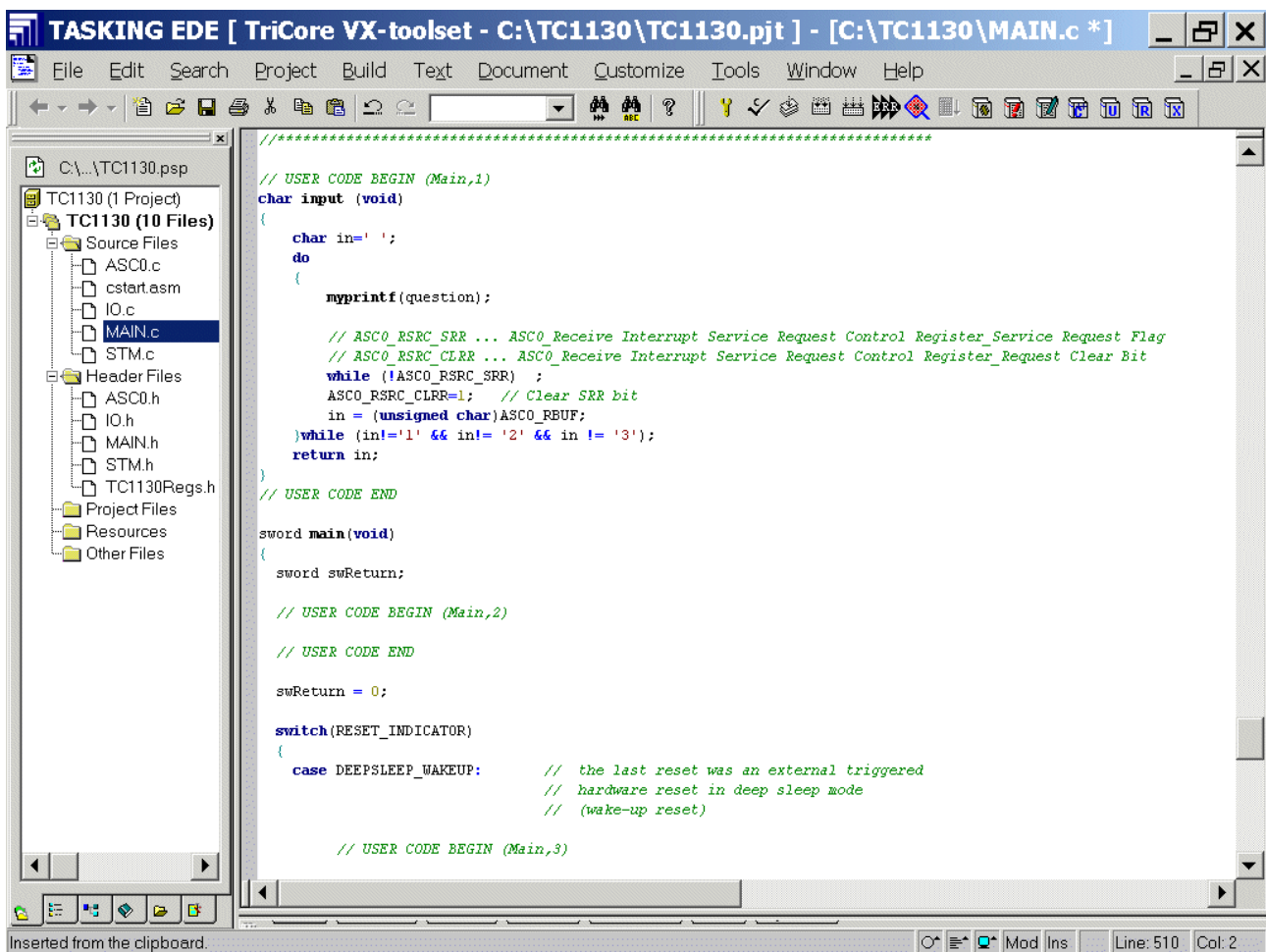
```

Inserted from the clipboard. Mod Ins Line: 107 Col: 17

Double click: **Main.c** insert User Code [function: input()]:

```
char input (void)
{
    char in=' ';
    do
    {
        myprintf(question);

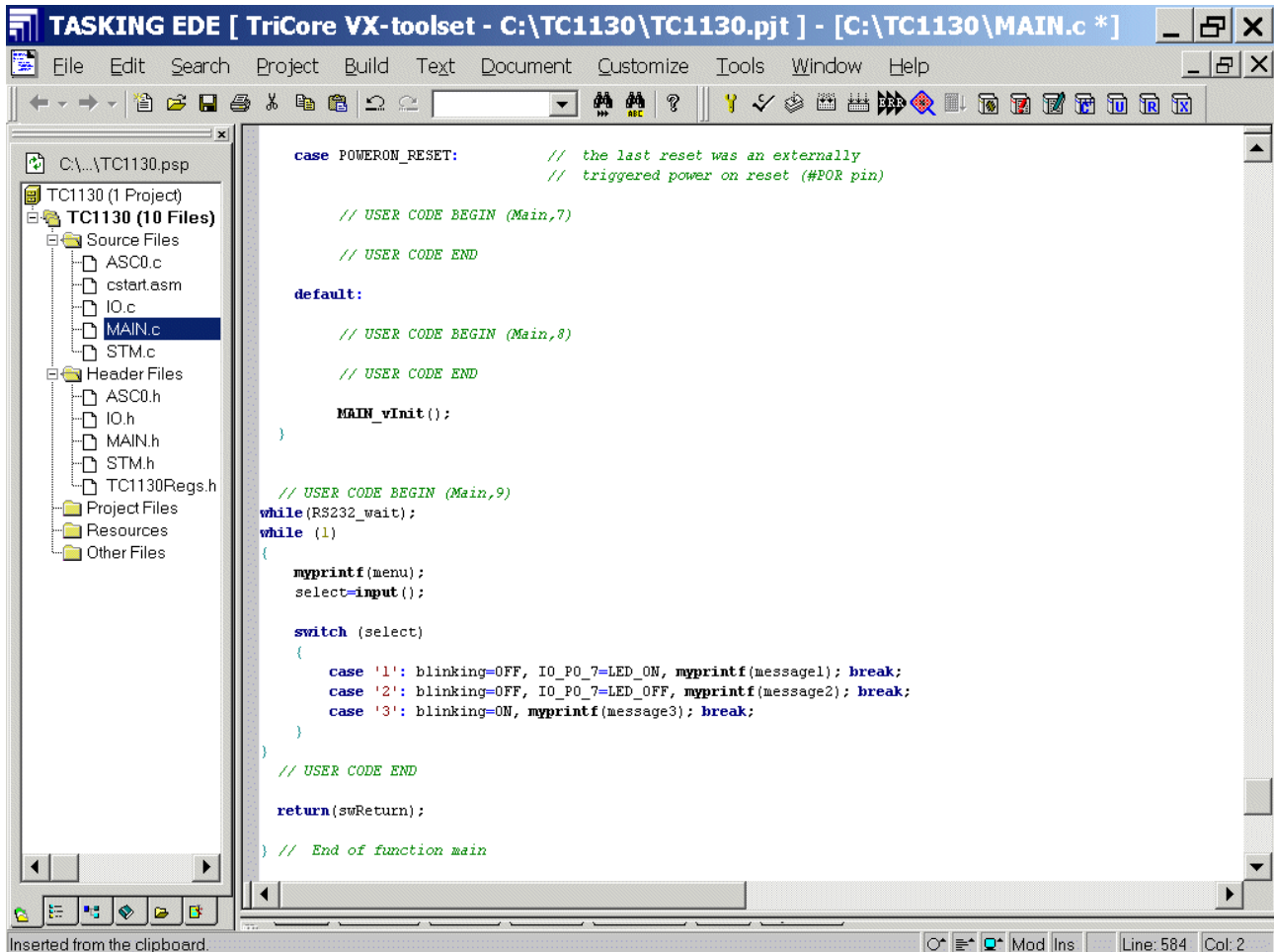
        // ASC0_RSRC_SRR ... ASC0_Receive Interrupt Service Request Control Register_Service Request Flag
        // ASC0_RSRC_CLRR ... ASC0_Receive Interrupt Service Request Control Register_Request Clear Bit
        while (!ASC0_RSRC_SRR) ;
        ASC0_RSRC_CLRR=1;    // Clear SRR bit
        in = (unsigned char)ASC0_RBUF;
    }while (in!='1' && in!= '2' && in != '3');
    return in;
}
```



Double click: [Main.c](#) insert Code:

```
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;
    }
}
```



The screenshot shows the TASKING EDE IDE interface. The title bar reads "TASKING EDE [TriCore VX-toolset - C:\TC1130\TC1130.pjt] - [C:\TC1130\MAIN.c *]". The menu bar includes File, Edit, Search, Project, Build, Text, Document, Customize, Tools, Window, and Help. The toolbar contains various icons for file operations and development. The left sidebar shows a project tree for "TC1130 (1 Project)" with subfolders for "Source Files" and "Header Files". The "Source Files" folder is expanded, showing files like ASC0.c, cstart.asm, IO.c, MAIN.c (selected), STM.c, and TC1130Regs.h. The main editor window displays the source code for MAIN.c, which includes comments for user code and a main function with a menu loop. The status bar at the bottom indicates "Inserted from the clipboard." and "Line: 584 Col: 2".

```
case POWERON_RESET: // the last reset was an externally
                    // triggered power on reset (#POR pin)

// USER CODE BEGIN (Main,7)

// USER CODE END

default:

// USER CODE BEGIN (Main,8)

// USER CODE END

MAIN_vinit();
}

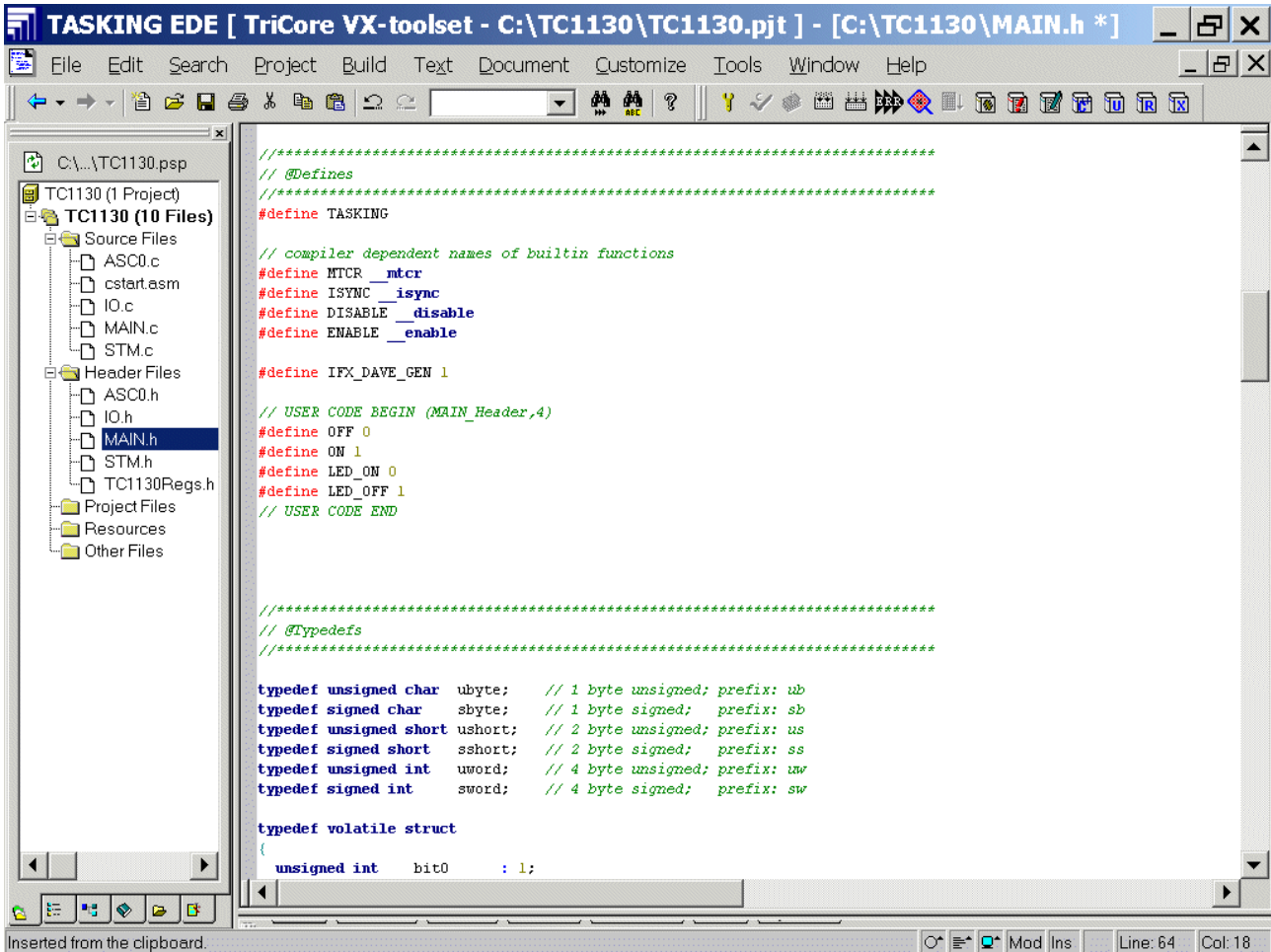
// USER CODE BEGIN (Main,9)
while(RS232_wait);
while (1)
{
    myprintf(menu);
    select=input();

    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;
    }
}
// USER CODE END

return(swReturn);
} // End of function main
```

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

```
#define OFF 0
#define ON 1
#define LED_ON 0
#define LED_OFF 1
```



The screenshot shows the TASKING EDE IDE interface. The left pane displays the project structure for TC1130, with the 'MAIN.h' file selected under 'Header Files'. The main editor window shows the content of 'MAIN.h', which includes various preprocessor directives and typedefs. The code is as follows:

```

//*****
// @Defines
//*****
#define TASKING

// compiler dependent names of builtin functions
#define MTCR __mtrc
#define ISYNC __isync
#define DISABLE __disable
#define ENABLE __enable

#define IFX_DAVE_GEN 1

// USER CODE BEGIN (MAIN_Header,4)
#define OFF 0
#define ON 1
#define LED_ON 0
#define LED_OFF 1
// USER CODE END

//*****
// @Typedefs
//*****

typedef unsigned char  ubyte;    // 1 byte unsigned; prefix: ub
typedef signed char    sbyte;    // 1 byte signed; prefix: sb
typedef unsigned short ushort;  // 2 byte unsigned; prefix: us
typedef signed short   sshort;   // 2 byte signed; prefix: ss
typedef unsigned int   uword;    // 4 byte unsigned; prefix: uw
typedef signed int     sword;    // 4 byte signed; prefix: sw

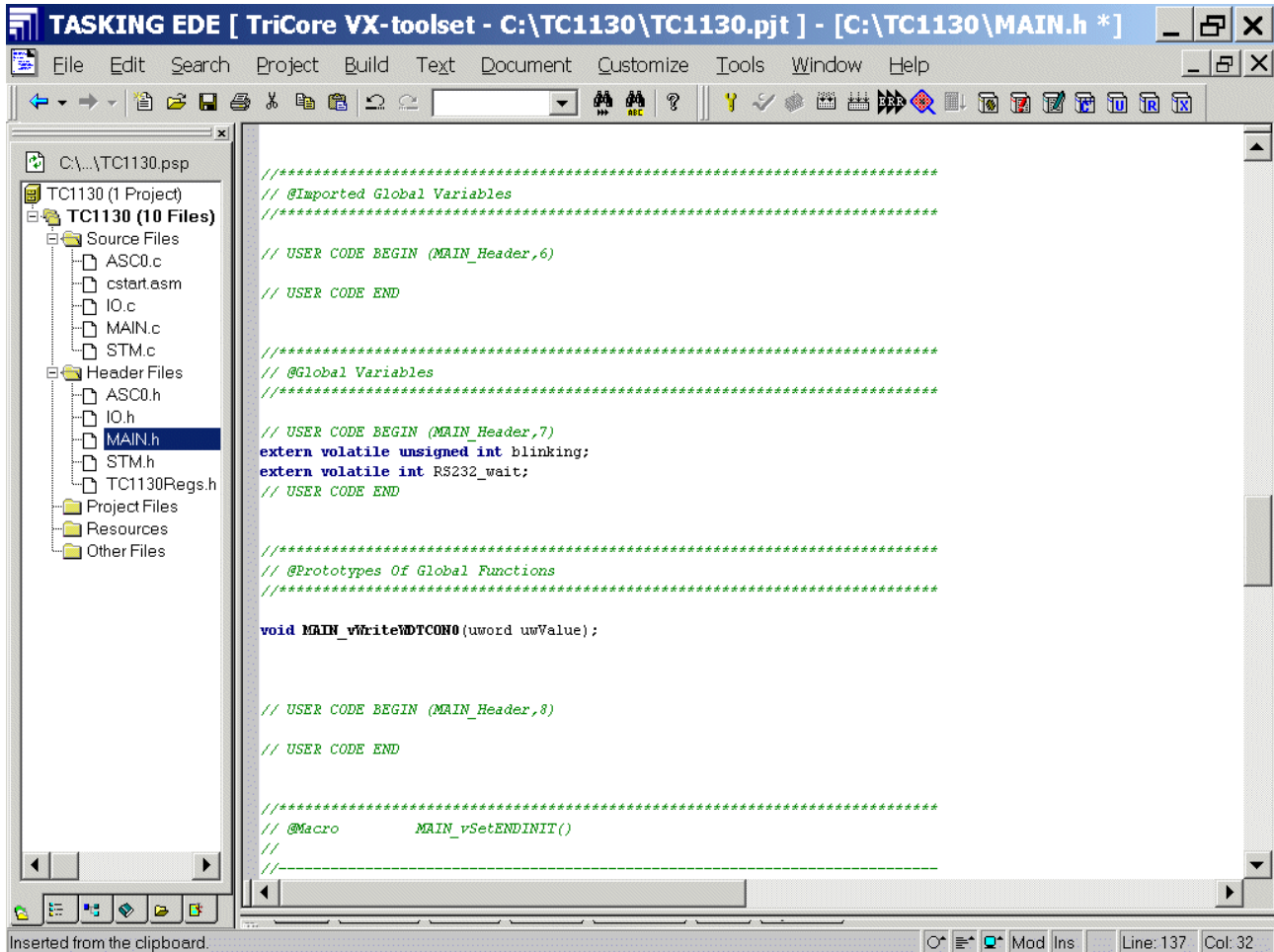
typedef volatile struct
{
    unsigned int  bit0    : 1;

```

The status bar at the bottom indicates "Inserted from the clipboard." and shows the current cursor position as "Line: 64 Col: 18".

Double click: **Main.h** and **insert** Global Variables (Extern Declarations):

```
extern volatile unsigned int blinking;
extern volatile int RS232_wait;
```



The screenshot shows the TASKING EDE IDE interface. The left pane displays a project tree for 'TC1130 (10 Files)', with 'MAIN.h' selected under 'Header Files'. The main editor window shows the content of 'MAIN.h', which includes sections for imported global variables, user code, global variables, prototypes of global functions, and a macro definition. The global variables section contains the following code:

```

// @Global Variables
// *****

// USER CODE BEGIN (MAIN_Header,7)
extern volatile unsigned int blinking;
extern volatile int RS232_wait;
// USER CODE END

// @Prototypes Of Global Functions
// *****

void MAIN_vWriteWDTC0N0(uword uwValue);

// USER CODE BEGIN (MAIN_Header,8)
// USER CODE END

// @Macro      MAIN_vSetENDINIT()
//
// -----

```

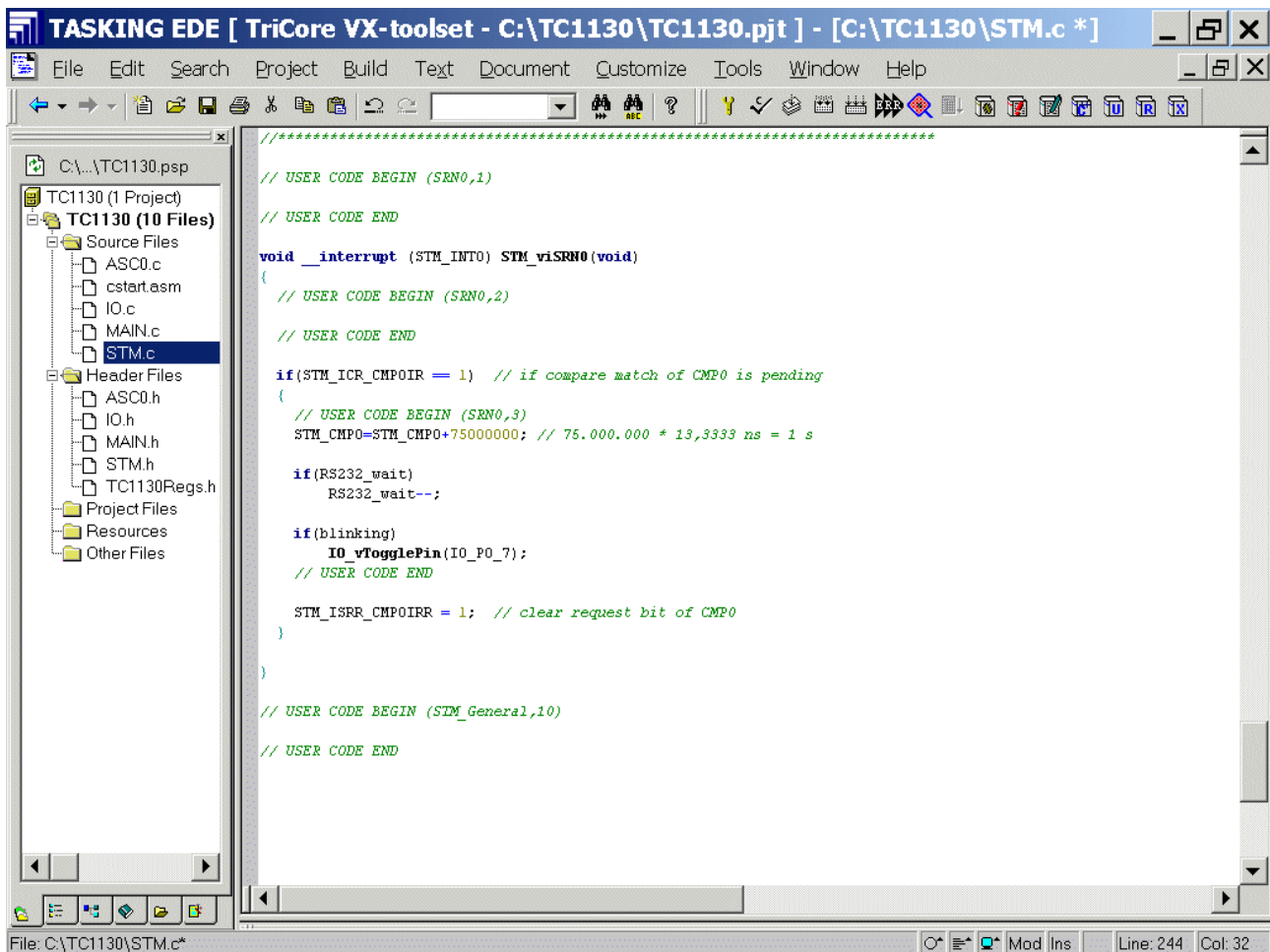
The status bar at the bottom indicates 'Line: 137 Col: 32'.

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);
```



```
*****
// USER CODE BEGIN (SRN0,1)
// USER CODE END

void __interrupt (STM_INT0) STM_viSRN0(void)
{
    // USER CODE BEGIN (SRN0,2)
    // USER CODE END

    if(STM_ICR_CMP0IR = 1) // if compare match of CMP0 is pending
    {
        // USER CODE BEGIN (SRN0,3)
        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);
        // USER CODE END

        STM_ISR_CMP0IRR = 1; // clear request bit of CMP0
    }

    // USER CODE BEGIN (STM_General,10)
    // USER CODE END
}
```

Note:

$75.000.000 * 13,3333 \text{ ns} = 1 \text{ s}$

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: 0b100011110000110100011000000 = 209217728 => 2,789 sec
3. Interrupt : 0b10100011110000110100011000000 = 343435456 => 4,579 sec





Additional Information:

August 2003

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.

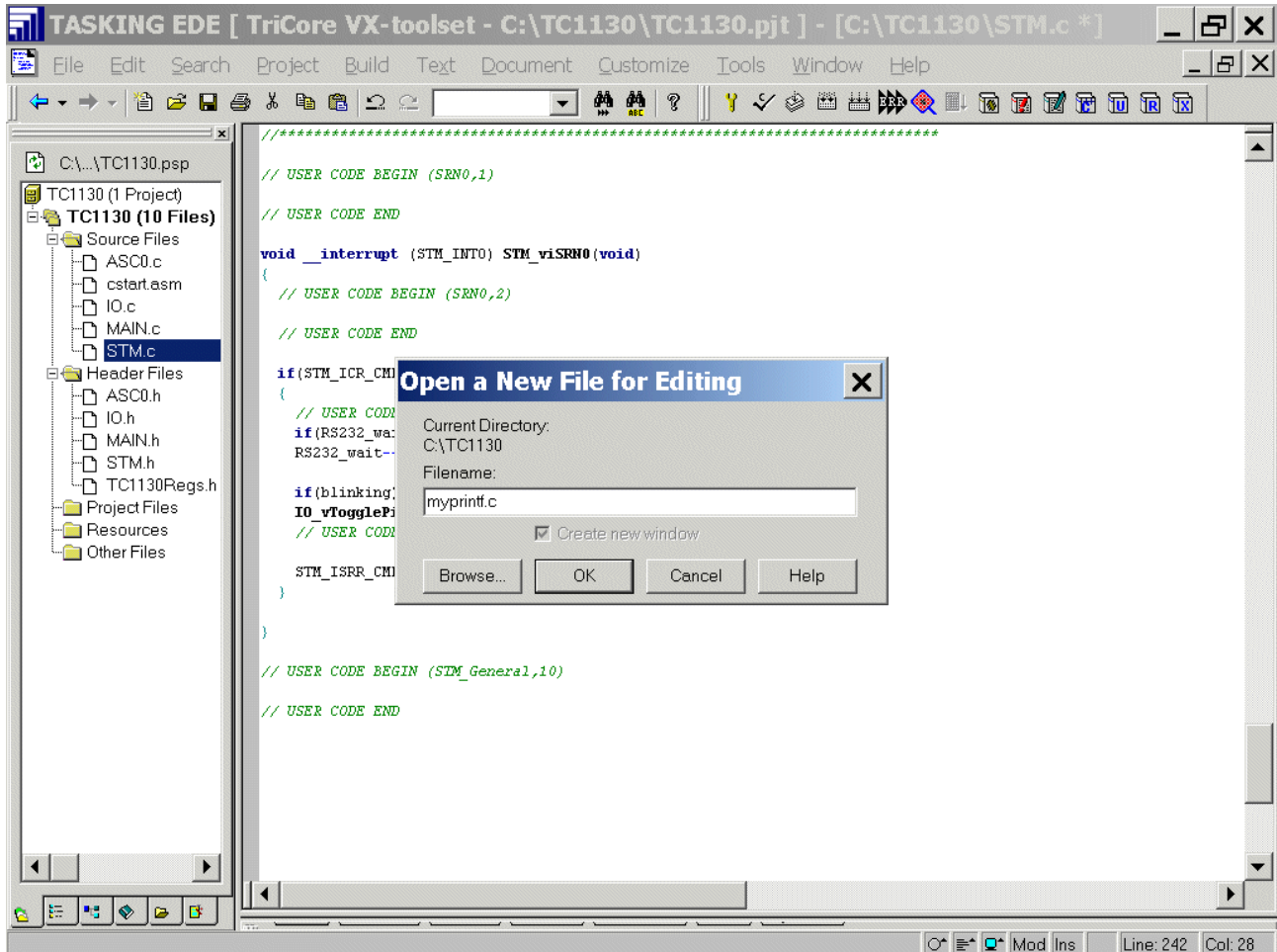
EXAMPLE

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



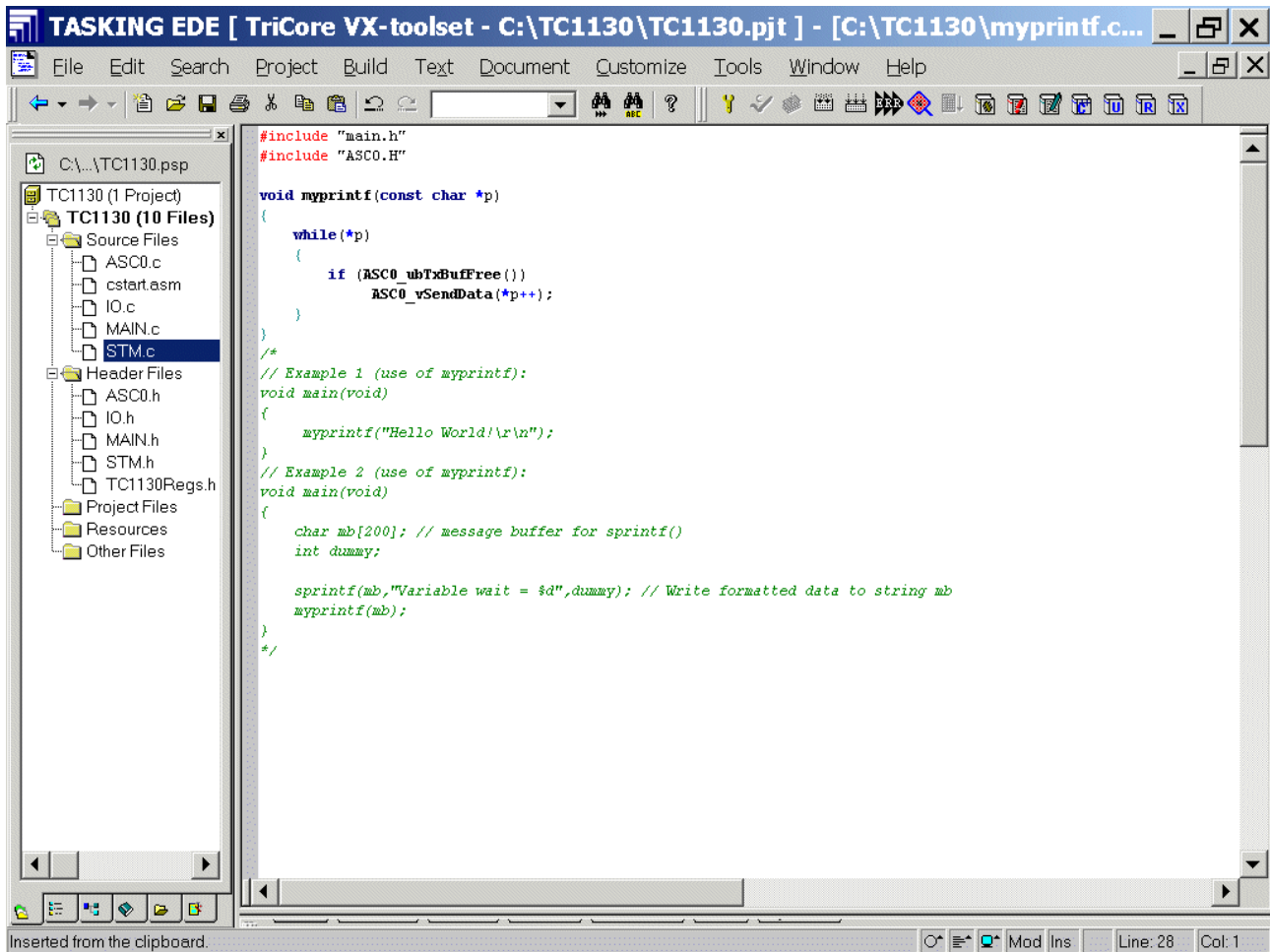
OK

Insert User Code for myprintf():

```
#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);
}
*/
```



TASKING EDE [TriCore VX-toolset - C:\TC1130\TC1130.pjt] - [C:\TC1130\myprintf.c...

File Edit Search Project Build Text Document Customize Tools Window Help

C:\...TC1130.psp

- TC1130 (1 Project)
 - TC1130 (10 Files)
 - Source Files
 - ASC0.c
 - cstart.asm
 - IO.c
 - MAIN.c
 - STM.c
 - Header Files
 - ASC0.h
 - IO.h
 - MAIN.h
 - STM.h
 - TC1130Regs.h
 - Project Files
 - Resources
 - Other Files

```

#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;

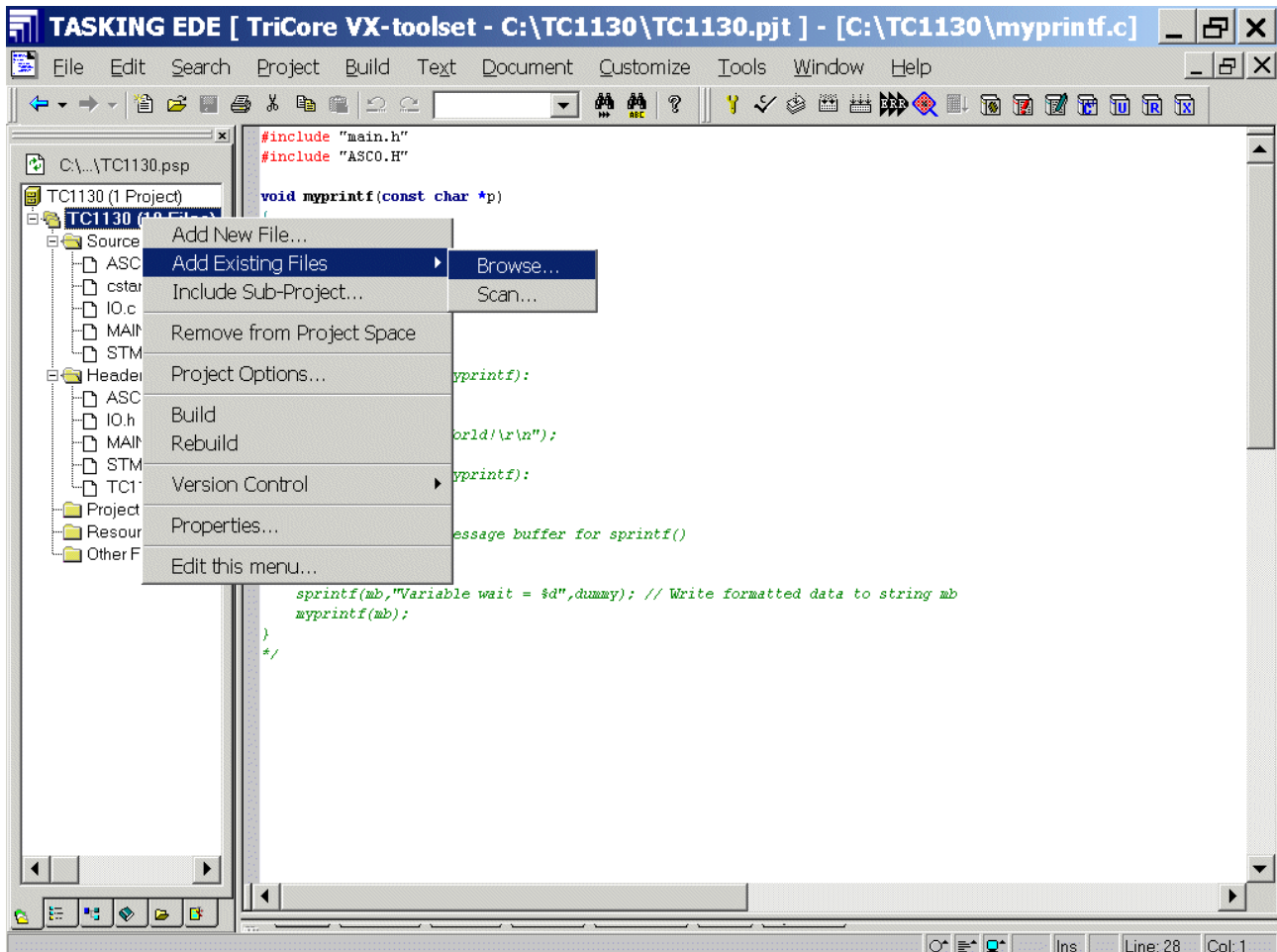
    sprintf(mb,"Variable wait = %d",dummy); // Write formatted data to string mb
    myprintf(mb);
}
*/

```

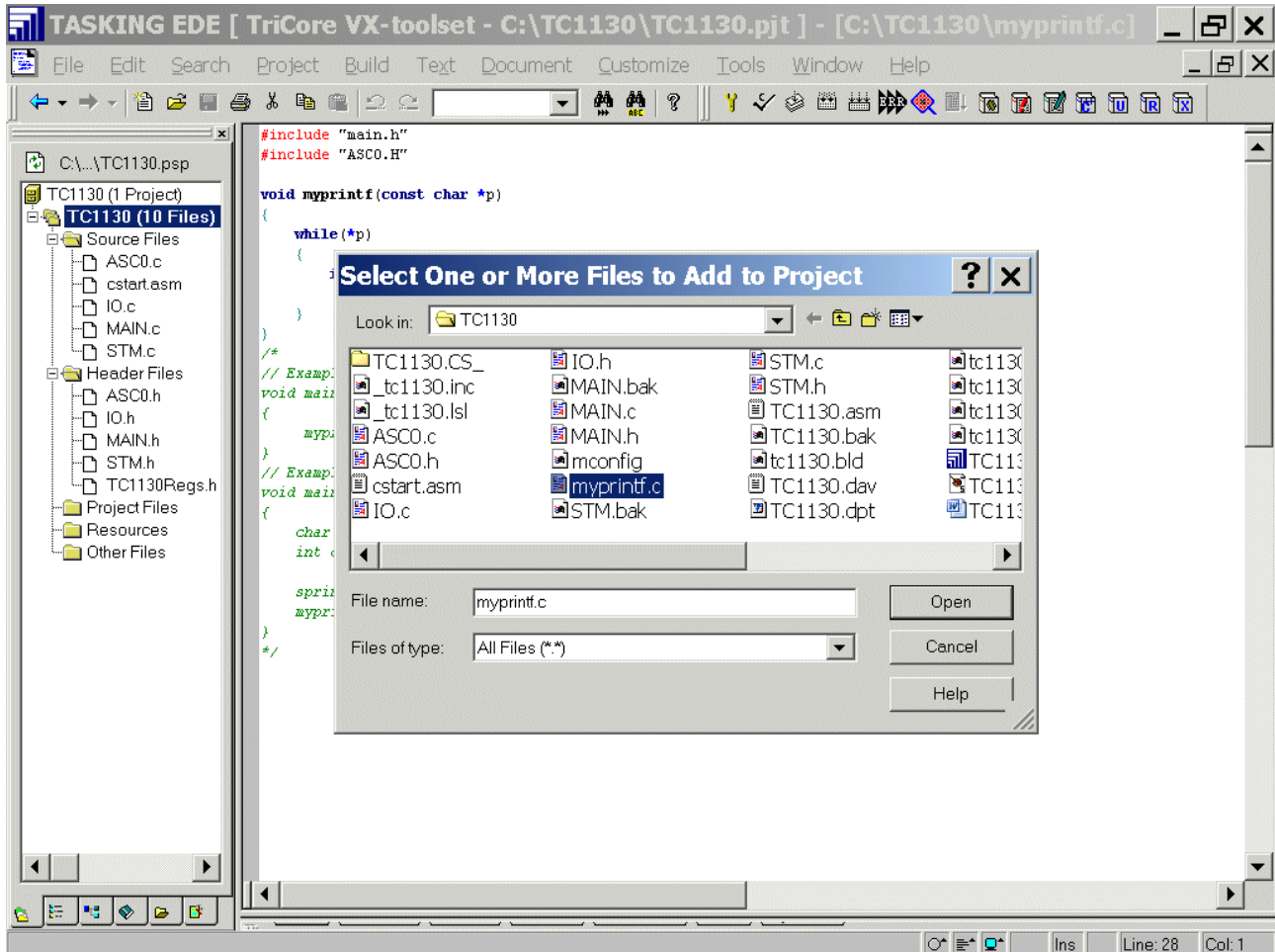
Inserted from the clipboard. Mod Ins Line: 28 Col: 1

File
Save all

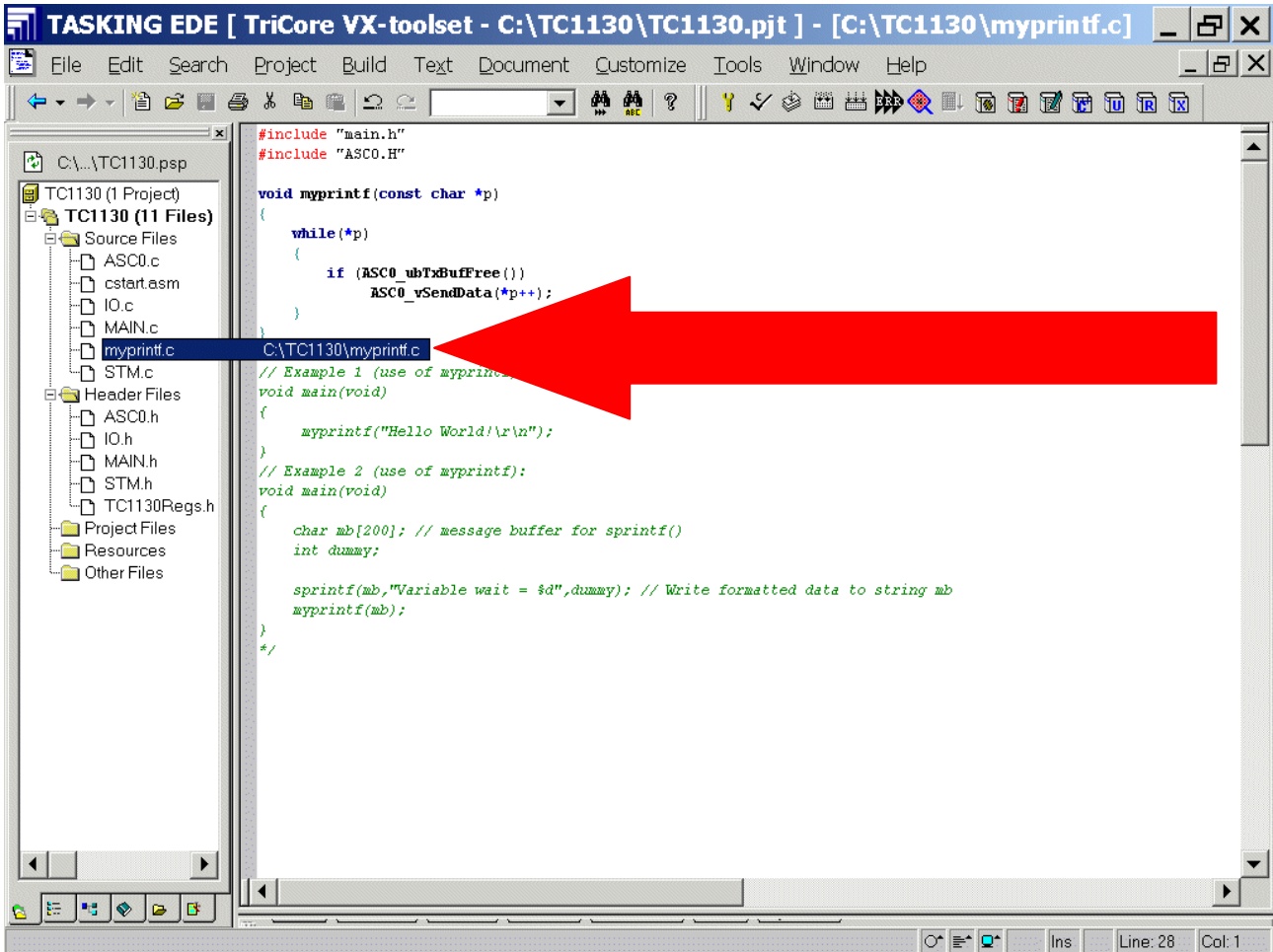
(Project Window **File View**) – TC1130 (Files) – **right mouse button click** – Add Existing Files – Browse



Select myprintf.c

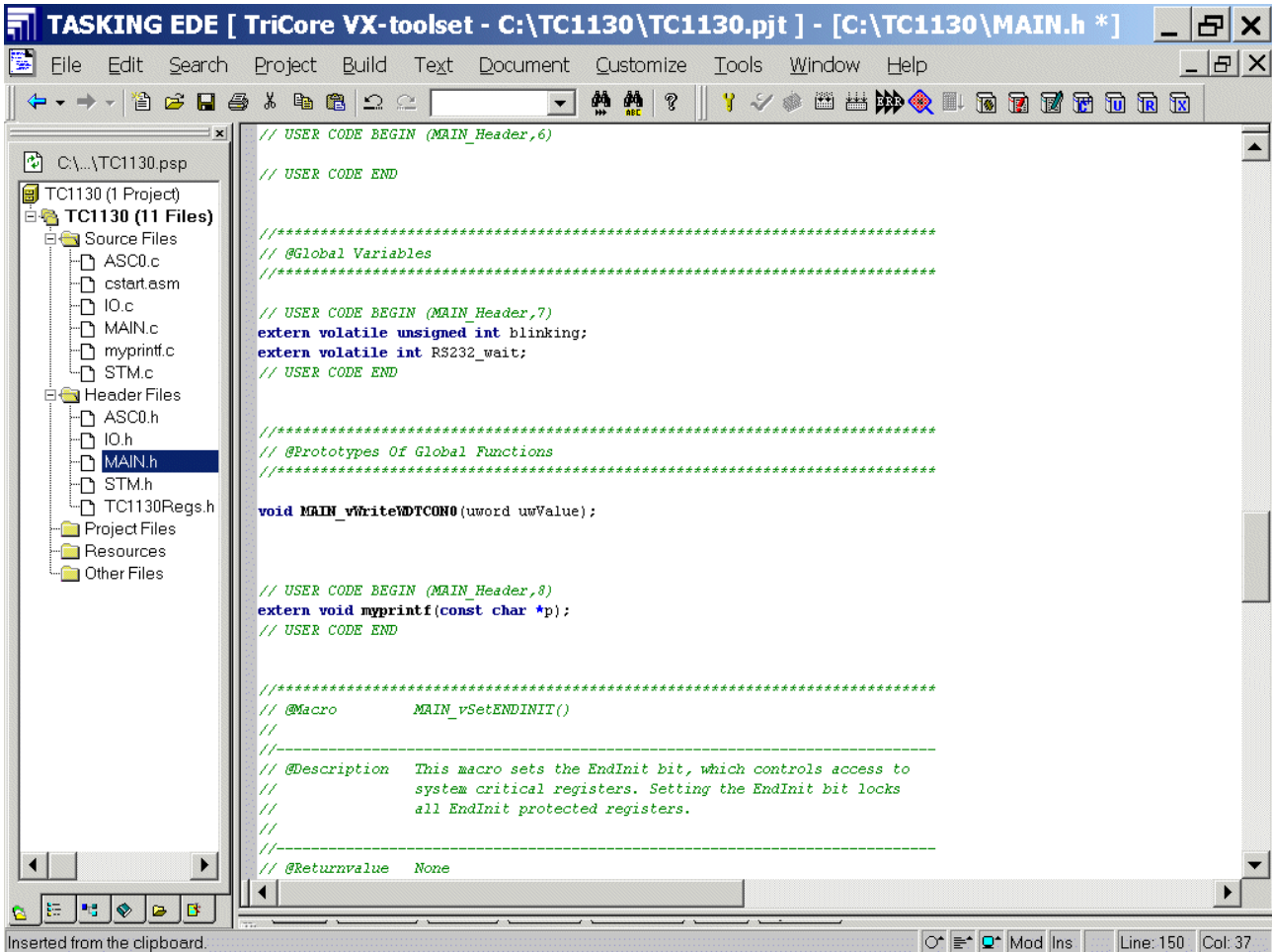


Open - OK



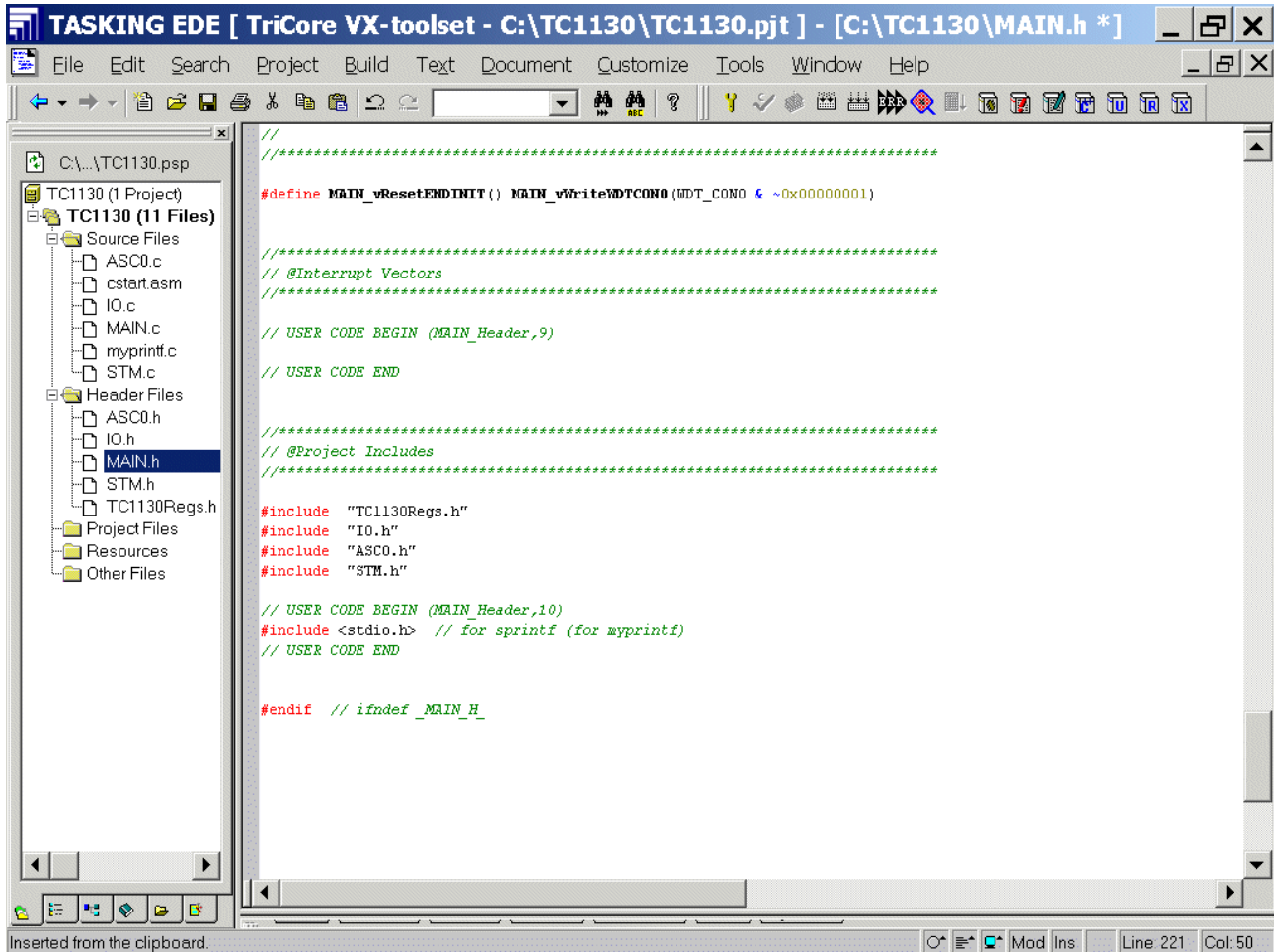
Double click: **Main.h** and insert Prototypes of Global Functions (Extern Declaration):

```
extern void myprintf(const char *p);
```



Double click: [Main.h](#) and insert required Header for printf:

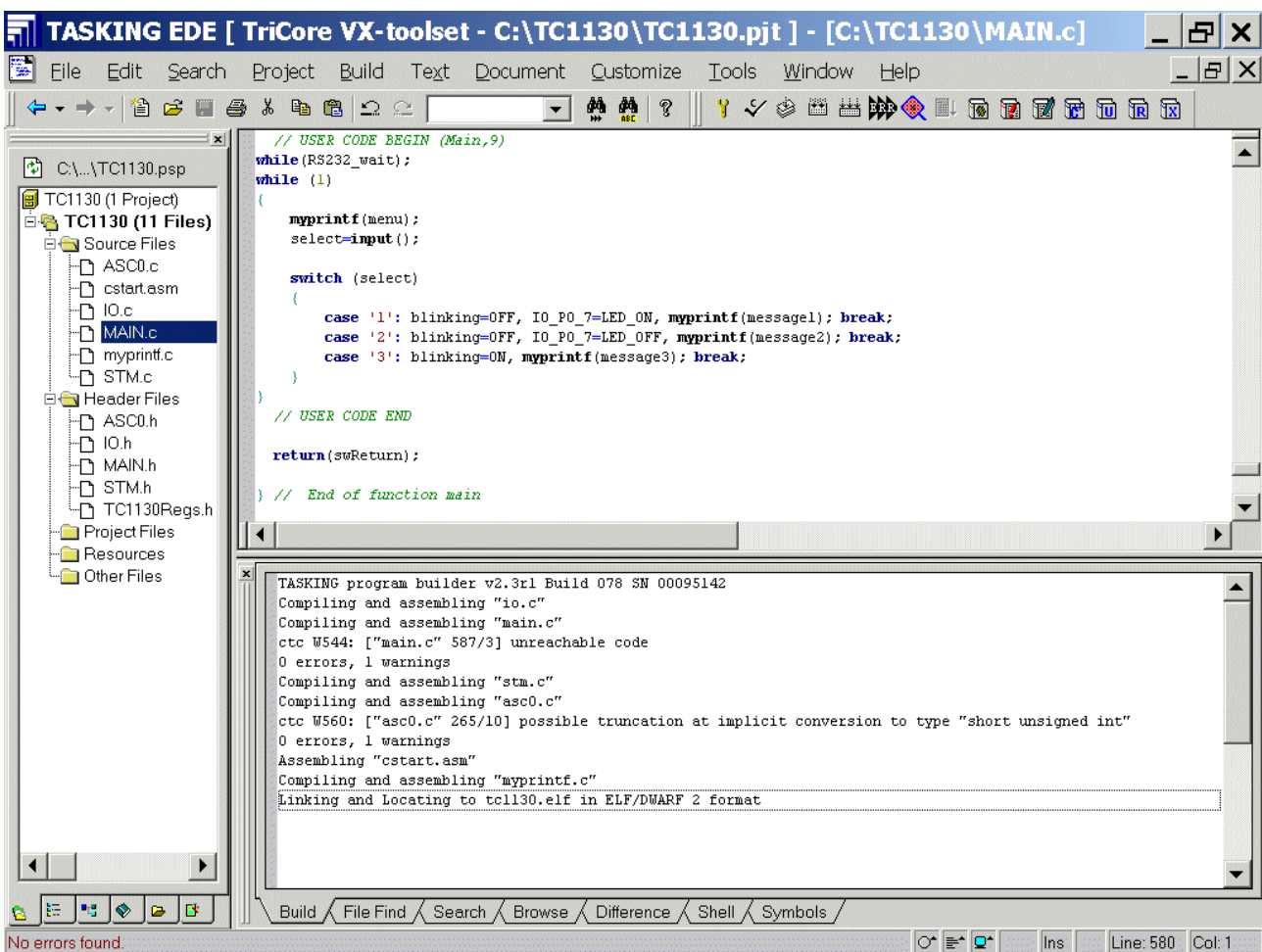
```
#include <stdio.h> // for printf (for myprintf)
```



Generate your application program:

Build
Rebuild

OR



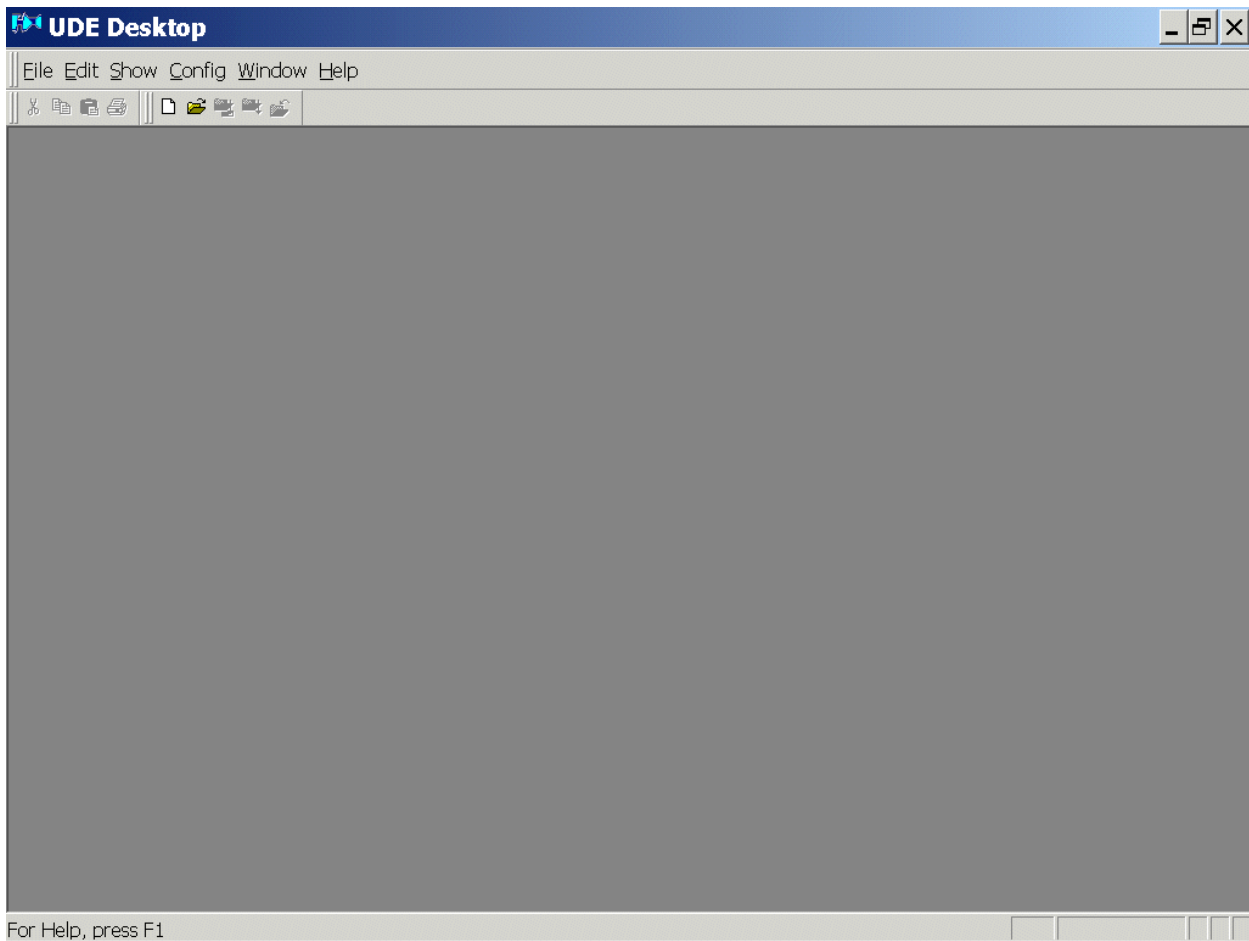
Now you can close both your project and Tasking EDE:

File - Close Project Space
File - Exit

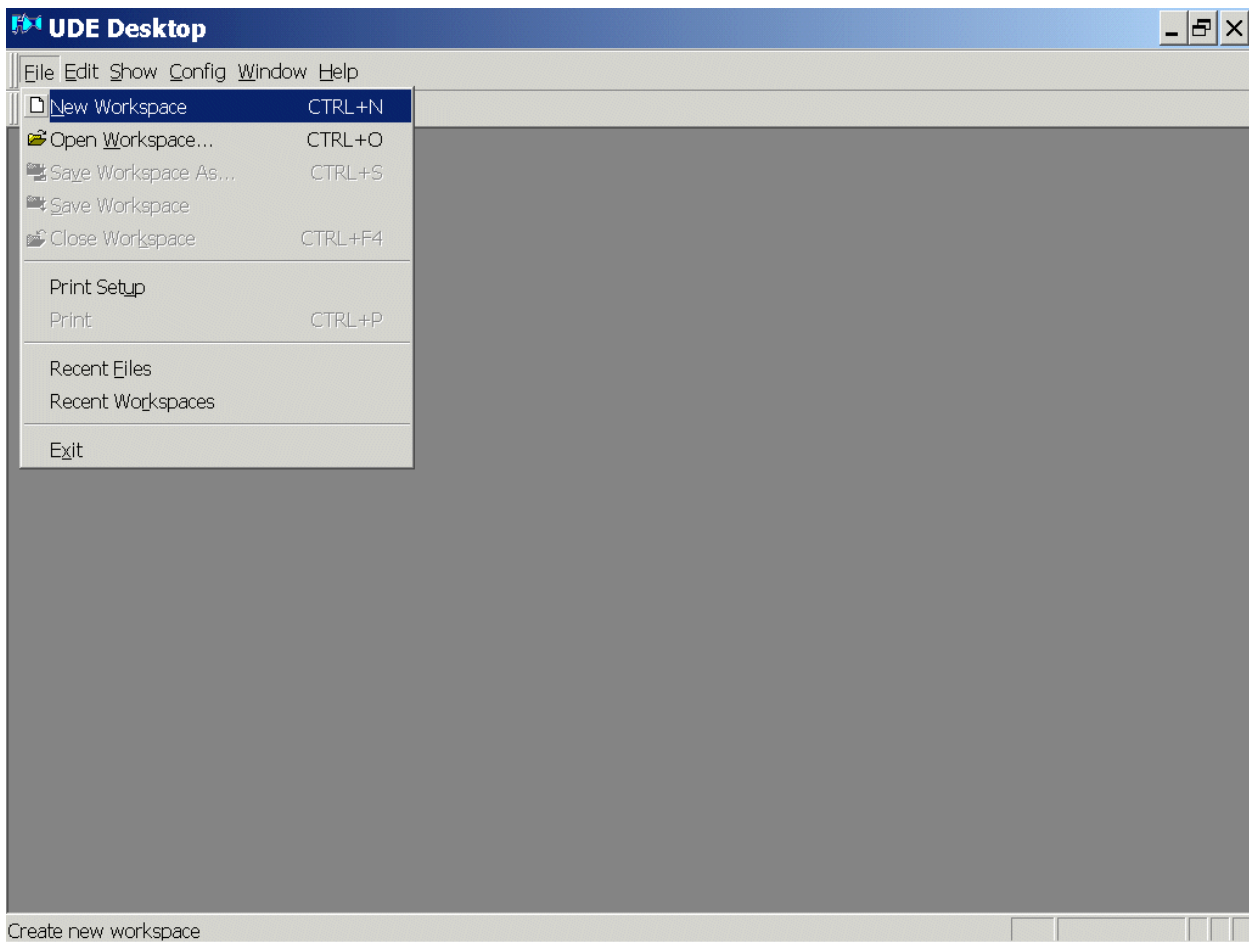
Programming is now complete. You can now **load** and **run** your program
(Therefore we are going to use the pls Debugger):



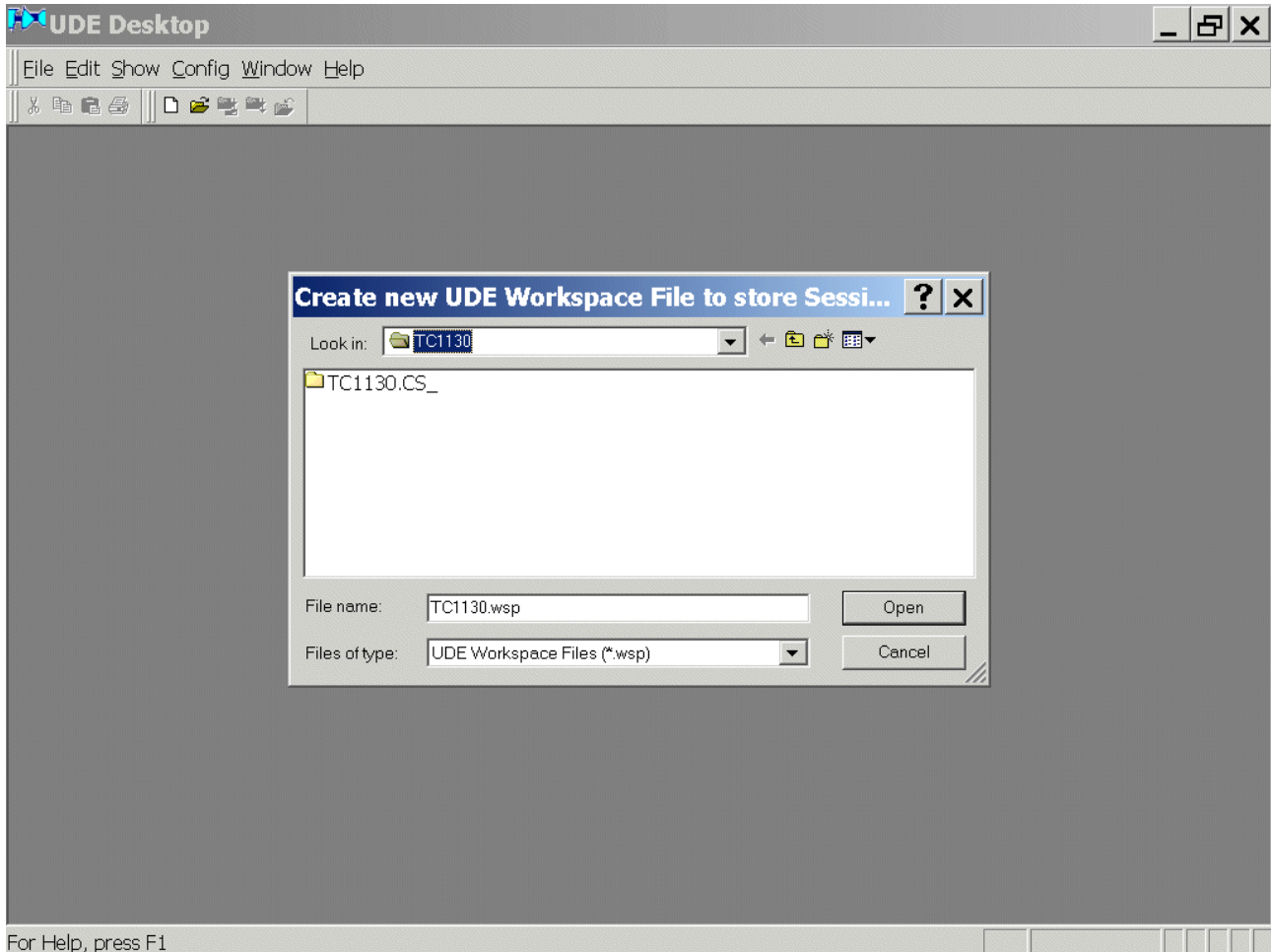
Start pls-Debugger



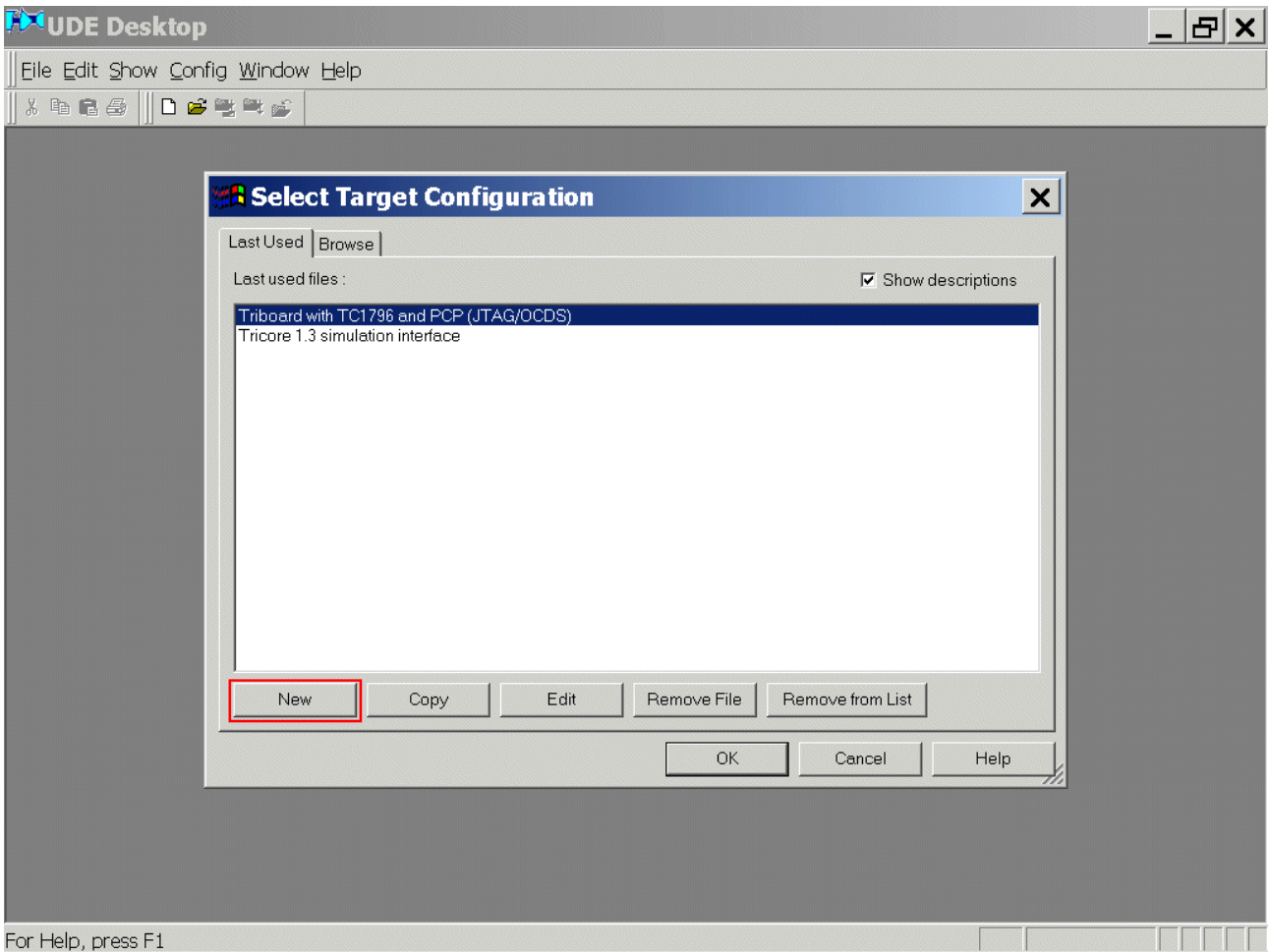
File – New Workspace



Look in: **select** C:\TC1130

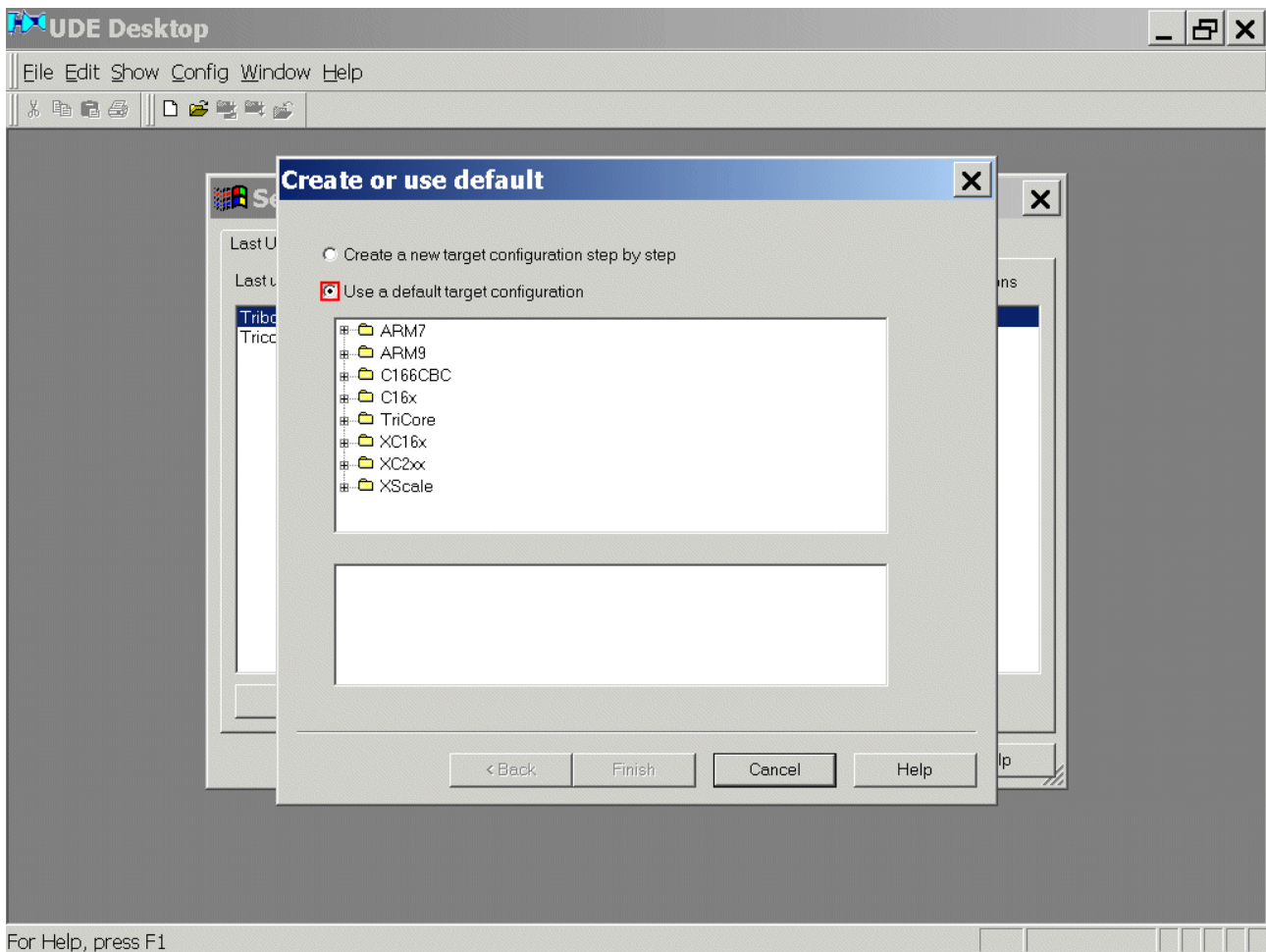


Open

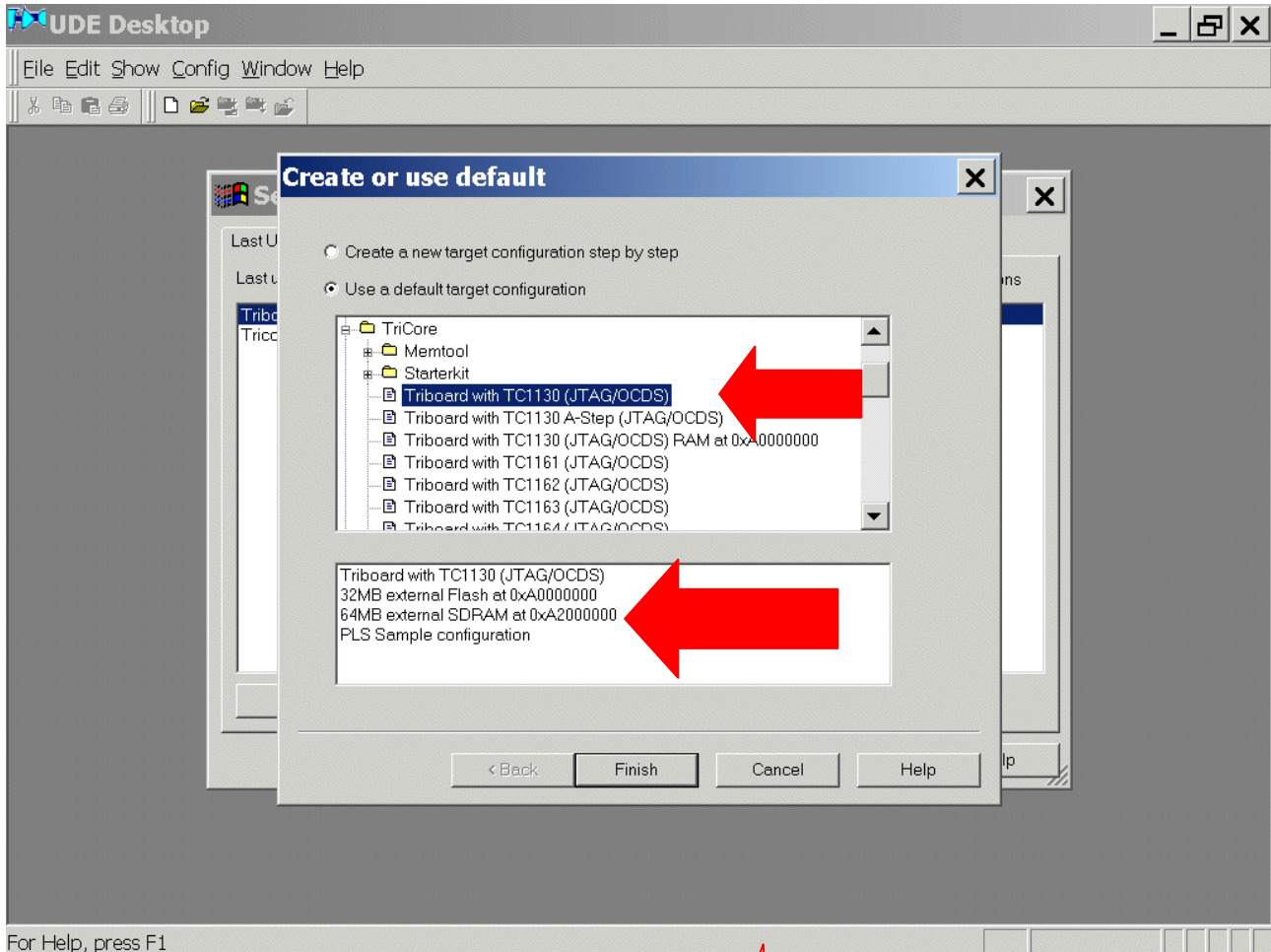


New

Click Use a default target configuration

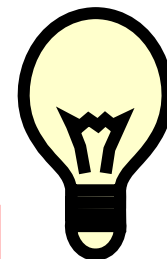


Select Triboard with TC1130 (JTAG/OCDS)

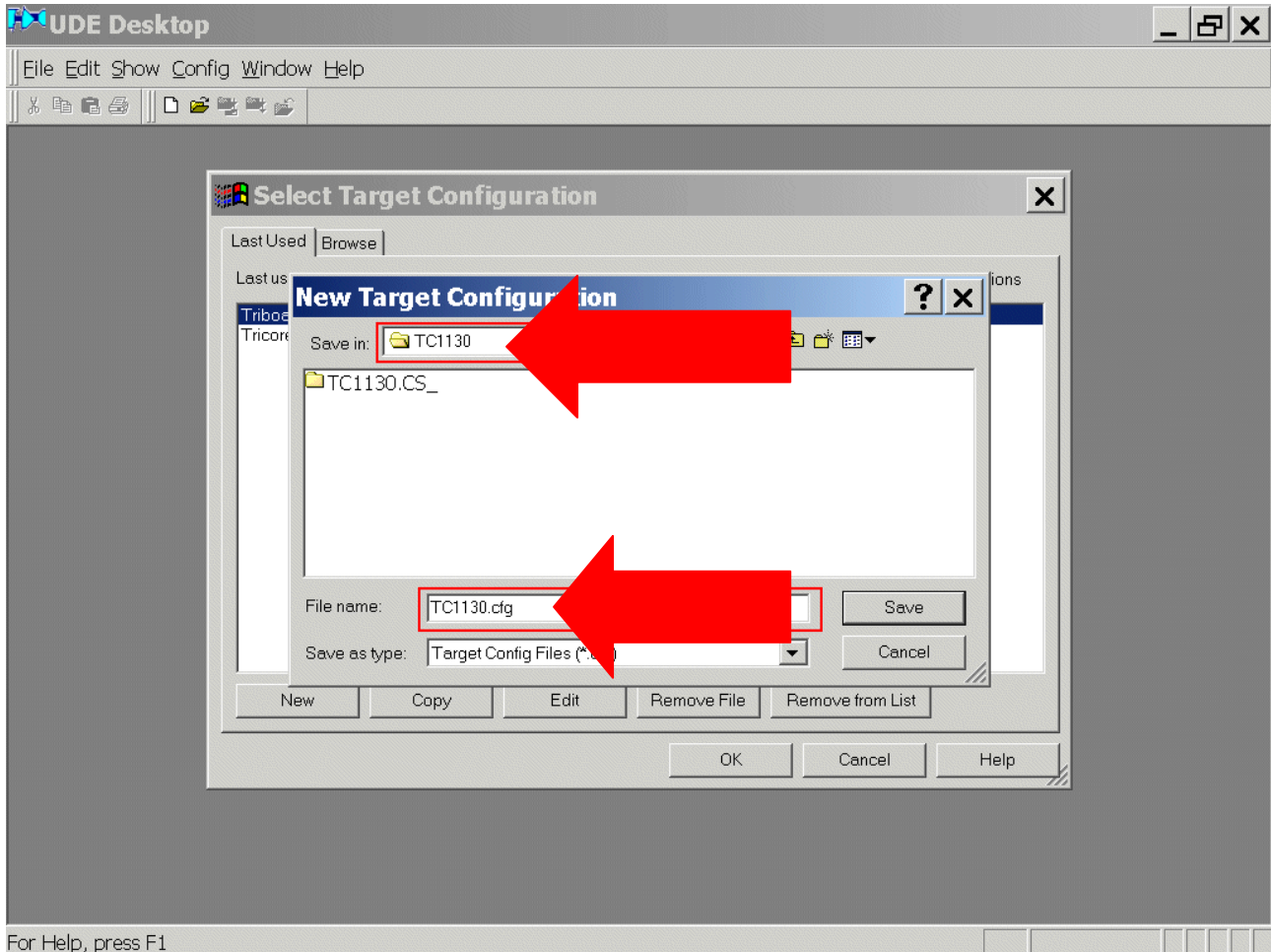


Finish

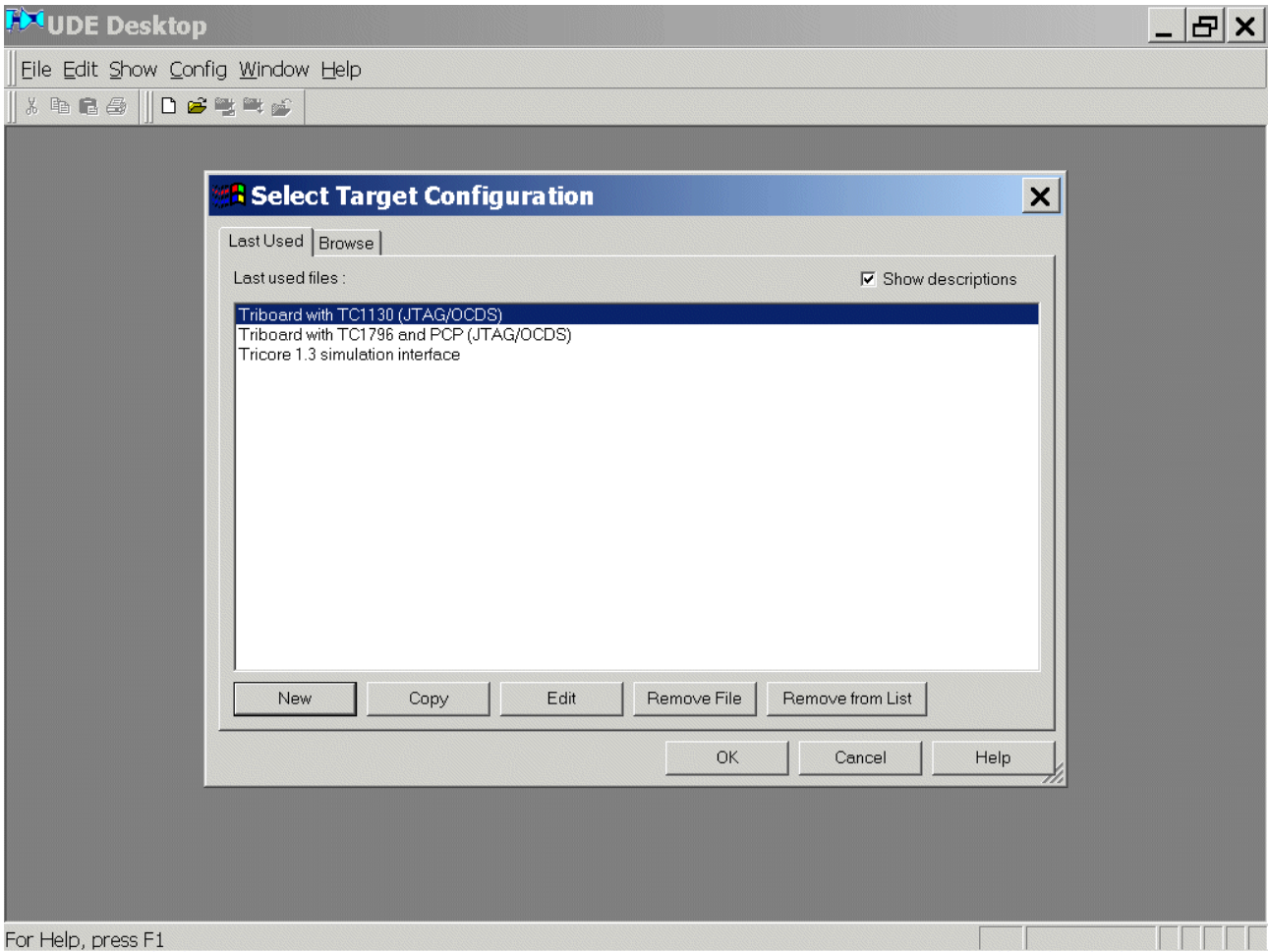
Note: The correct value for „64MB external SDRAM at 0xA2000000” will be 0xA4000000 later!



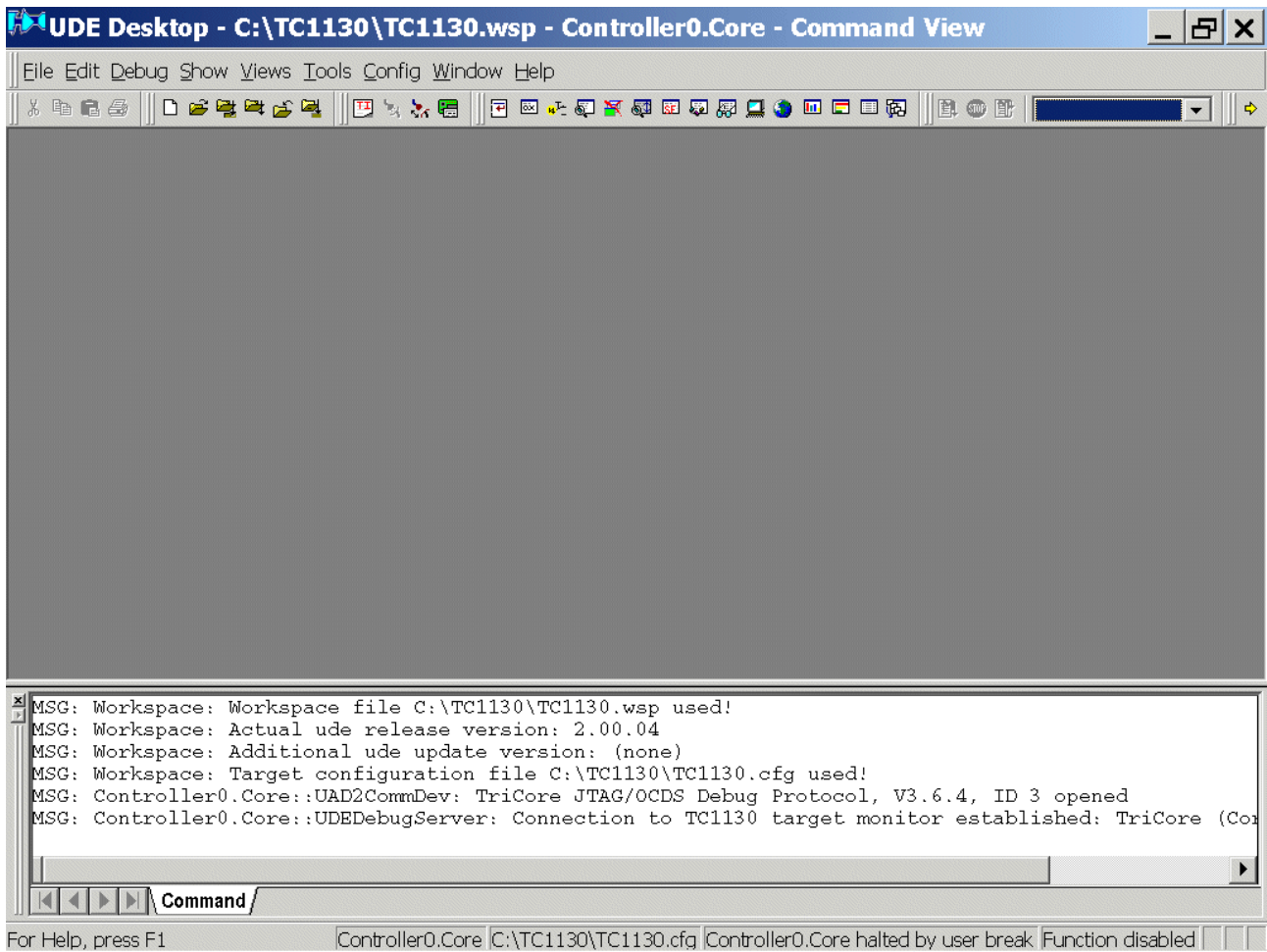
Save in: **select** C:\TC1130
File name: **insert** TC1130



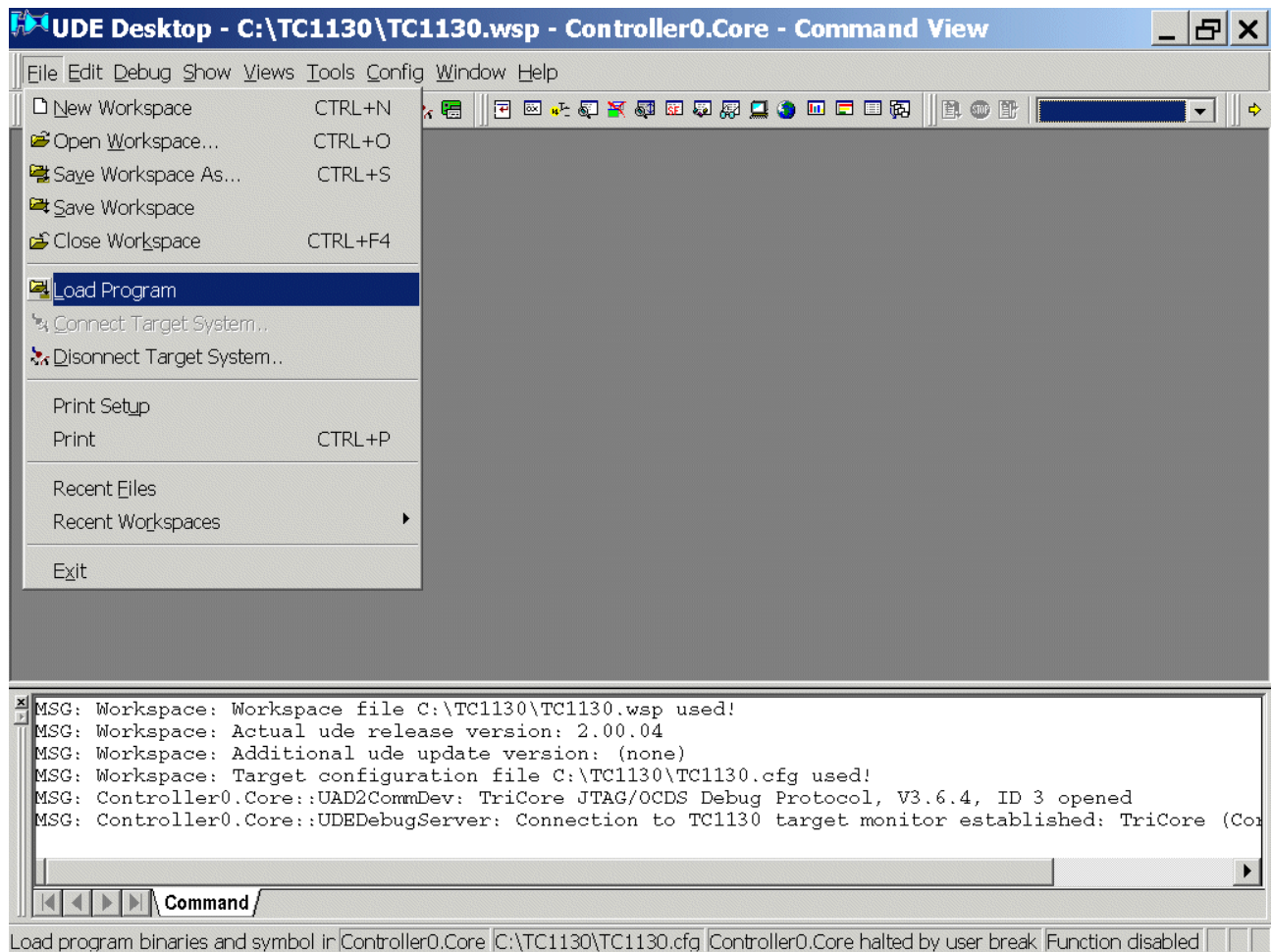
Save



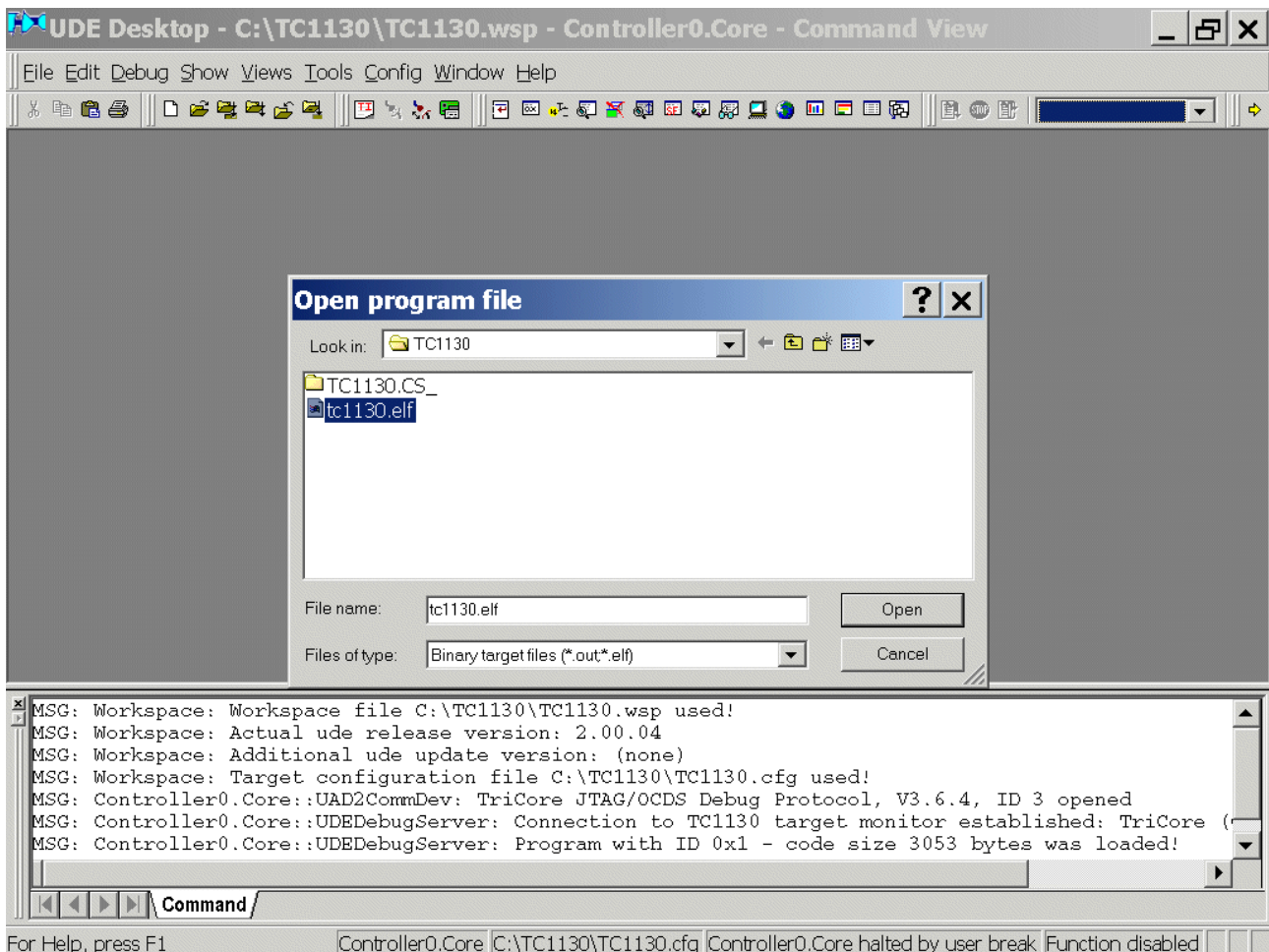
OK



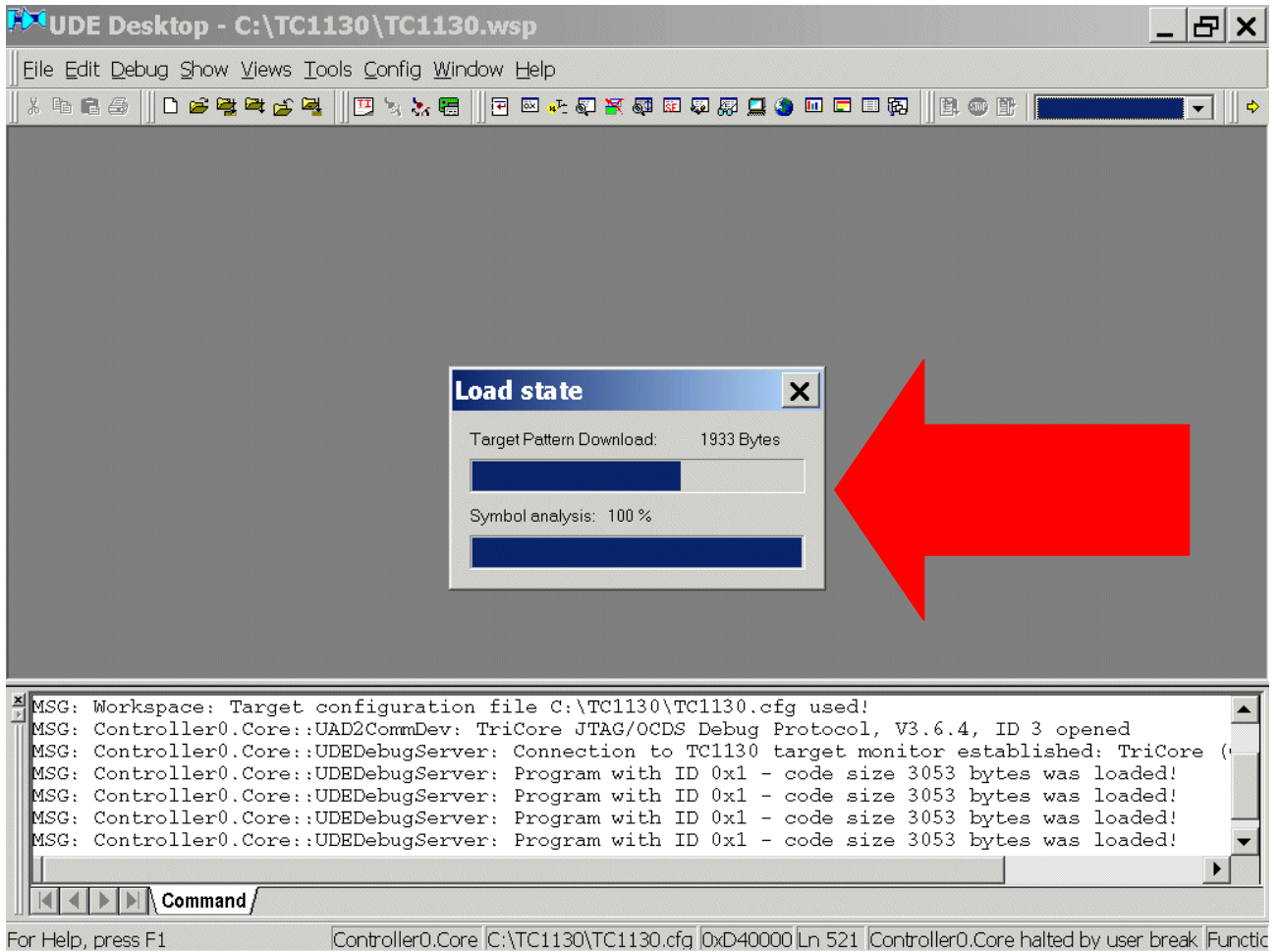
File – Load Program



Select TC1130.elf



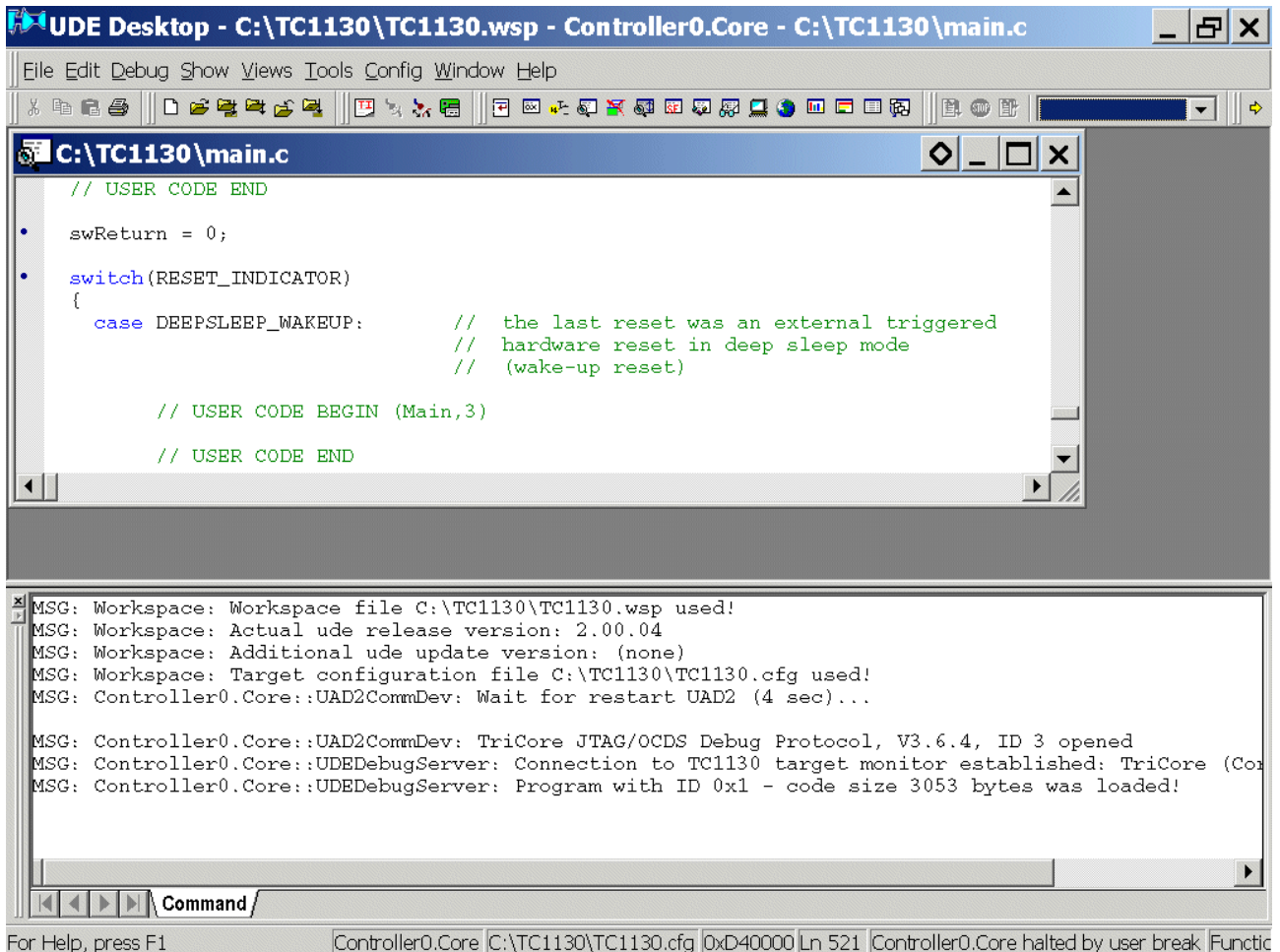
Open



The screenshot shows the UDE Desktop interface. At the top, the title bar reads "UDE Desktop - C:\TC1130\TC1130.wsp". Below the title bar is a menu bar with "File", "Edit", "Debug", "Show", "Views", "Tools", "Config", "Window", and "Help". A toolbar with various icons is located below the menu bar. The main workspace area is mostly empty, with a "Load state" dialog box centered. The dialog box has a blue header with the text "Load state" and a close button. It contains two progress bars: the first is labeled "Target Pattern Download: 1933 Bytes" and is nearly full; the second is labeled "Symbol analysis: 100%" and is also nearly full. A large red arrow points from the right side of the dialog box towards the left. Below the workspace is a command window with a scroll bar. The command window contains the following text:

```
MSG: Workspace: Target configuration file C:\TC1130\TC1130.cfg used!  
MSG: Controller0.Core::UAD2CommDev: TriCore JTAG/OCDS Debug Protocol, V3.6.4, ID 3 opened  
MSG: Controller0.Core::UDEDebugServer: Connection to TC1130 target monitor established: TriCore (  
MSG: Controller0.Core::UDEDebugServer: Program with ID 0x1 - code size 3053 bytes was loaded!  
MSG: Controller0.Core::UDEDebugServer: Program with ID 0x1 - code size 3053 bytes was loaded!  
MSG: Controller0.Core::UDEDebugServer: Program with ID 0x1 - code size 3053 bytes was loaded!  
MSG: Controller0.Core::UDEDebugServer: Program with ID 0x1 - code size 3053 bytes was loaded!
```

At the bottom of the window, there is a status bar with the text "For Help, press F1" on the left, "Controller0.Core C:\TC1130\TC1130.cfg |0xD40000 Ln 521" in the center, and "Controller0.Core halted by user break" on the right.



The screenshot shows the UDE Desktop IDE interface. The main window displays the source code for `C:\TC1130\main.c`. The code includes a `switch` statement for `RESET_INDICATOR` with a `case` for `DEEPSLEEP_WAKEUP`. The command window at the bottom shows the following messages:

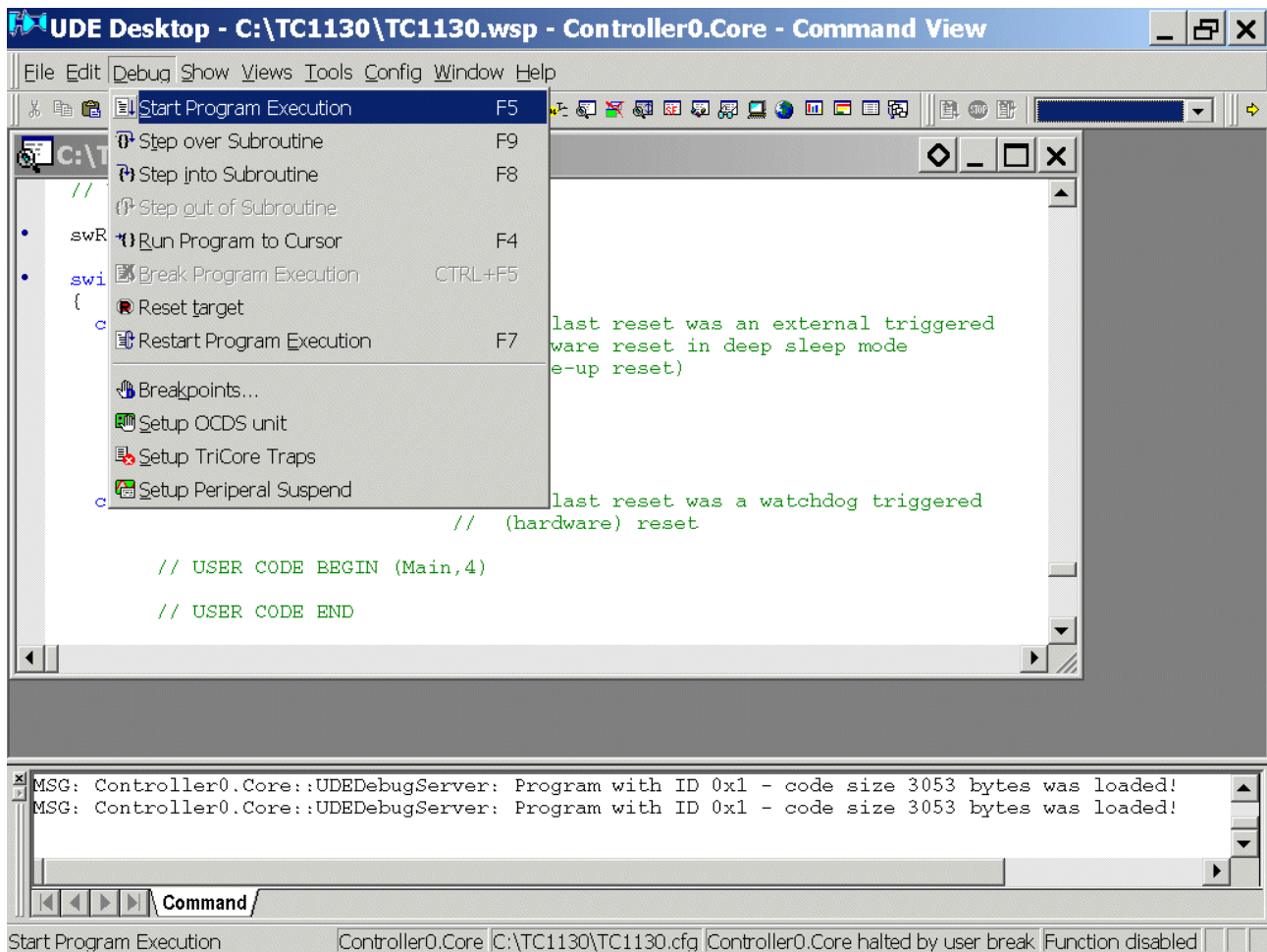
```

MSG: Workspace: Workspace file C:\TC1130\TC1130.wsp used!
MSG: Workspace: Actual ude release version: 2.00.04
MSG: Workspace: Additional ude update version: (none)
MSG: Workspace: Target configuration file C:\TC1130\TC1130.cfg used!
MSG: Controller0.Core::UAD2CommDev: Wait for restart UAD2 (4 sec)...

MSG: Controller0.Core::UAD2CommDev: TriCore JTAG/OCDS Debug Protocol, V3.6.4, ID 3 opened
MSG: Controller0.Core::UDEDebugServer: Connection to TC1130 target monitor established: TriCore (Co
MSG: Controller0.Core::UDEDebugServer: Program with ID 0x1 - code size 3053 bytes was loaded!
  
```

The status bar at the bottom indicates: `For Help, press F1` | `Controller0.Core C:\TC1130\TC1130.cfg 0xD40000 Ln 521` | `Controller0.Core halted by user break` | `Func...`

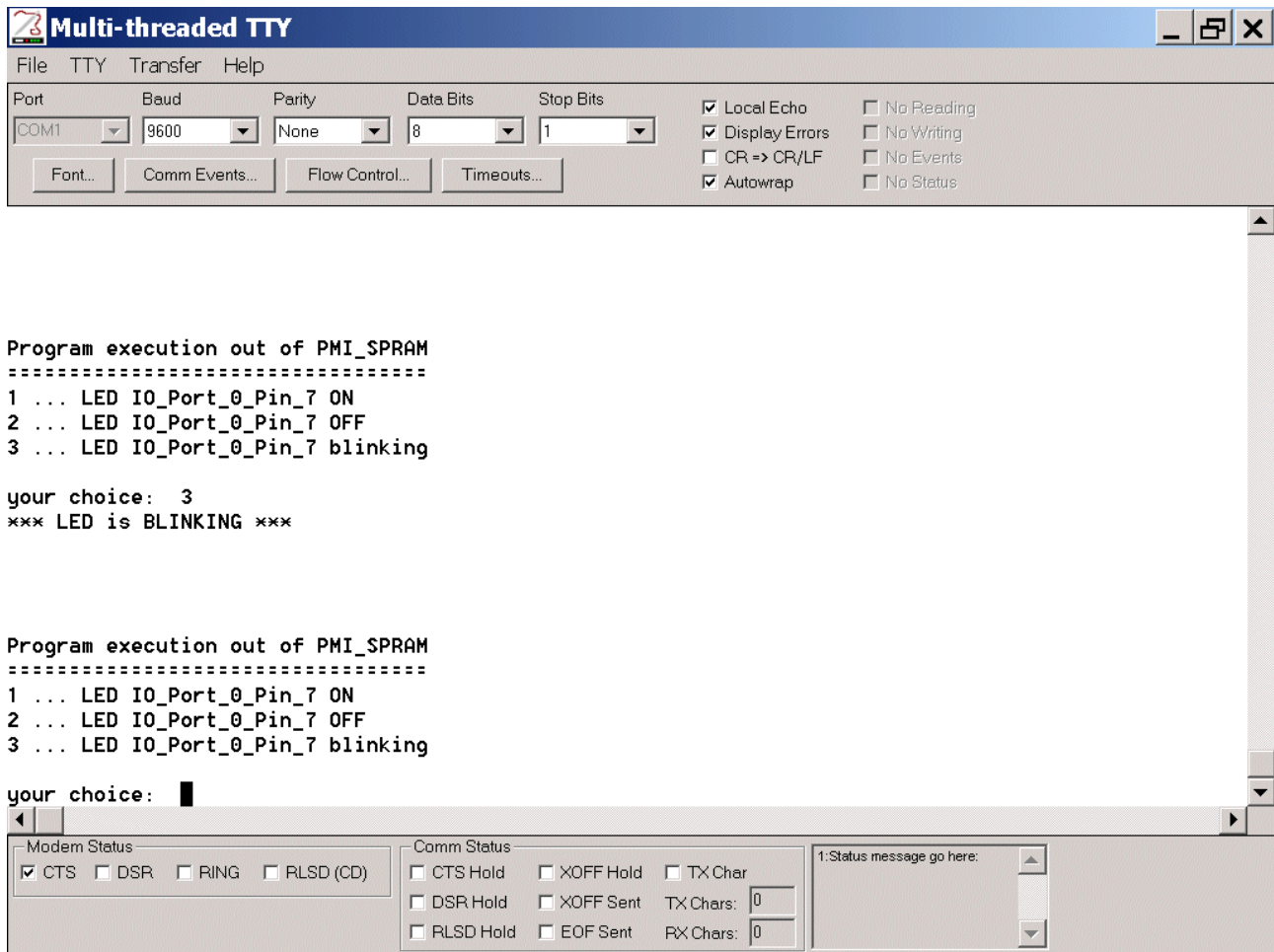
Debug – Start Program Execution



The screenshot shows the UDE Desktop interface for Controller0.Core. The 'Debug' menu is open, displaying various options such as 'Start Program Execution' (F5), 'Step over Subroutine' (F9), 'Run Program to Cursor' (F4), and 'Reset target'. The main window displays assembly code with comments in green, including 'last reset was an external triggered', 'ware reset in deep sleep mode', and 'e-up reset)'. The console window at the bottom shows two messages: 'MSG: Controller0.Core::UDEDebugServer: Program with ID 0x1 - code size 3053 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:

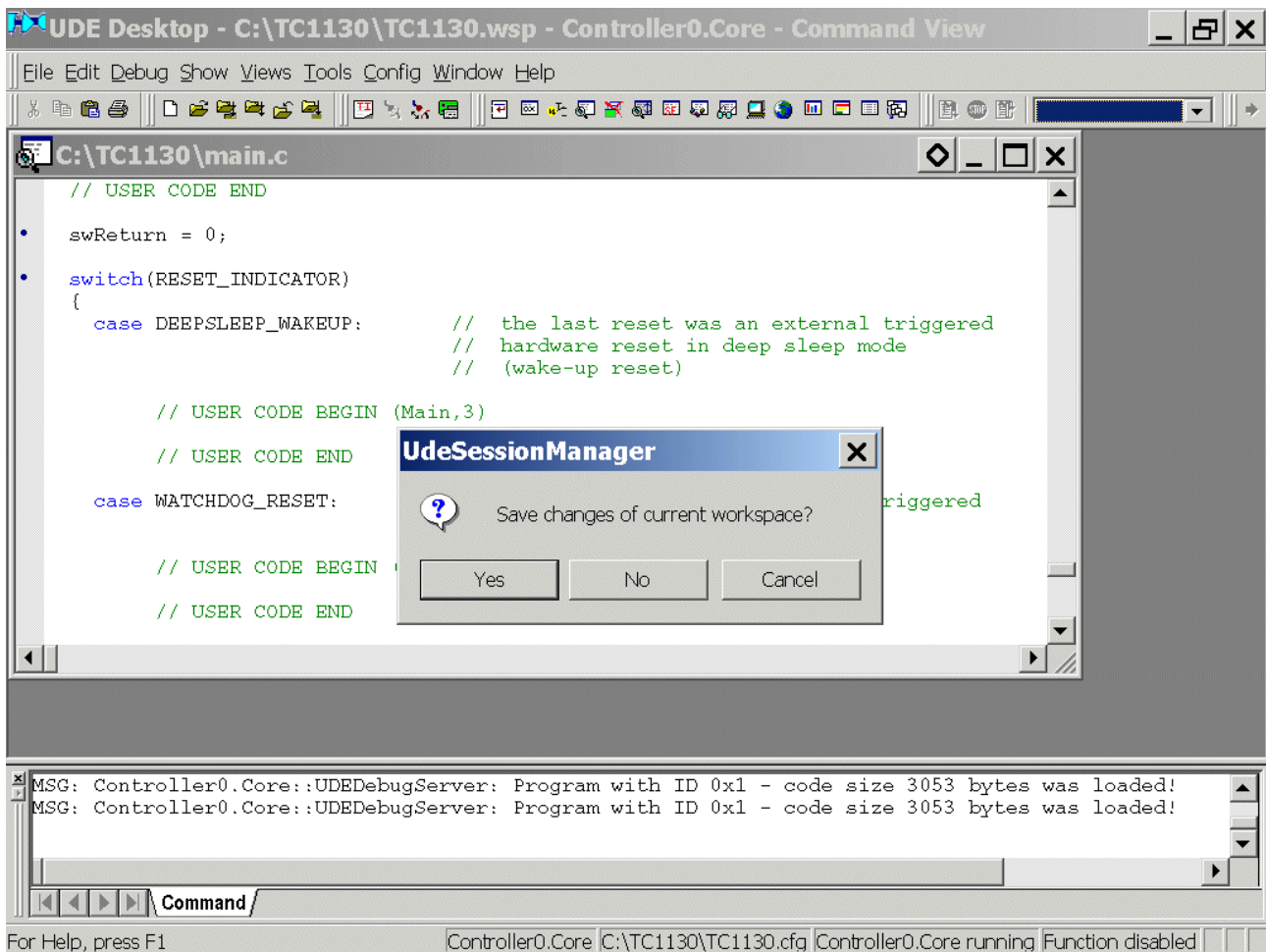


Note:

As an example for “any terminal-program” we use mttty.

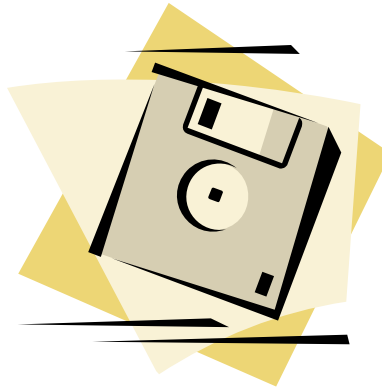
MTTY can be downloaded @ http://www.freeware.de/software/DetailEN_MTTY_19383.html

File – Close Workspace

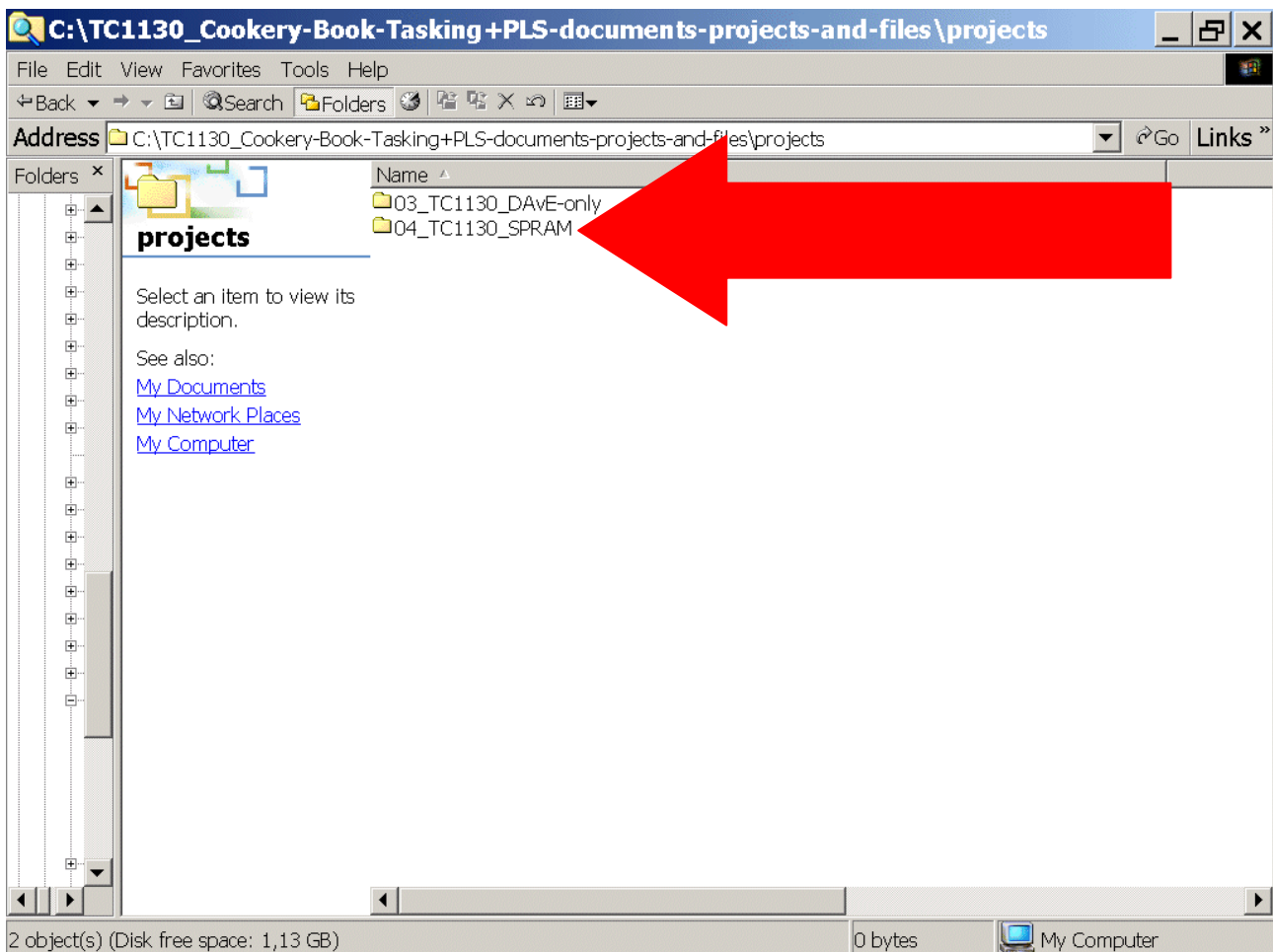


Yes

File - Exit



We recommend now to **copy and store** your project-directory “C:\TC1130” to “04_TC1130_SPRAM”:

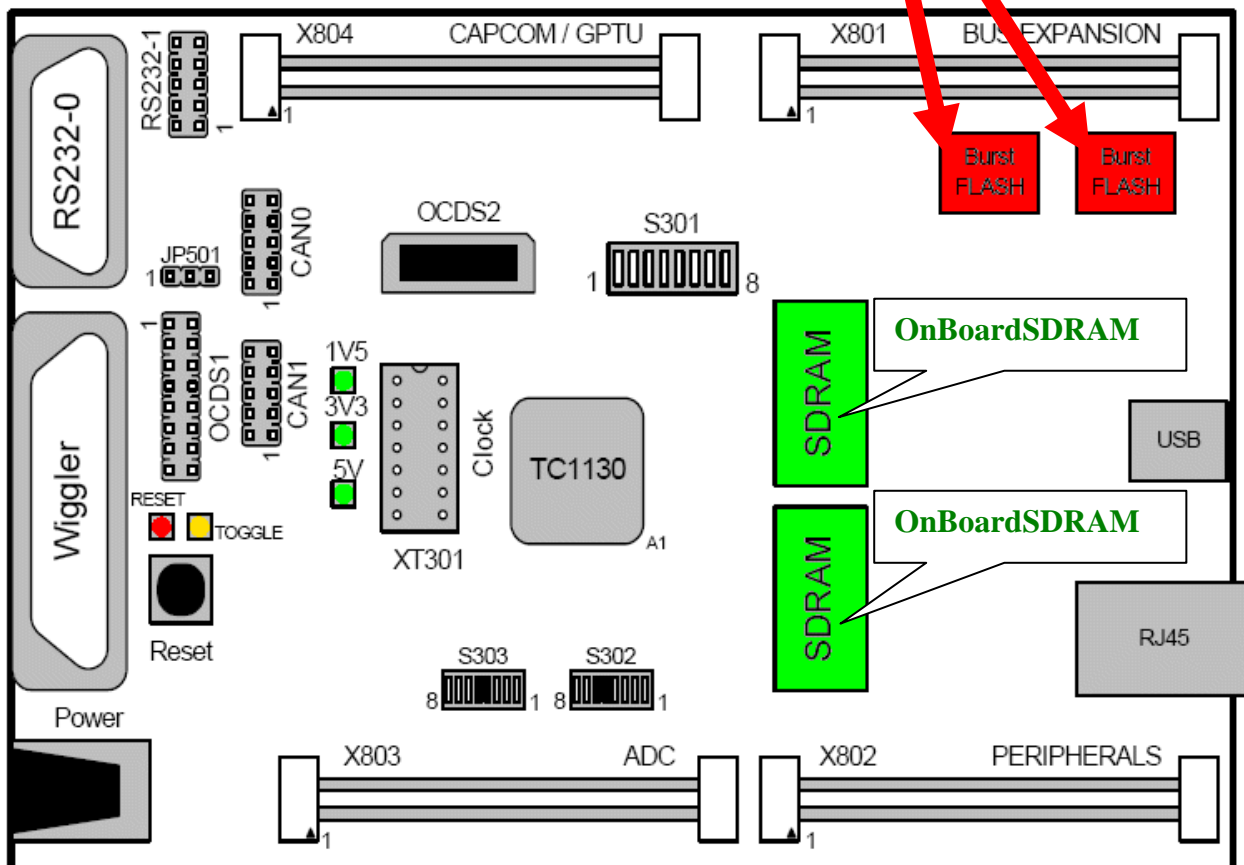


5.) Using of the TASKING - EDE Development Tools:

“ROM”:
Locating programs into the Intel’s 32 MBytes OnBoardFlash,
“RAM”:
Using
OnChipSRAM (DMI_SPRAM+DMU_SRAM) + Infineon’s 64 MBytes OnBoardSDRAM

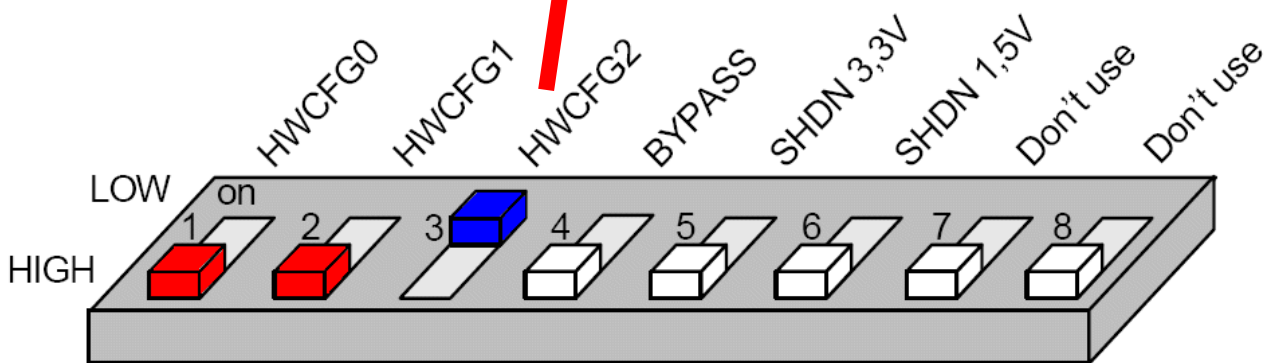
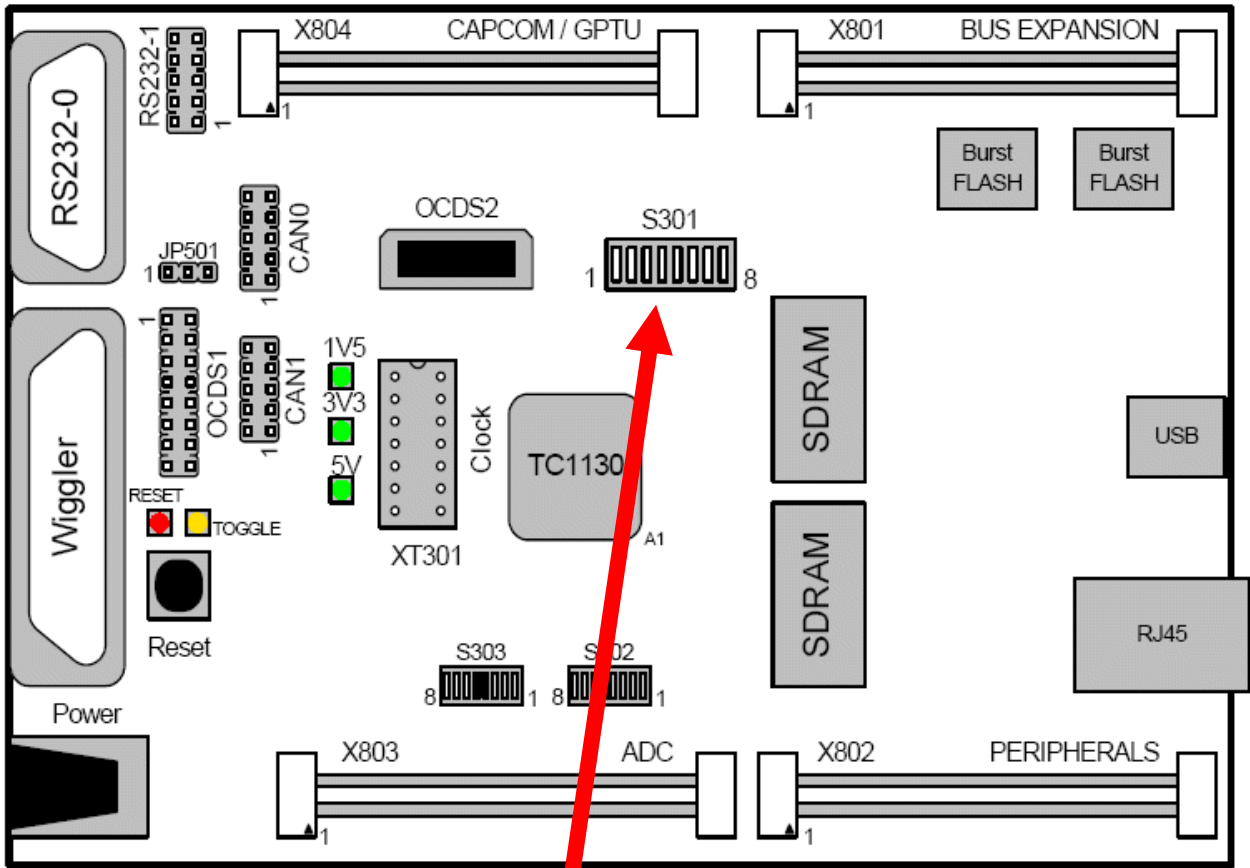


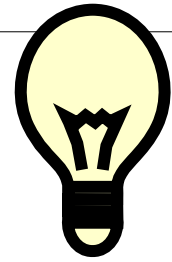
Write programs for execution from OnBoardFlash



TC1130-Execution-Environment: OnBoardFlash 

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

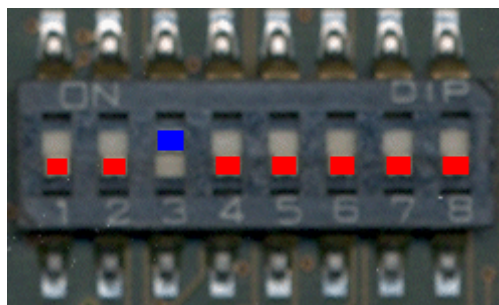
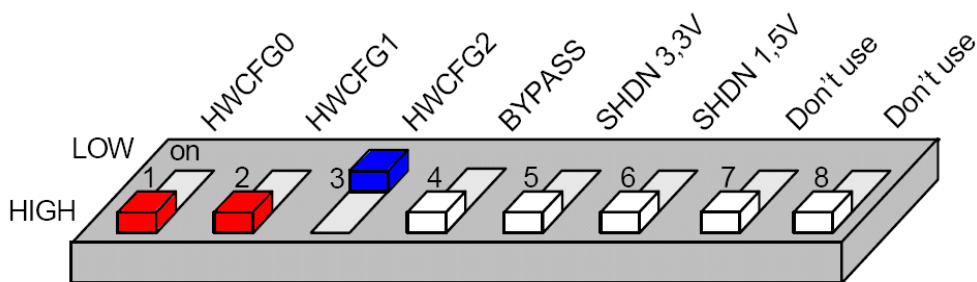
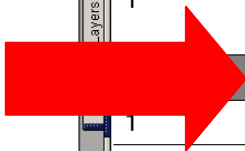




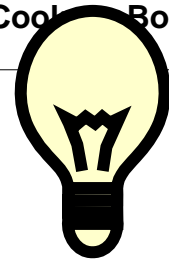
Additional Information:

Adobe Reader - [TC1130 TriBoard Manual.pdf]

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



S301:
3 : ON
1, 2, 4, 5, 6, 7, 8 : OFF



Additional Information:

Adobe Reader - [tc1130_um_v1.3_2004_11_sys.pdf]

File Edit View Document Tools Window Help

100%

Table 7-1 TC1130 Block Address Map

Segment	Address Range	Size	Description	DMI Acc.	PMI Acc.	
0-7	0000 0000 _H - 7FFF FFFF _H	2 GB	MMU Space	via FPI	via FPI	c a c h e d
8	8000 0000 _H - 8FFF FFFF _H	256 MB	External Memory Space mapped from Segment 10	via LMB	via LMB	
9	9000 0000 _H - 9FDF FFFF _H	256 MB	Reserved	via FPI	via FPI	
	A000 0000 _H - AFBF FFFF _H	252 MB	External Memory Space	via LMB	via LMB	n o n - c a c h e d
	AFC0 0000 _H - AFC0 FFFF _H	64 KB	DMU Space			
	AFC1 0000 _H - AFFF FFFF _H	≈ 4 MB	Reserved			
11	B000 0000 _H - BFFF FFFF _H	256 MB	Reserved	via FPI	via FPI	h e d

8,15 x 11,58 in

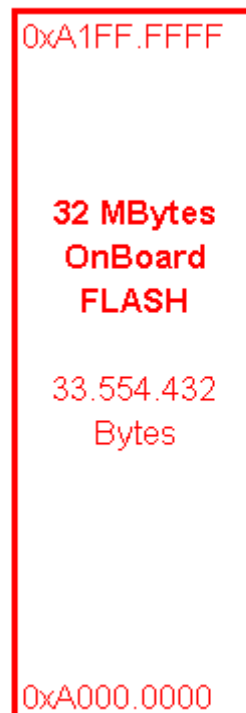
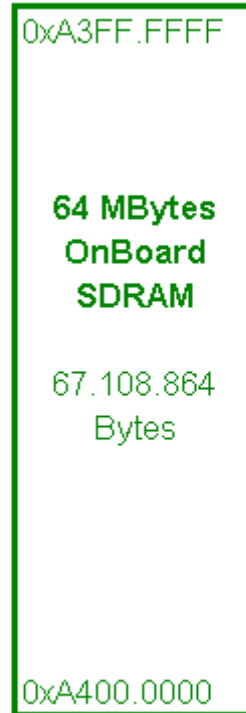
242 of 979

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:





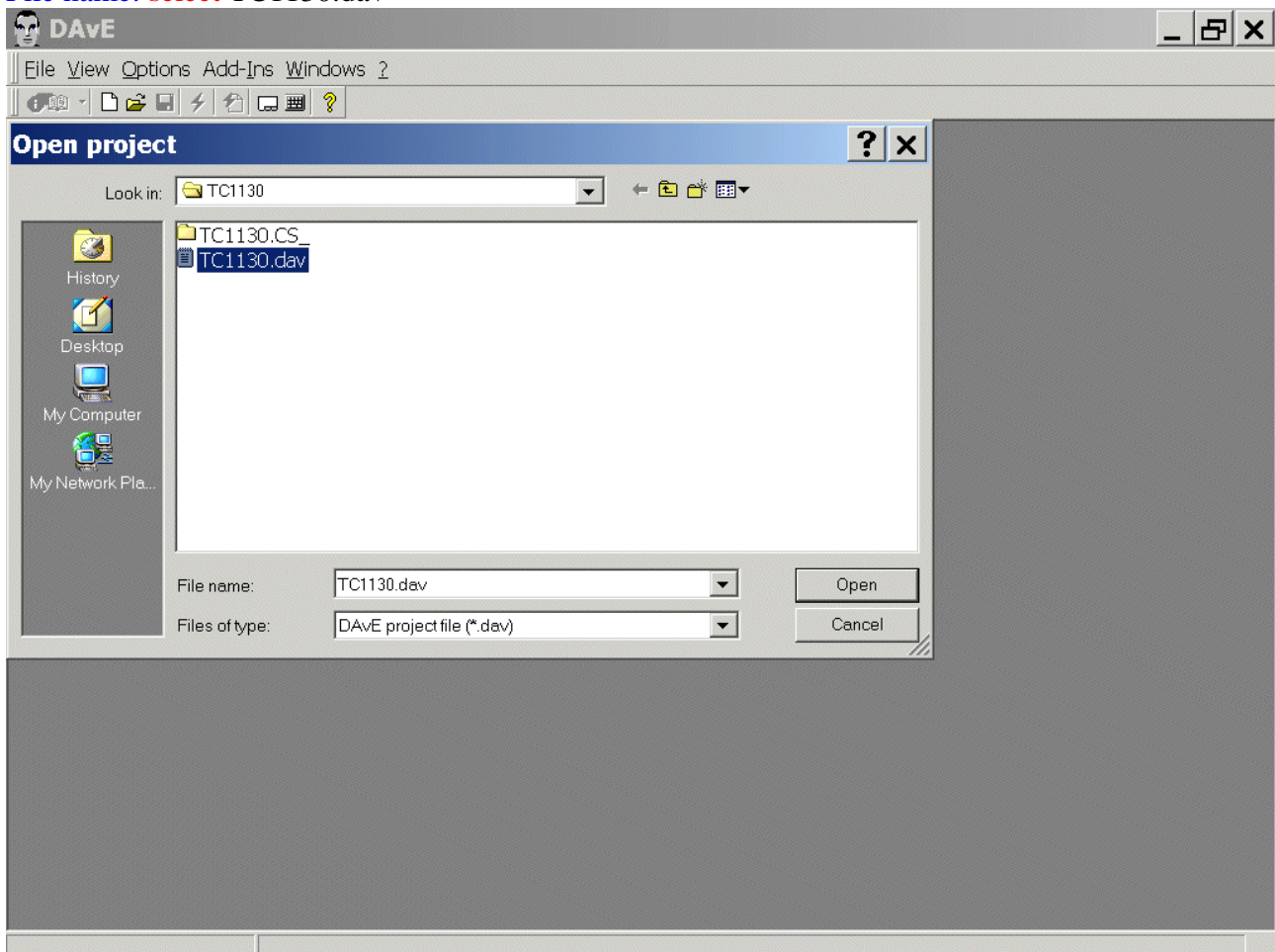
Start DAVe and open the TC1130 project:

File

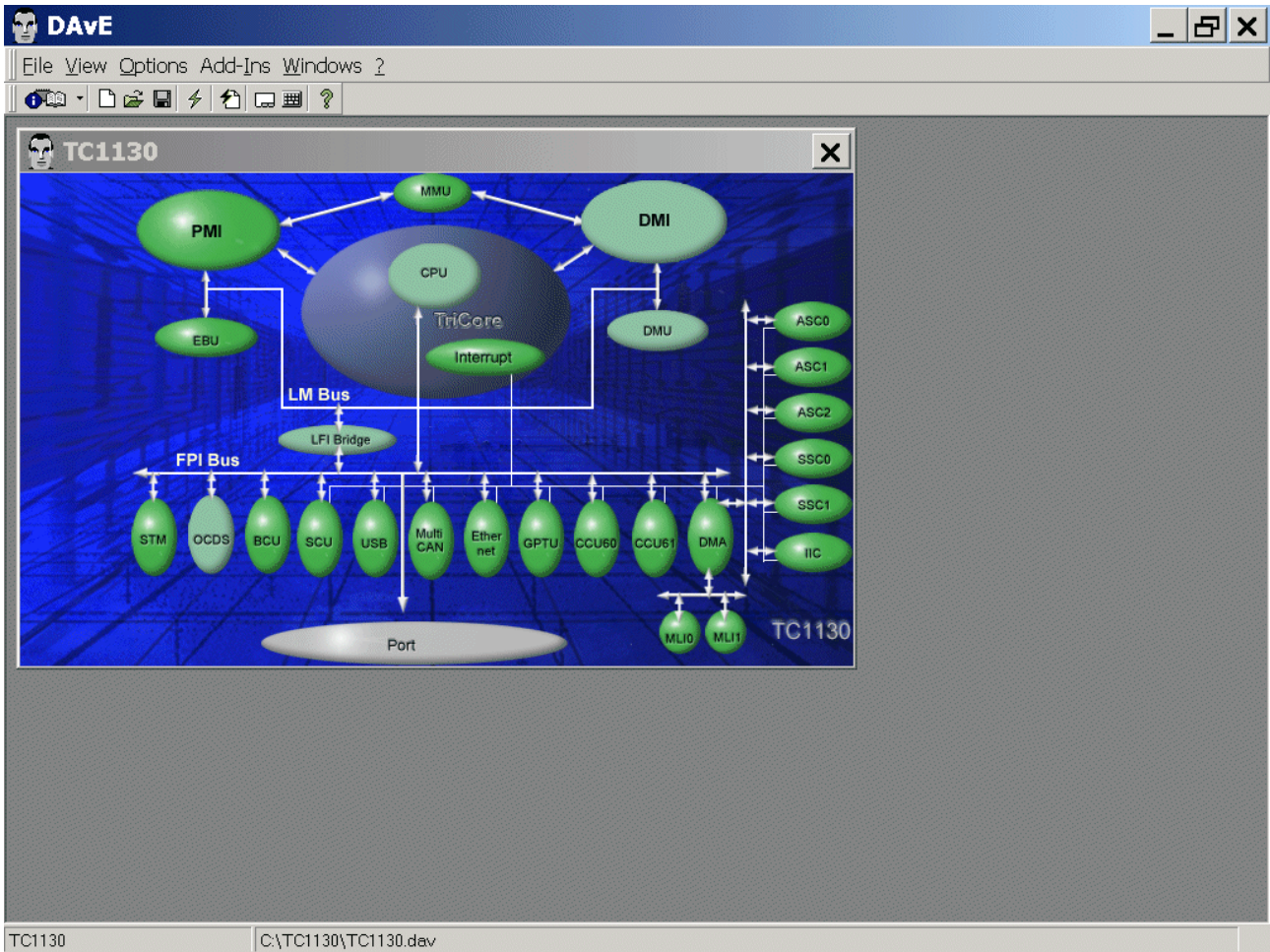
Open

Look in: select C:\TC1130

File name: select TC1130.dav



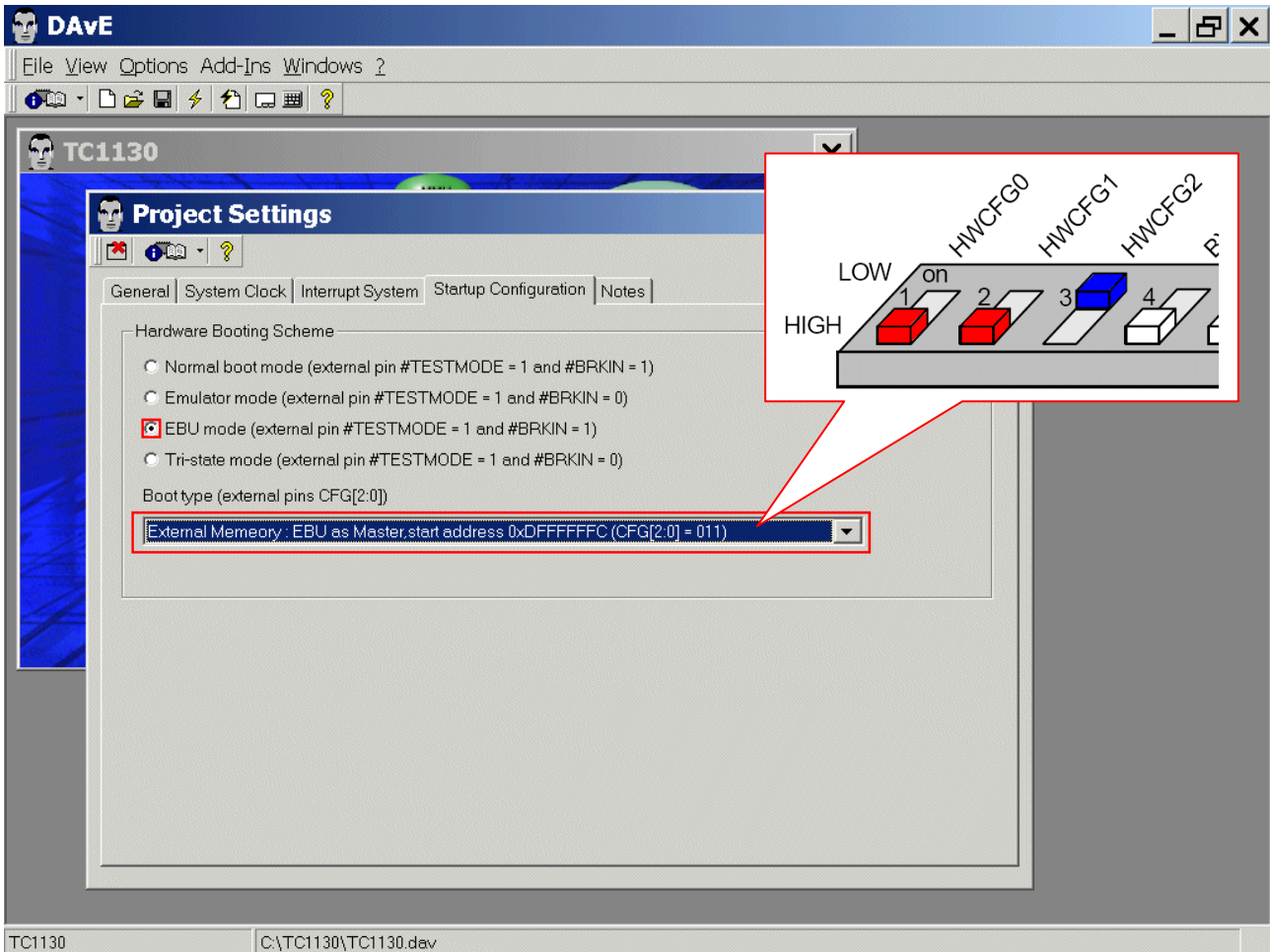
Open



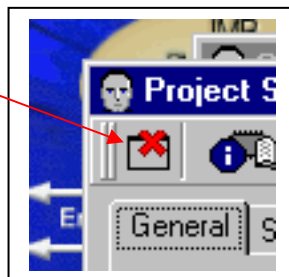
File – Project Settings

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.

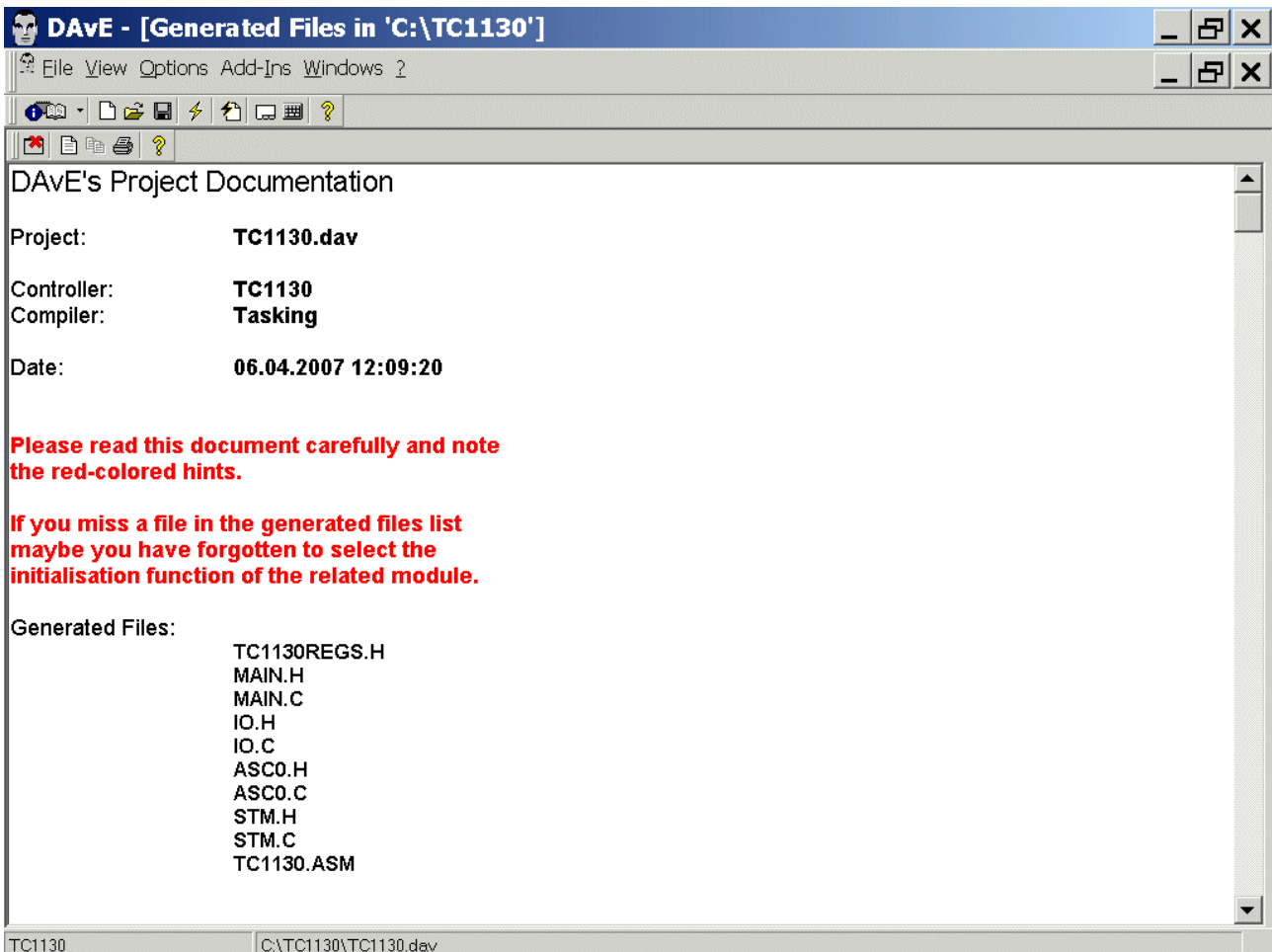


Generate Code:

<p>File Generate Code</p>	<p>or click </p>
---	---



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



File
Exit
Save changes?
click **Yes**

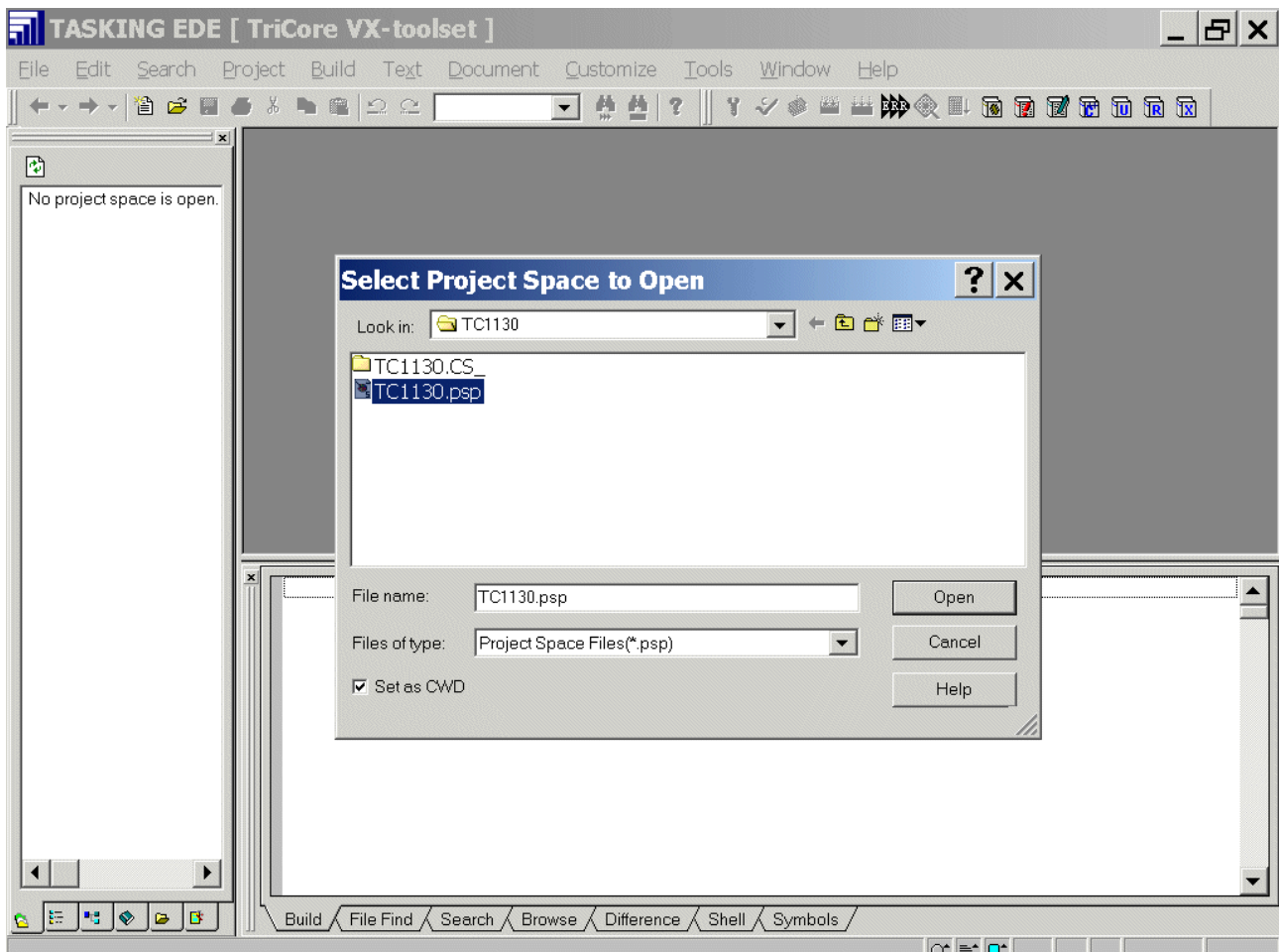


Start Tasking EDE and open the project:

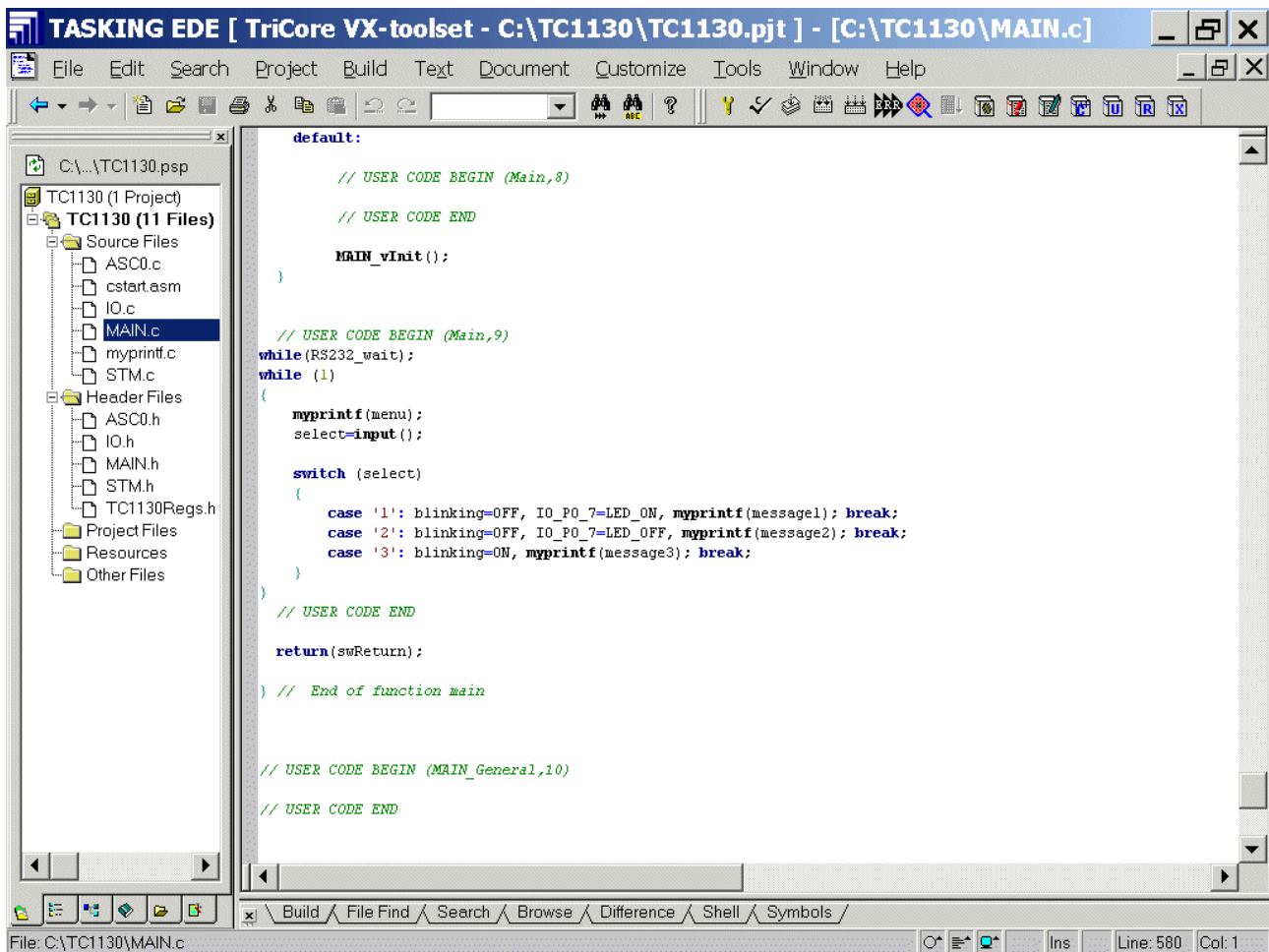
File – Open Project Space

Look in: select C:\TC1130

File name: select TC1130.psp



Open



The screenshot shows the TASKING EDE IDE interface. The title bar reads "TASKING EDE [TriCore VX-toolset - C:\TC1130\TC1130.pjt] - [C:\TC1130\MAIN.c]". The menu bar includes File, Edit, Search, Project, Build, Text, Document, Customize, Tools, Window, and Help. The toolbar contains various icons for file operations and development tools. On the left, a project tree shows the structure of the TC1130 project, including source files (ASC0.c, cstart.asm, IO.c, MAIN.c, myprintf.c, STM.c) and header files (ASC0.h, IO.h, MAIN.h, STM.h, TC1130Regs.h). The main editor window displays the following C code:

```

default:

    // USER CODE BEGIN (Main,8)

    // USER CODE END

    MAIN_vInit();
}

// USER CODE BEGIN (Main,9)
while (RS232_wait);
while (1)
{
    myprintf(menu);
    select=input();

    switch (select)
    {
        case '1': blinking=OFF, IO_PO_7=LED_ON, myprintf(message1); break;
        case '2': blinking=OFF, IO_PO_7=LED_OFF, myprintf(message2); break;
        case '3': blinking=ON, myprintf(message3); break;
    }
}

// USER CODE END

return(swReturn);
} // End of function main

// USER CODE BEGIN (MAIN_General,10)

// USER CODE END

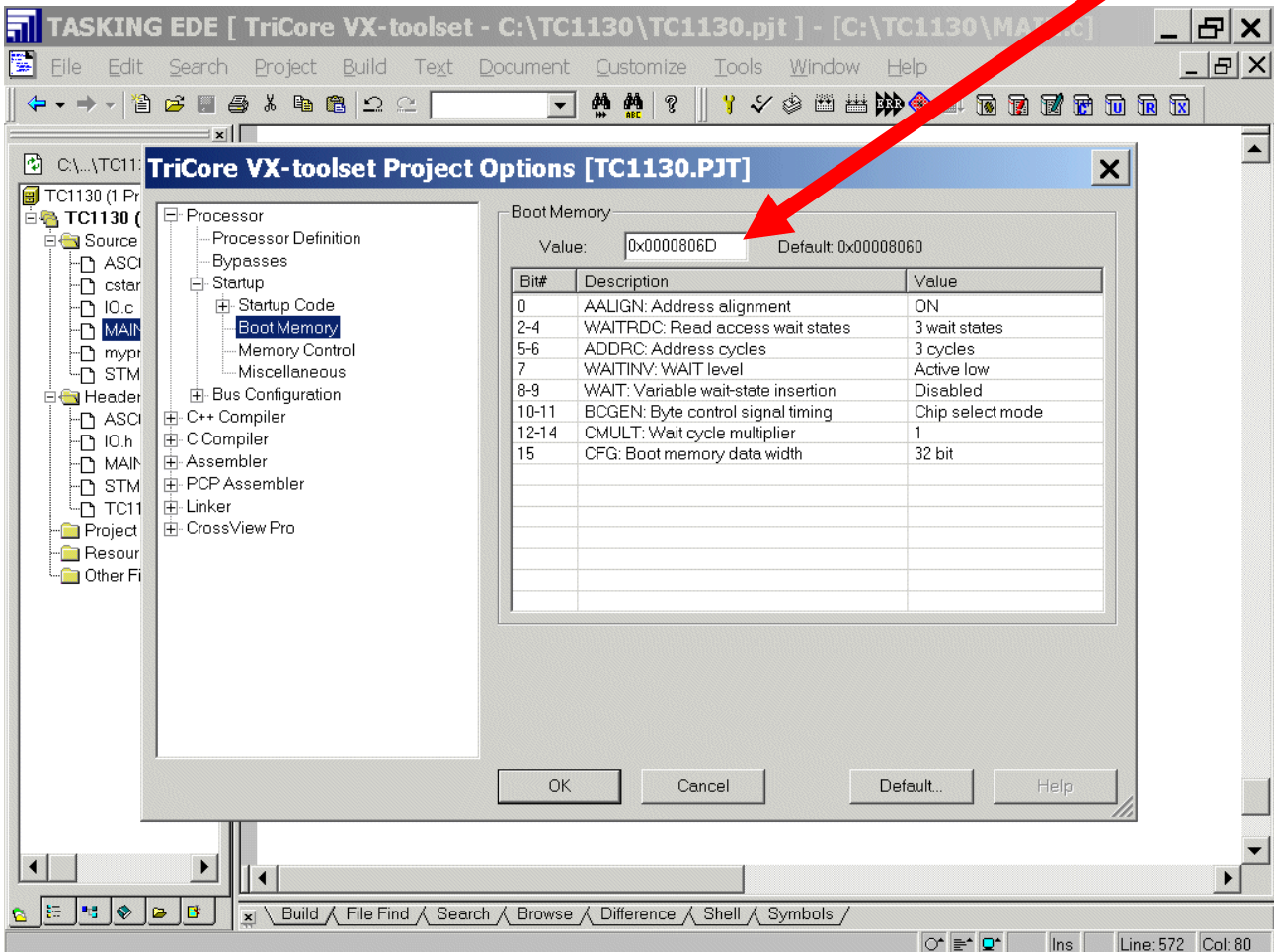
```

The status bar at the bottom indicates the current file is C:\TC1130\MAIN.c, and the cursor is at Line 580, Column 1.

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

Project – Project Options

Processor: Startup: Startup Code: BootMemory: Boot Memory: Value: insert 0x0000806D



Note:

EBU Boot Configuration Value

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16		
RES32																	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
CON F32 BIT	CMULT	BCGEN	WAIT	WAI TINV	ADDRC	WAITRDC	RES	AALI GN									



External Boot Memory Configuration Word:

If external boot is selected, the EBU will perform (exactly) one external bus read access to a specific address (0x000004 / 0x000010) of the memory device attached to CS0.

The result of this read access is used to configure the EBU.

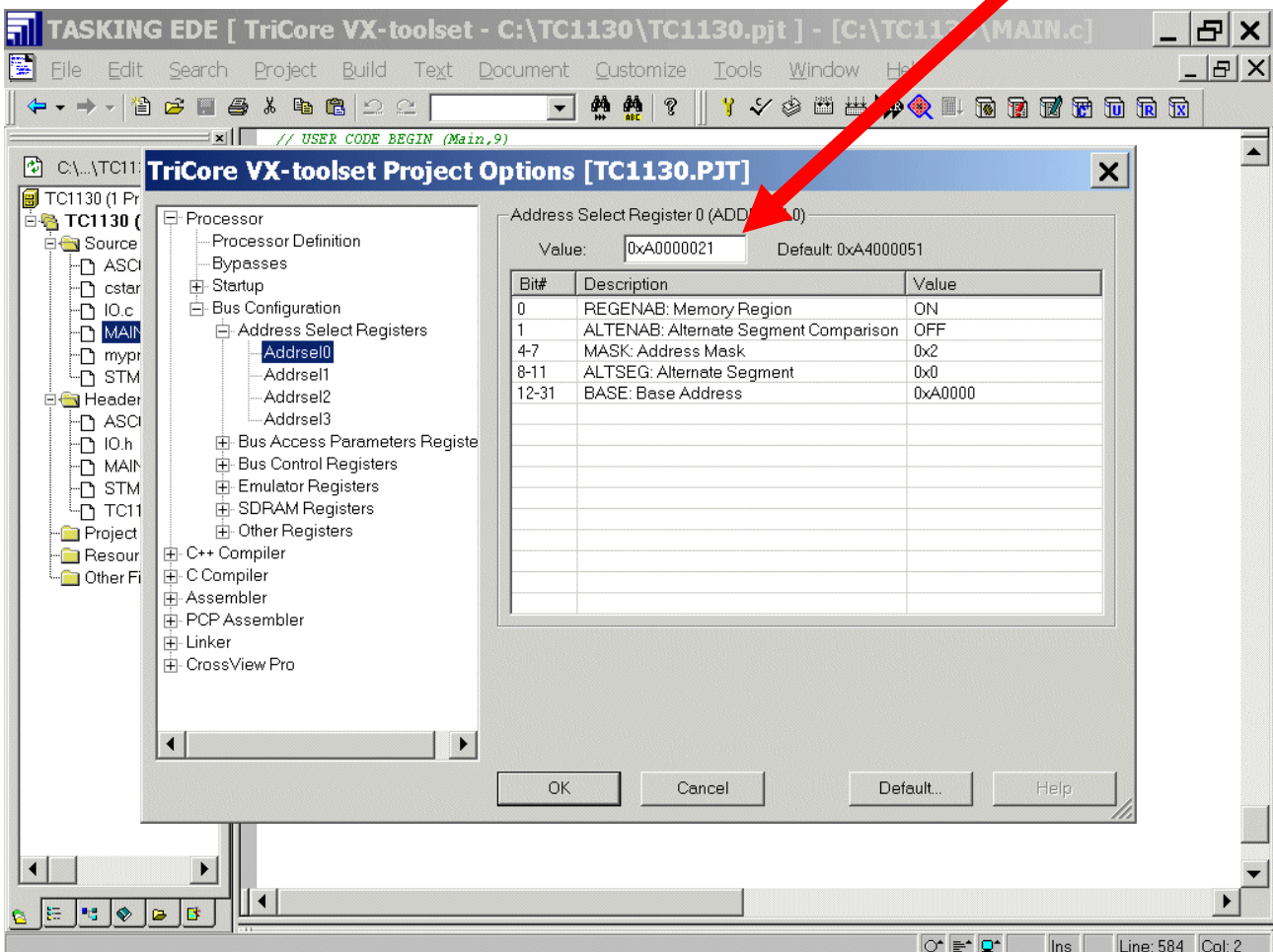
[To see “Troubleshooting” click here.](#)



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

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

Processor: Bus Configuration: Address Select Registers: AddrSel0: insert 0xA0000021



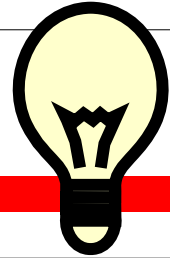


Table 14-11 EBU Address Regions, Registers, and Chip Selects

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

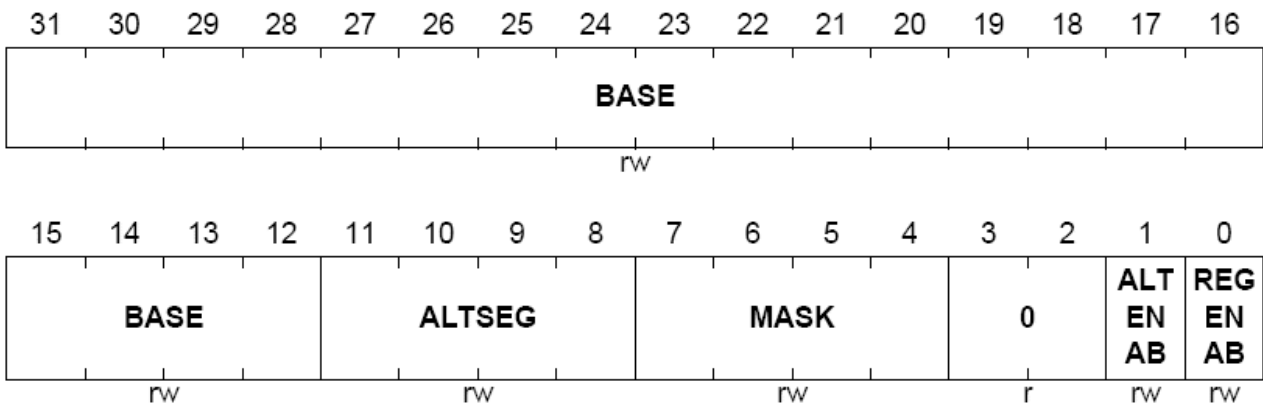
ADDRSEL[3:0]

EBU Address Select Register x

ADDRSEL[3:1] Reset Value: 0000 0000_H

ADDRSEL0 Reset Value (internal boot): 0000 0000_H

ADDRSEL0 Reset Value (external boot): A000 0001_H



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 = 100000000000000000000000000000 b

Base = A0000

Mask = 2

	1	2		= Mask
[31:27]	[26]	[25]	[24:0]	
		1	000000000000000000000000000000	= 32 MBytes OnBoardFlash



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

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

Processor: Bus Configuration: Address Select Registers: AddrSel1: insert 0xA4000011

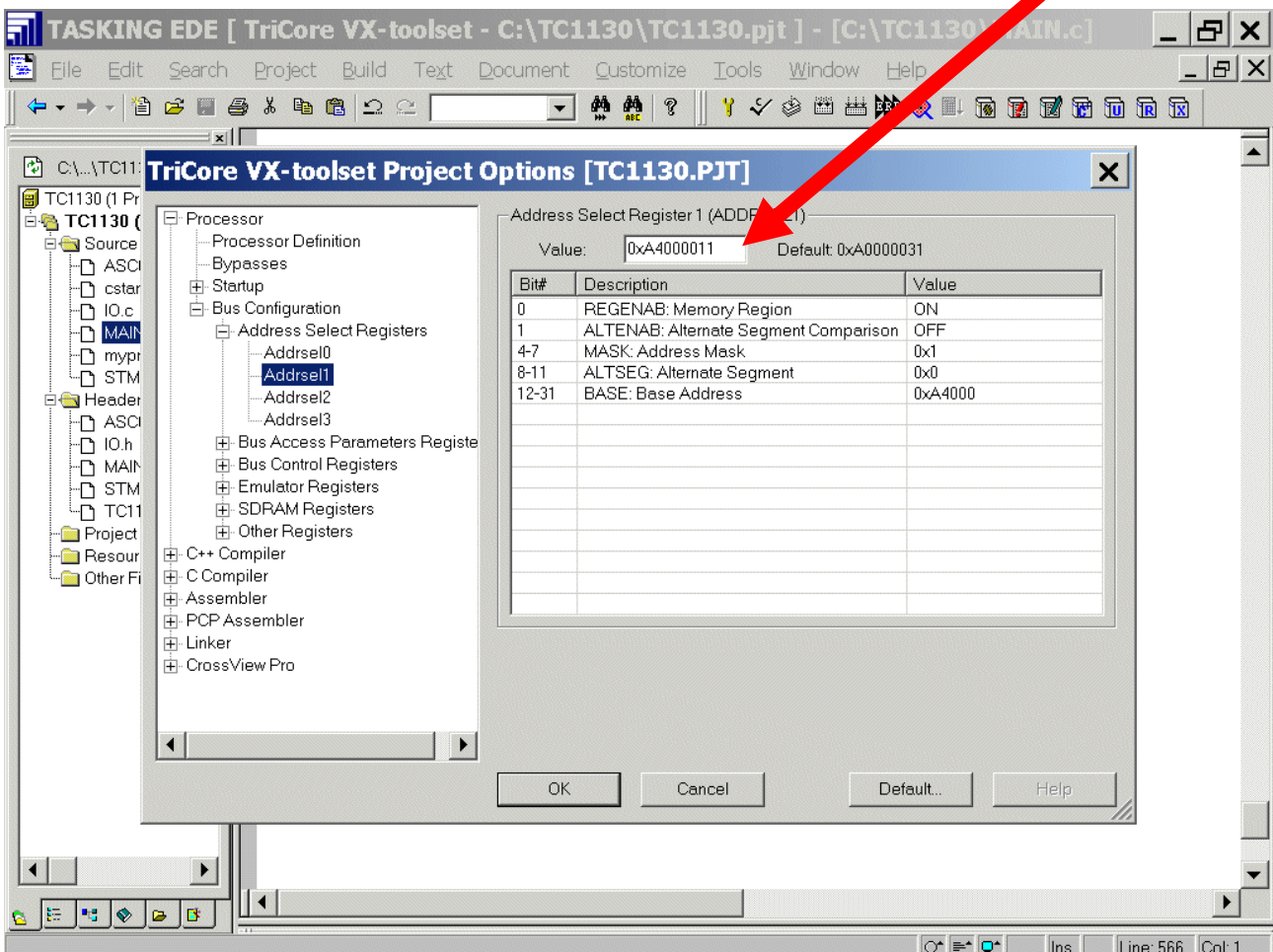




Table 14-11 EBU Address Regions, Registers, and Chip Selects

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

Note:

64 MBytes are an address-space between 0x0000.0000 and 0x0400.0000. Therefore you should use a boundary of 0x0400.0000.



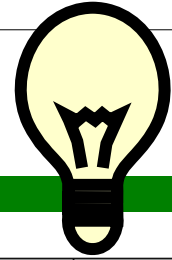


Table 14-11 EBU Address Regions, Registers, and Chip Selects

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

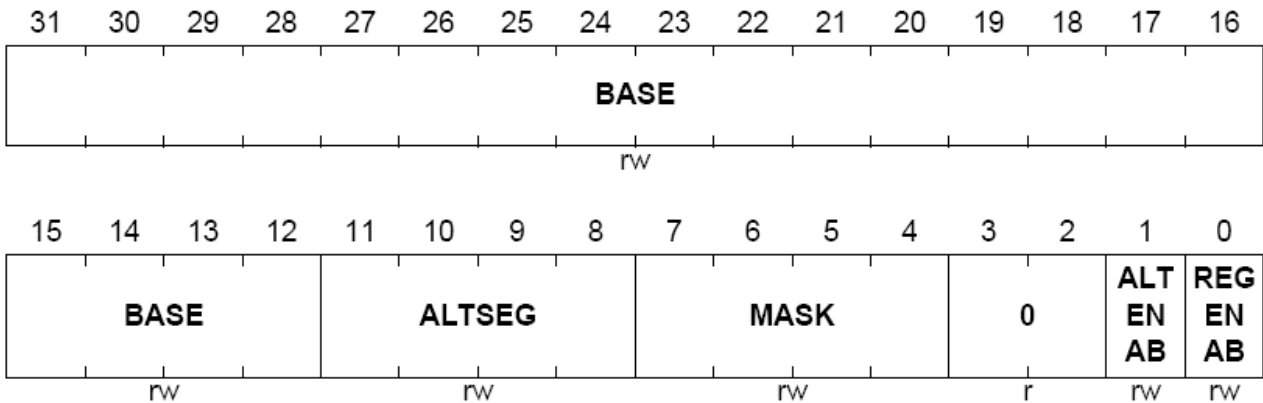
ADDRSEL[3:0]

EBU Address Select Register x

ADDRSEL[3:1] Reset Value: 0000 0000_H

ADDRSEL0 Reset Value (internal boot): 0000 0000_H

ADDRSEL0 Reset Value (external boot): A000 0001_H



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 = 10000000000000000000000000000000 b

Base = A4000

Mask = 1

	1		= Mask
[31:27]	[26]	[25:0]	
	1	00000000000000000000000000000000	= 64 MBytes OnBoardSDRAM

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

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

Note: "User region 2" = NOT USED !!!

Processor: Bus Configuration: Address Select Registers: AddrSel2: insert 0xA2000030

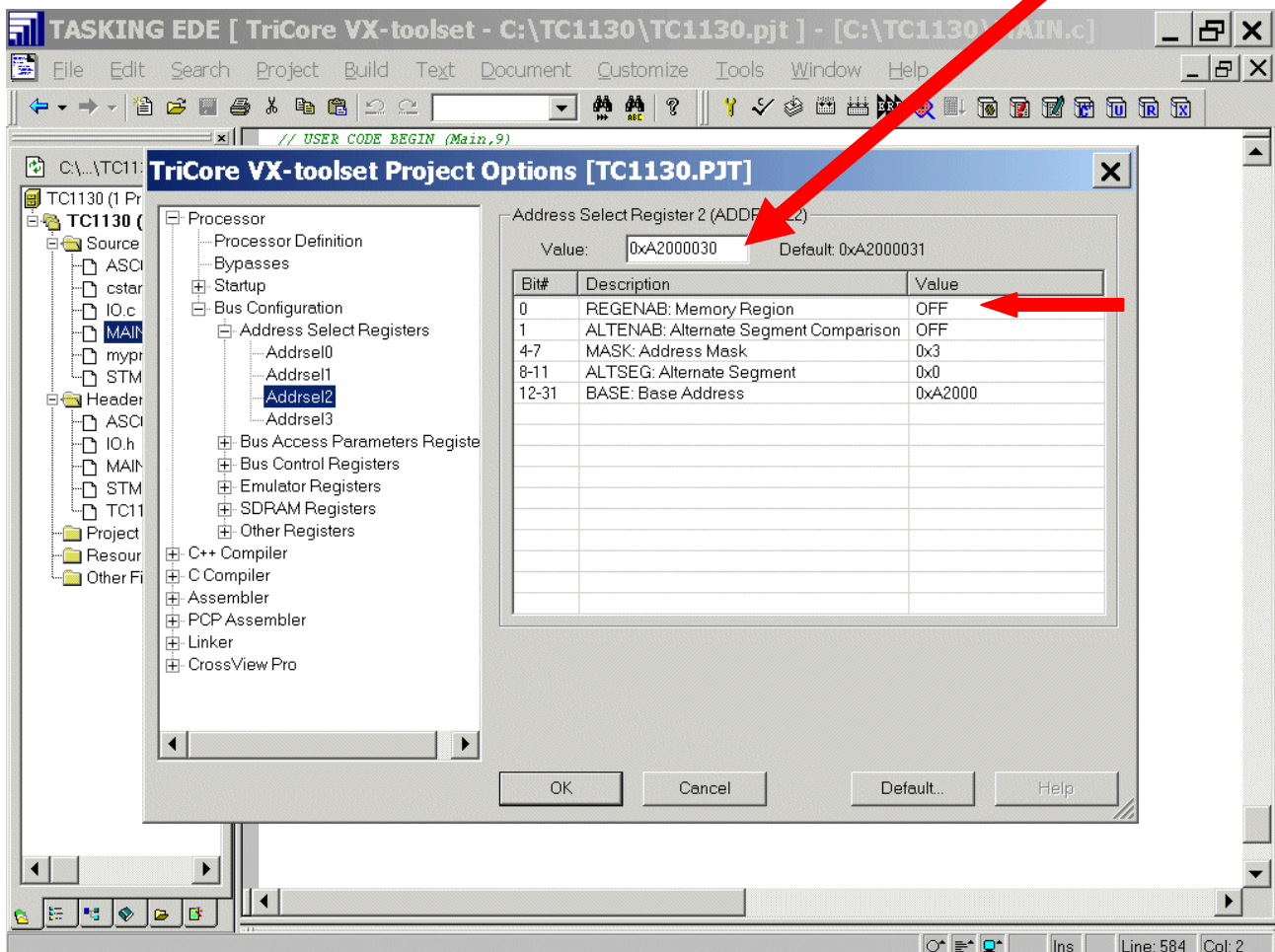




Table 14-11 EBU Address Regions, Registers, and Chip Selects

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

Processor: Bus Configuration: Bus Access Parameters Registers: Busap0: insert 0xC6DB0000

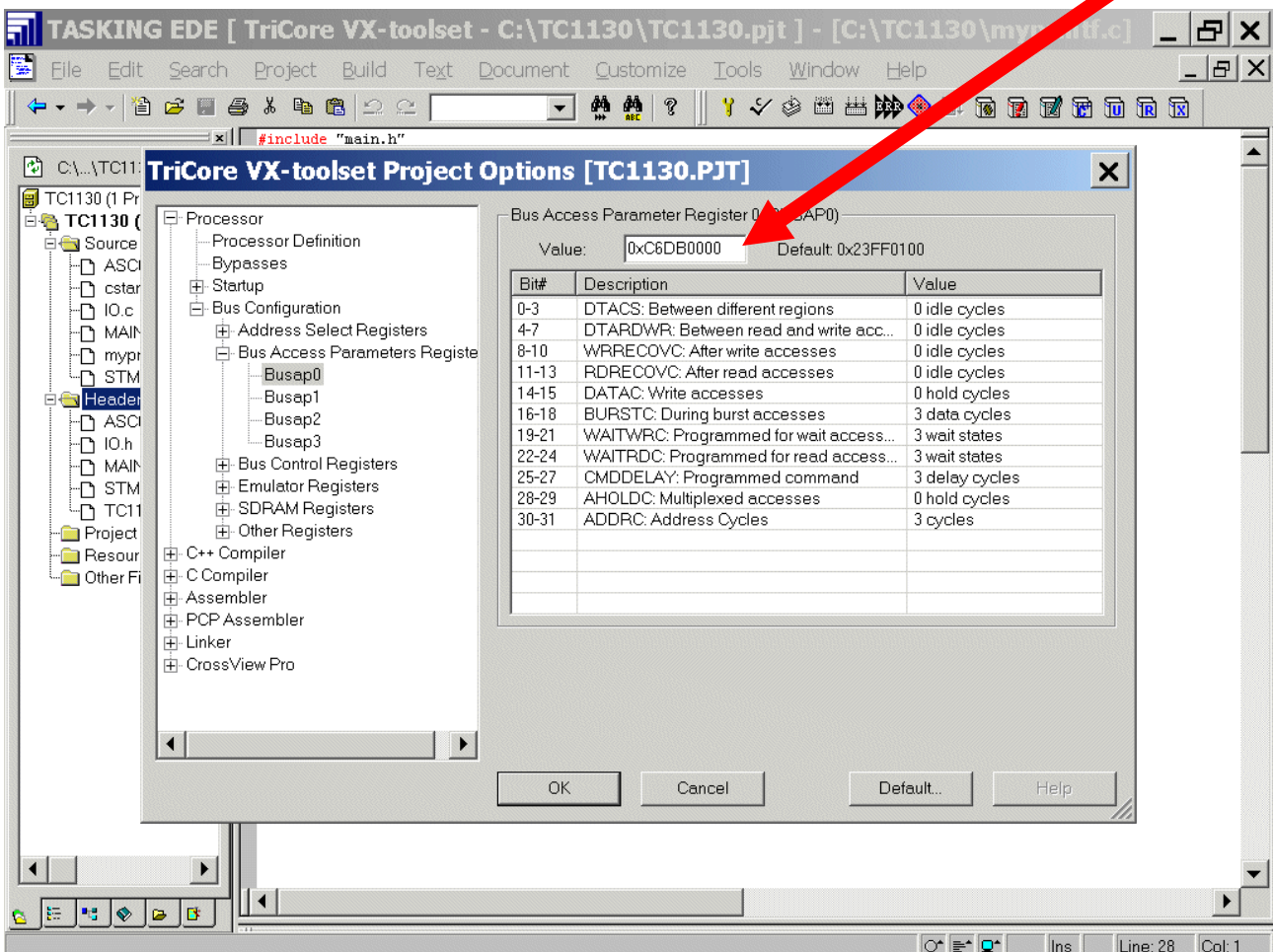
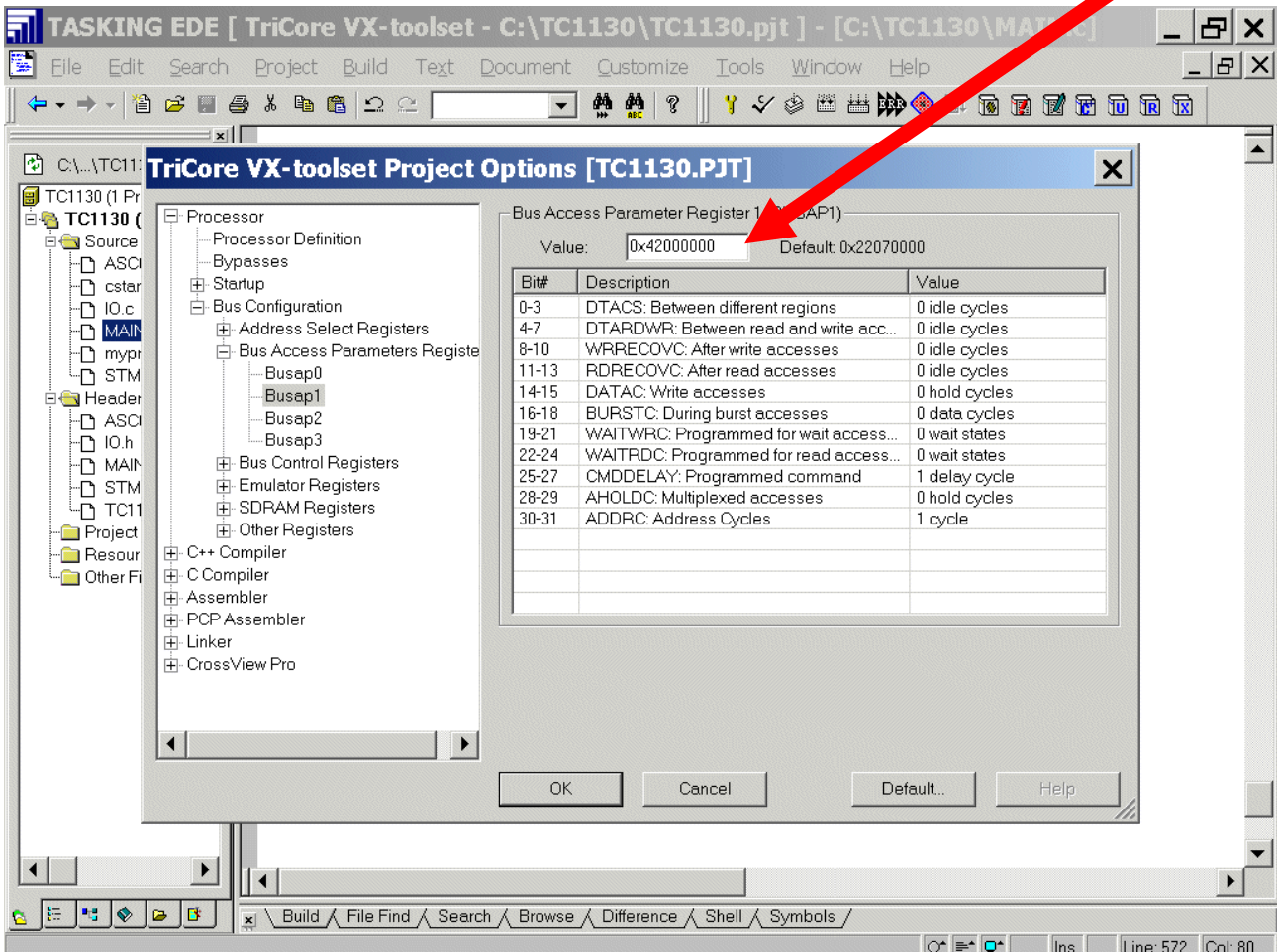




Table 14-11 EBU Address Regions, Registers, and Chip Selects

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

Processor: Bus Configuration: Bus Access Parameters Registers: Busap1: insert 0x42000000



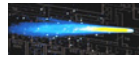


Table 14-11 EBU Address Regions, Registers, and Chip Selects

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

Processor: Bus Configuration: Bus Control Registers: Buscon0: insert 0x00922300

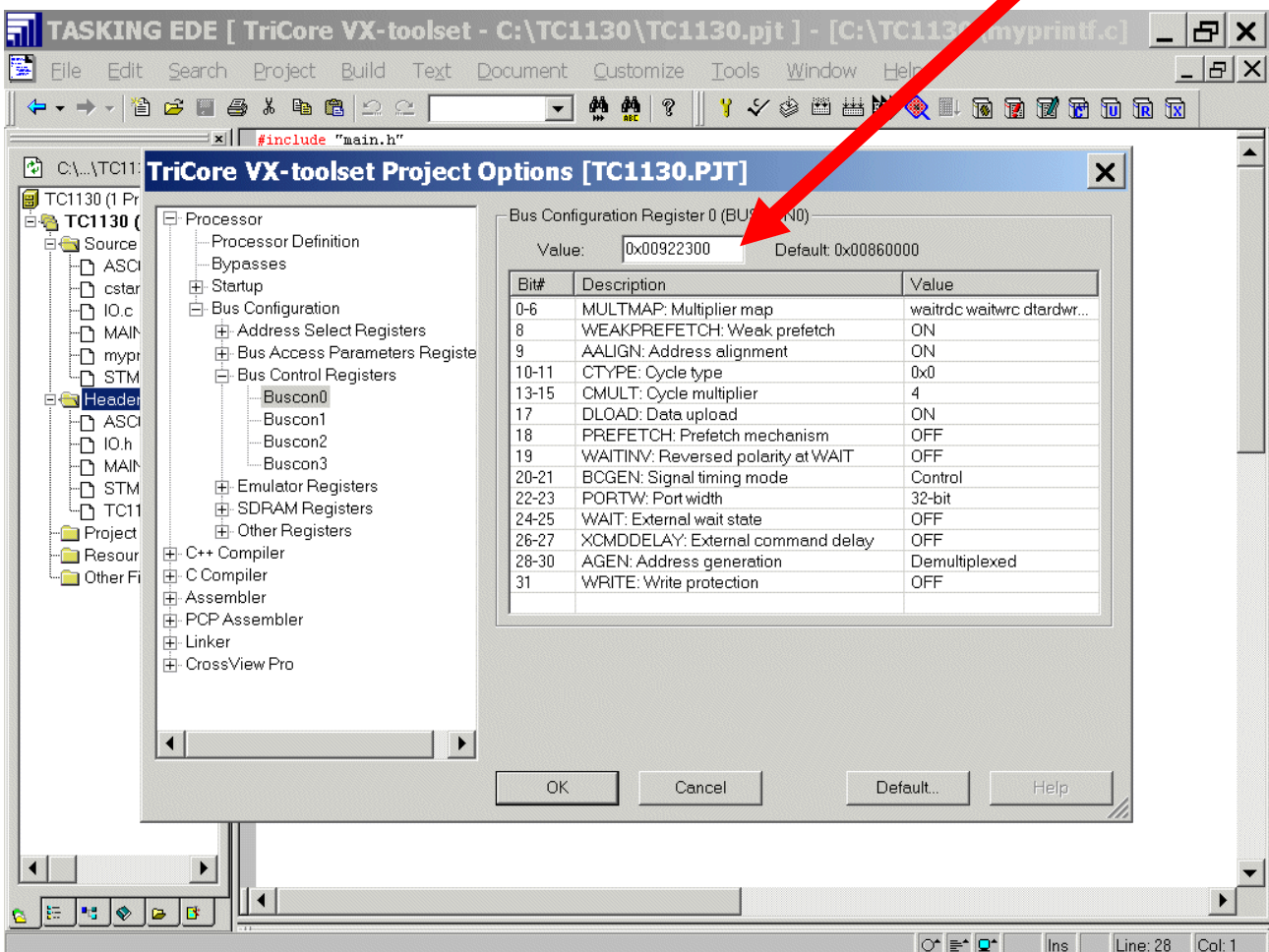
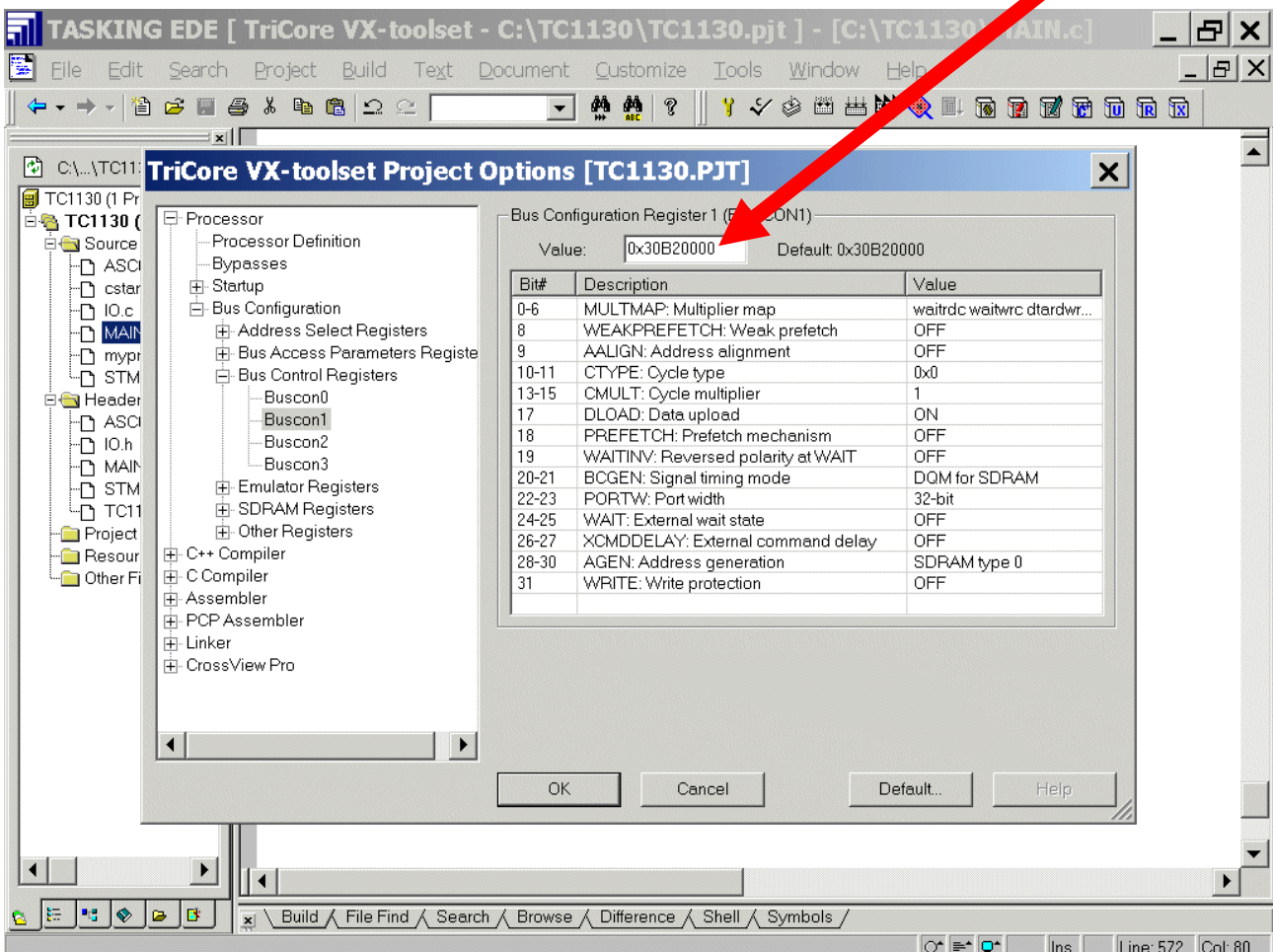




Table 14-11 EBU Address Regions, Registers, and Chip Selects

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

Processor: Bus Configuration: Bus Control Registers: Buscon1: check/insert 0x30B20000





Processor: Bus Configuration: SDRAM Registers: Sdrhref0: insert 0x000000C9

TriCore VX-toolset Project Options [TC1130.PJT]

Processor

- Processor Definition
- Bypasses
- Startup
- Bus Configuration
 - Address Select Registers
 - Bus Access Parameters Register
 - Bus Control Registers
 - Emulator Registers
 - SDRAM Registers
 - Sdrhref0
 - Sdrhref1
 - Sdrhref2
 - Sdrhref3
 - Sdrhref4
 - Sdrhref5
 - Sdrhref6
 - Sdrhref7
 - Sdrhref8
 - Sdrhref9
 - Sdrhref10
 - Sdrhref11
 - Sdrhref12
 - Sdrhref13
 - Sdrhref14
 - Sdrhref15
 - Sdrhref16
 - Sdrhref17
 - Sdrhref18
 - Sdrhref19
 - Sdrhref20
 - Sdrhref21
 - Sdrhref22
 - Sdrhref23
 - Sdrhref24
 - Sdrhref25
 - Sdrhref26
 - Sdrhref27
 - Sdrhref28
 - Sdrhref29
 - Sdrhref30
 - Sdrhref31
 - Sdrhref32
 - Sdrhref33
 - Sdrhref34
 - Sdrhref35
 - Sdrhref36
 - Sdrhref37
 - Sdrhref38
 - Sdrhref39
 - Sdrhref40
 - Sdrhref41
 - Sdrhref42
 - Sdrhref43
 - Sdrhref44
 - Sdrhref45
 - Sdrhref46
 - Sdrhref47
 - Sdrhref48
 - Sdrhref49
 - Sdrhref50
 - Sdrhref51
 - Sdrhref52
 - Sdrhref53
 - Sdrhref54
 - Sdrhref55
 - Sdrhref56
 - Sdrhref57
 - Sdrhref58
 - Sdrhref59
 - Sdrhref60
 - Sdrhref61
 - Sdrhref62
 - Sdrhref63
 - Sdrhref64
 - Sdrhref65
 - Sdrhref66
 - Sdrhref67
 - Sdrhref68
 - Sdrhref69
 - Sdrhref70
 - Sdrhref71
 - Sdrhref72
 - Sdrhref73
 - Sdrhref74
 - Sdrhref75
 - Sdrhref76
 - Sdrhref77
 - Sdrhref78
 - Sdrhref79
 - Sdrhref80
 - Sdrhref81
 - Sdrhref82
 - Sdrhref83
 - Sdrhref84
 - Sdrhref85
 - Sdrhref86
 - Sdrhref87
 - Sdrhref88
 - Sdrhref89
 - Sdrhref90
 - Sdrhref91
 - Sdrhref92
 - Sdrhref93
 - Sdrhref94
 - Sdrhref95
 - Sdrhref96
 - Sdrhref97
 - Sdrhref98
 - Sdrhref99
 - Other Registers
- C++ Compiler
- C Compiler
- Assembler
- PCP Assembler
- Linker
- CrossView Pro

SDRAM Refresh Register 0 (Sdrhref0)

Value: 0x000000C9 Default: 0x000000D7

Bit#	Description	Value
0-5	REFRESHC: Refresh counter period	0x9
6-8	REFRESHR: Number of refresh commands	4
10	SELFREX: Self refresh exit	OFF
12	SELFREN: Self refresh entry	OFF
13	AUTOSELFR: Automatic self refresh	OFF

OK Cancel Default.. Help

Build / File Find / Search / Browse / Difference / Shell / Symbols /

Ins Line: 572 Col: 80



Processor: Bus Configuration: SDRAM Registers: Sdrmref1: insert/check 0x00000000

TriCore VX-toolset Project Options [TC1130.PJT]

Processor Definition

- Bypasses
- Startup
- Bus Configuration
 - Address Select Registers
 - Bus Access Parameters Register
 - Bus Control Registers
 - Emulator Registers
 - SDRAM Registers
 - Sdrmref0
 - Sdrmref1**
 - Sdrmod0
 - Sdrmod1
 - Other Registers
- C++ Compiler
- C Compiler
- Assembler
- PCP Assembler
- Linker
- CrossView Pro

SDRAM Refresh Register 0 (SDFREF0)

Value: Default: 0x00000000

Bit#	Description	Value
0-5	REFRESHC: Refresh counter period	0x0
6-8	REFRESHR: Number of refresh comman...	1

Buttons: OK, Cancel, Default.., Help

Status Bar: Build / File Find / Search / Browse / Difference / Shell / Symbols / | Ins | Line: 572 | Col: 80



Processor: Bus Configuration: SDRAM Registers: Sdrmcon0: insert 0x219E2075

TriCore VX-toolset Project Options [TC1130.PJT]

SDRAM Control Register 0 (SDFMCON0)

Value: Default: 0x01161075

Bit#	Description	Value
0-3	CRAS: Row to precharge delay counter	6 clock cycles
4-7	CRFSH: Refresh commands counter	8 NOP cycles
8-9	CRSC: Mode register setup time	1 NOP cycle
10-11	CRP: Row precharge time counter	1 NOP cycle
12-13	AWIDTH: Width of column address	Address(10:0)
14-15	CRCD: Row to column delay counter	1 NOP cycle
16-18	CRC: Row cycle time counter	7 NOP cycles
19-21	PAGEM: Mask for page tag	Bit 26 to 11
22-24	BANKM: Mask for bank tag	Bit 26 to 25
28-31	DTALTNCY: Latency cycle control	2 cycles

Buttons: OK, Cancel, Default.., Help



Processor: Bus Configuration: SDRAM Registers: Sdrmcon1: insert/check 0x00000000

TriCore VX-toolset Project Options [TC1130.PJT]

SDRAM Control Register 1 (SDRMCON1)

Value: Default: 0x00000000

Bit#	Description	Value
0-3	CRAS: Row to precharge delay counter	1 clock cycle
4-7	CRFSH: Refresh commands counter	1 NOP cycle
8-9	CRSC: Mode register setup time	1 NOP cycle
10-11	CRP: Row precharge time counter	1 NOP cycle
12-13	AWIDTH: Width of column address	Address(8:0)
14-15	CRCD: Row to column delay counter	1 NOP cycle
16-18	CRC: Row cycle time counter	1 NOP cycle
19-21	PAGEM: Mask for page tag	Always
22-24	BANKM: Mask for bank tag	Always
28-31	DTALTNCY: Latency cycle control	0 cycles

Buttons: OK, Cancel, Default.., Help

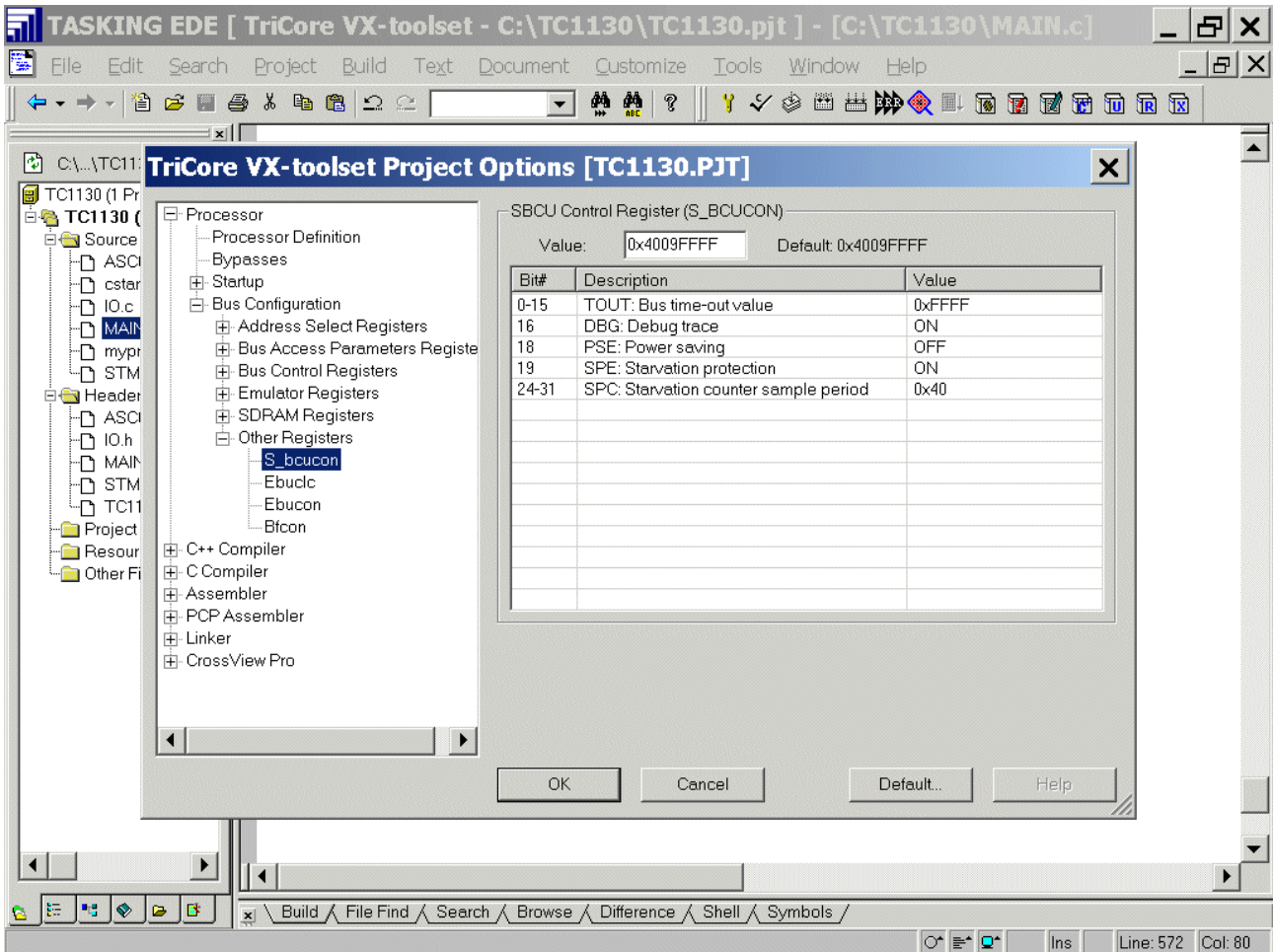


Processor: Bus Configuration: SDRAM Registers: Sdrmod0: insert/check 0x00000023

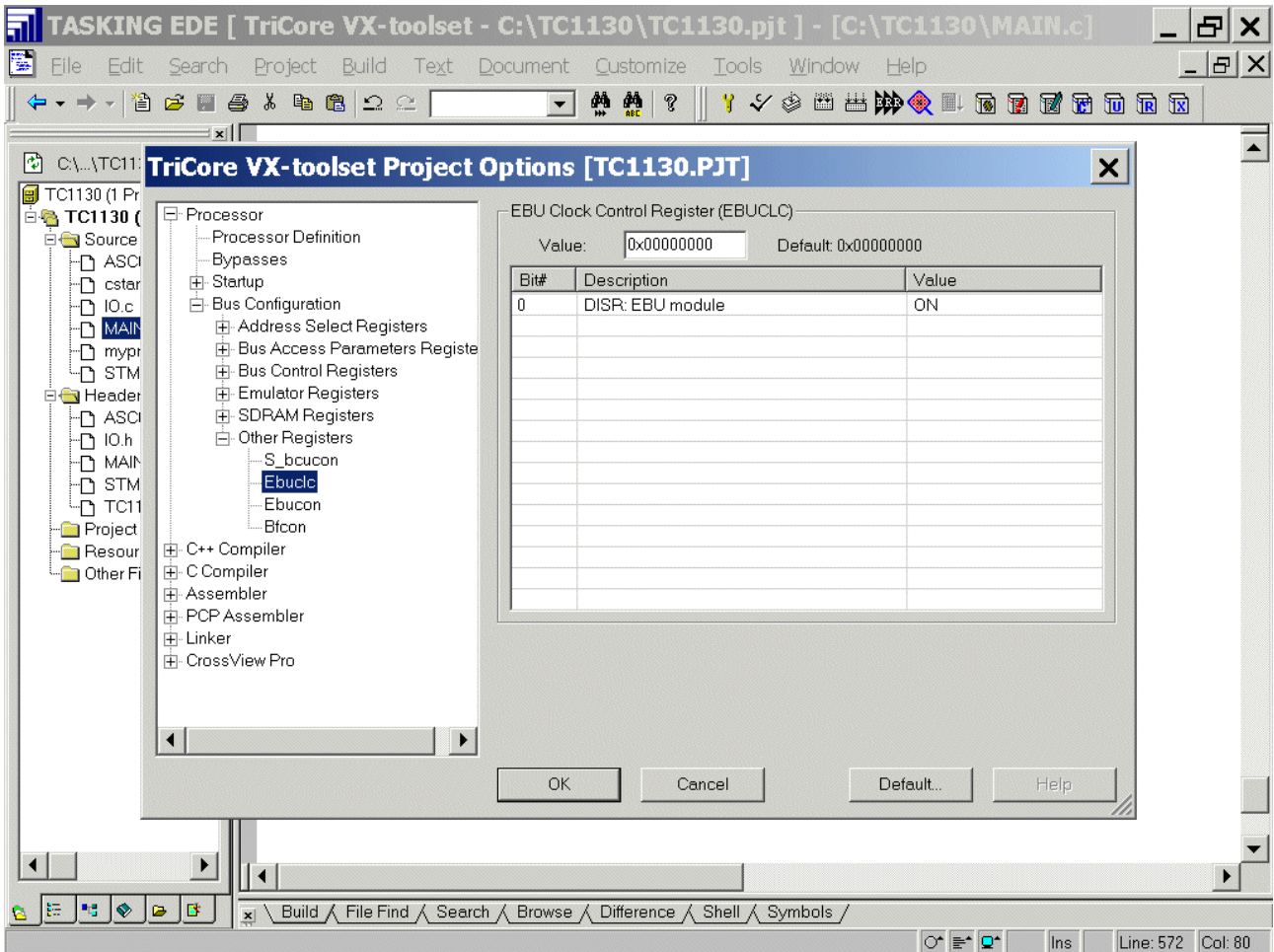
The screenshot shows the TASKING EDE IDE interface with the 'TriCore VX-toolset Project Options [TC1130.PJT]' dialog box open. The 'SDRAM Mode Register 0 (Sdrmod0)' value is set to 0x00000023. A red arrow points to the value field.

Bit#	Description	Value
0-2	BURSTL: Burst length	8
3	BTYP: Burst type	sequential burst
4-6	CASLAT: CAS latency	two clocks
7-13	OPMODE: Operation Mode	burst write

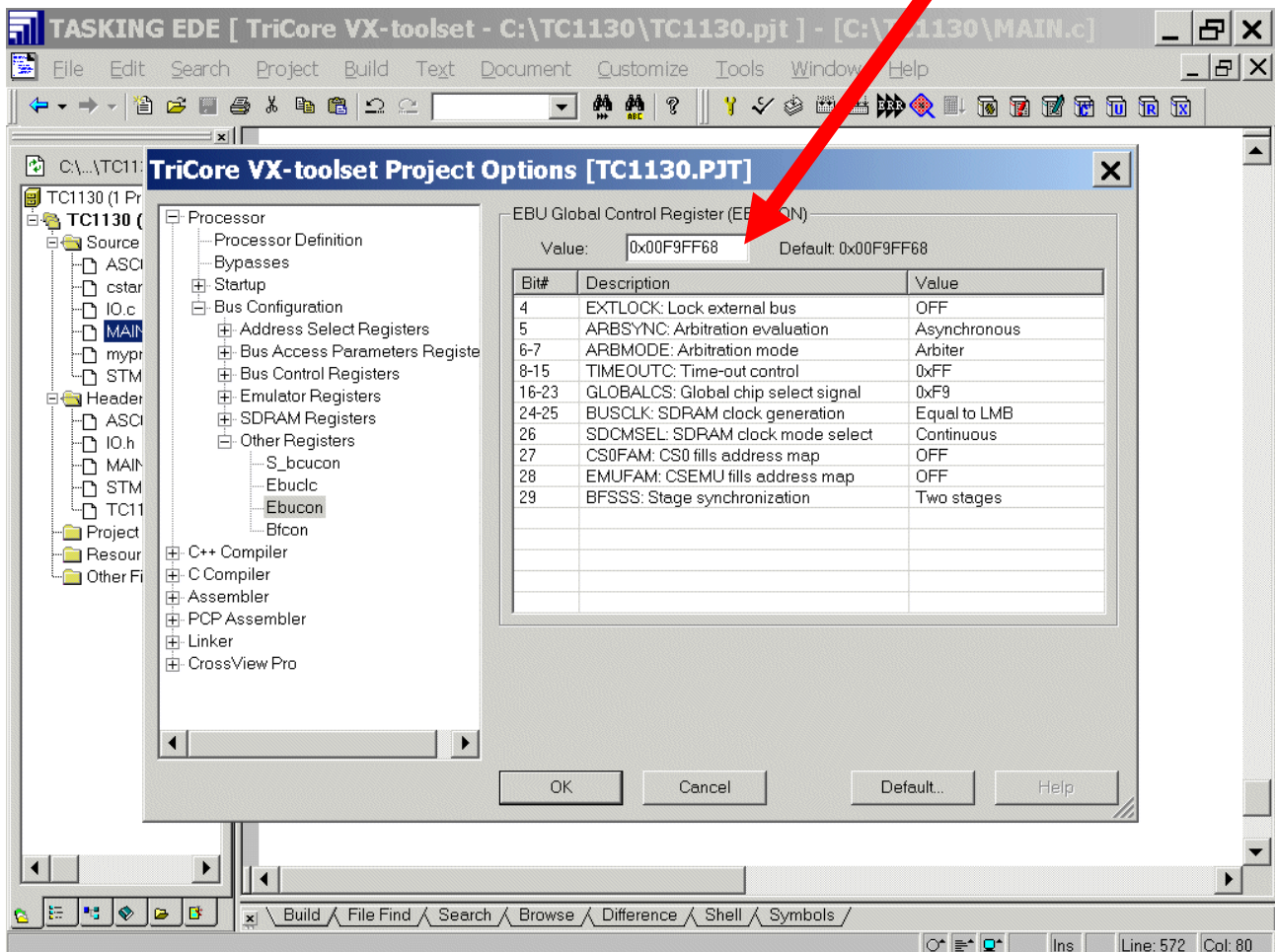
Processor: Bus Configuration: Other Registers: S_bcucon: (do nothing)



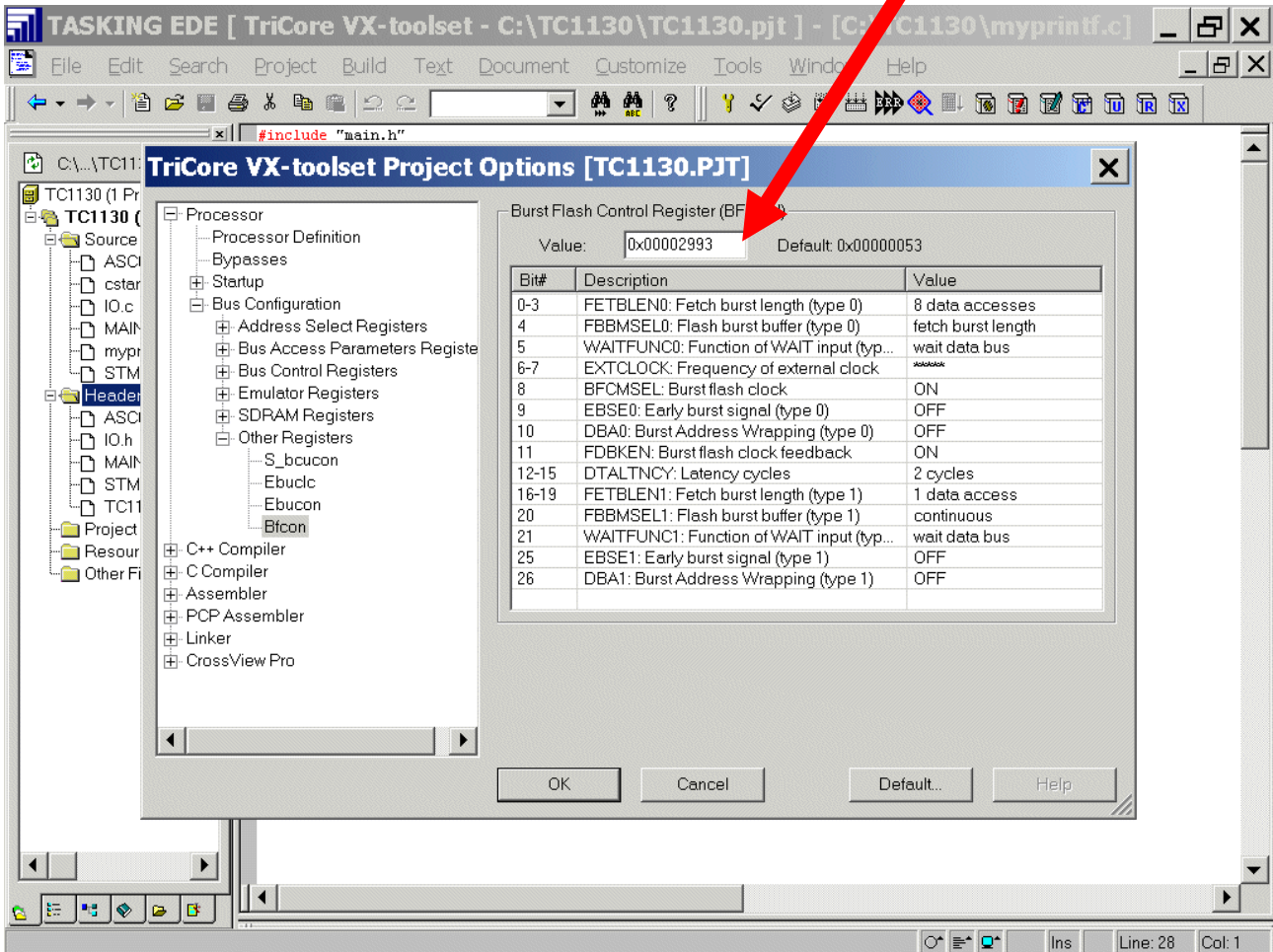
Processor: Bus Configuration: Other Registers: Ebuclc: (do nothing)



Processor: Bus Configuration: Other Registers: Ebucon: insert/check 0x00F9FF68



Processor: Bus Configuration: Other Registers: Bfcon: insert 0x00002993





Additional Information (remember):

Adobe Reader - [tc1130_um_v1.3_2004_11_sys.pdf]

File Edit View Document Tools Window Help

100%

Table 7-1 TC1130 Block Address Map

Segment	Address Range	Size	Description	DMI Acc.	PMI Acc.	
0-7	0000 0000 _H - 7FFF FFFF _H	2 GB	MMU Space	via FPI	via FPI	c a c h e d
8	8000 0000 _H - 8FFF FFFF _H	256 MB	External Memory Space mapped from Segment 10	via LMB	via LMB	
9	9000 0000 _H - 9FDF FFFF _H	256 MB	Reserved	via FPI	via FPI	
	A000 0000 _H - AFBF FFFF _H	252 MB	External Memory Space	via LMB	via LMB	n o n - c a c h e d
	AFC0 0000 _H - AFC0 FFFF _H	64 KB	DMU Space			
	AFC1 0000 _H - AFFF FFFF _H	≈ 4 MB	Reserved			
11	B000 0000 _H - BFFF FFFF _H	256 MB	Reserved	via FPI	via FPI	h e d

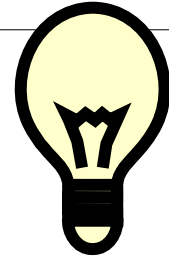
8,15 x 11,58 in

242 of 979

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



Additional Information:

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:

In 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 6Drrrrrr 2 30 276 call sprintf
```

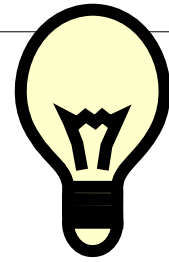
Since the linker says "patch error" it means that the call to `sprintf` is too big to fit expression '`rrrrrr`'. To make this work either the call must be changed or the libraries must be located closer to the application code. Try locating the libraries at 0xA1100000 from the same PAGE where you located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

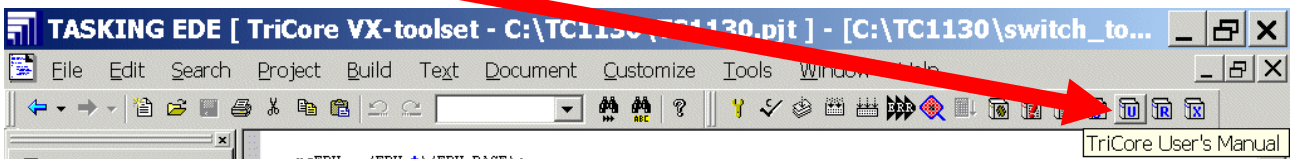
The `__indirect` keyword can be used to force the less efficient indirect call [the target address may be any 32 bit address within the whole memory space (4 GByte)].



Additional Information:

See also the Online-Manual:

Click [TriCore User's Manual](#)



TriCore User's Manual
_ □ ×

File Edit Bookmark Chapters Options Help

Contents Index Back Print

CHAPTER 3: TriCore C Language

3.9.3 FUNCTION CALLING MODES: __indirect

Functions are default called with a single word direct call. However, when you link the application and the target address appears to be out of reach (+/- 16 MB from the `callg` or `jpg` instruction), the linker generates an error. In this case you can use the `__indirect` keyword to force the less efficient, two and a half word indirect call to the function:

```

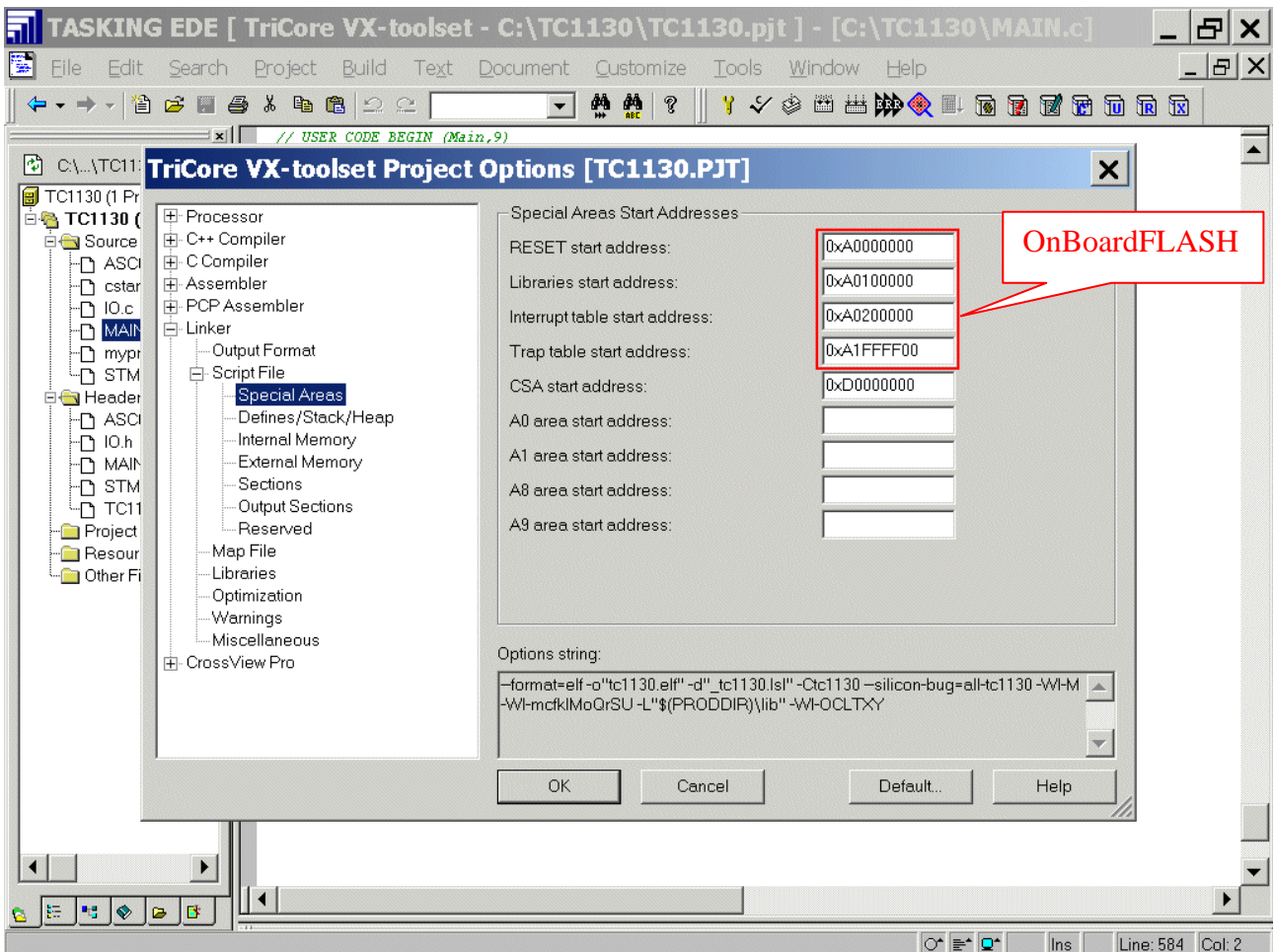
int __indirect foo( void )
{
...
}

```

With [compiler option `--indirect`](#) you tell the compiler to generate far calls for *all* functions.



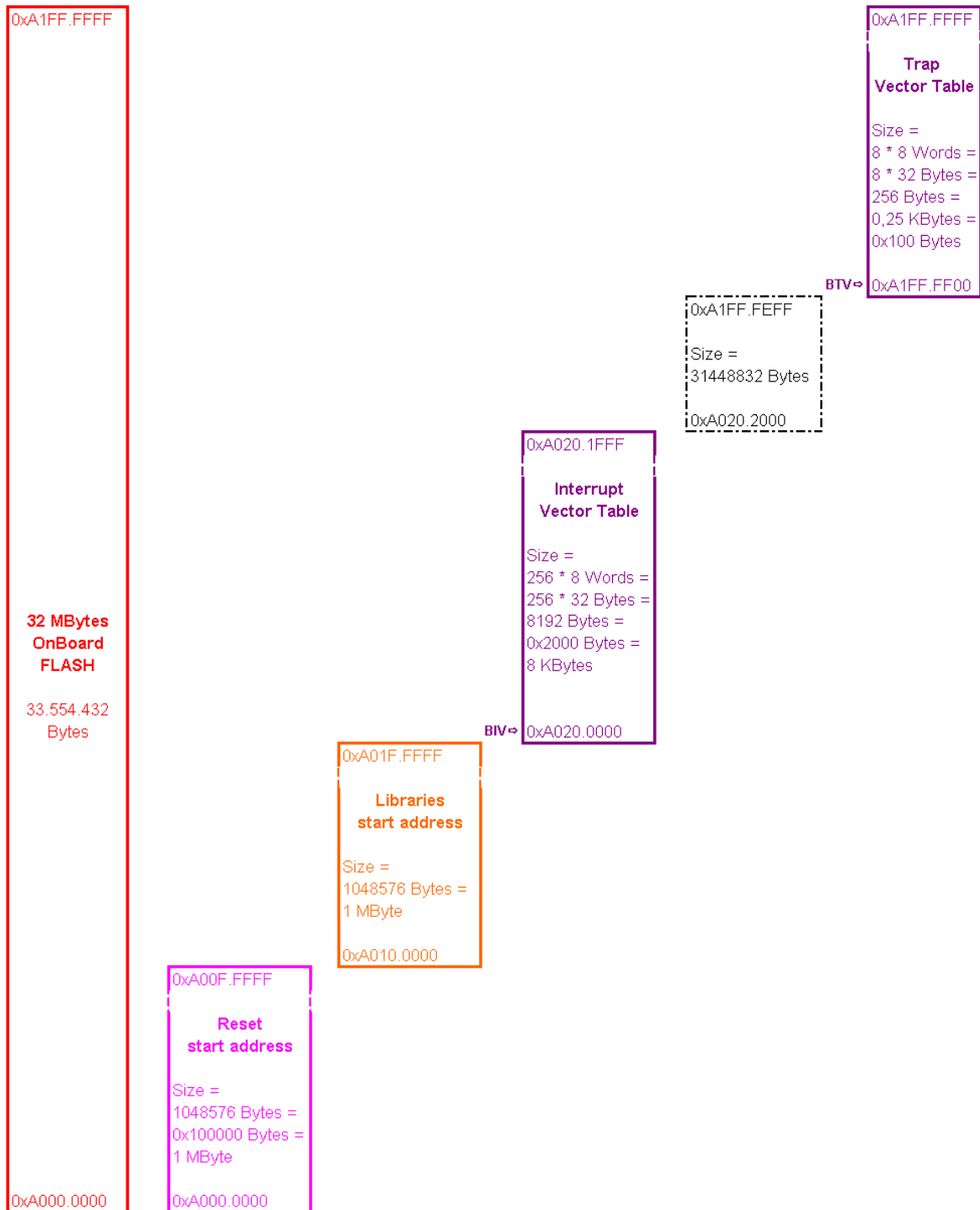
Linker: Script File: Special Areas: RESET start address: insert 0xA0000000 (OnBoardFLASH)
 Linker: Script File: Special Areas: Libraries start address: insert 0xA0100000 (OnBoardFLASH)
 Linker: Script File: Special Areas: Interrupt table start address: insert 0xA0200000 (OnBoardFLASH)
 Linker: Script File: Special Areas: Trap table start address: insert 0xA1FFFF00 (OnBoardFLASH)





Additional Information:

Memory-Map OnBoard-Flash:





Additional Information:

Interrupt-Vector-Table:

DAvE:

DAvE

TC1130

Interrupt System (INT)

Service Request Nodes | **Interrupts** | Functions | Parameters | Notes

	CPU Interrupt (max:255)	Level 0 (non interrupting)
Level 16		
Level 15		
Level 14		
Level 13		
Level 12		
Level 11		
Level 10		
Level 9	STM SRN 0	
Level 8		
Level 7		
Level 6		
Level 5		
Level 4		
Level 3		
Level 2		
Level 1		

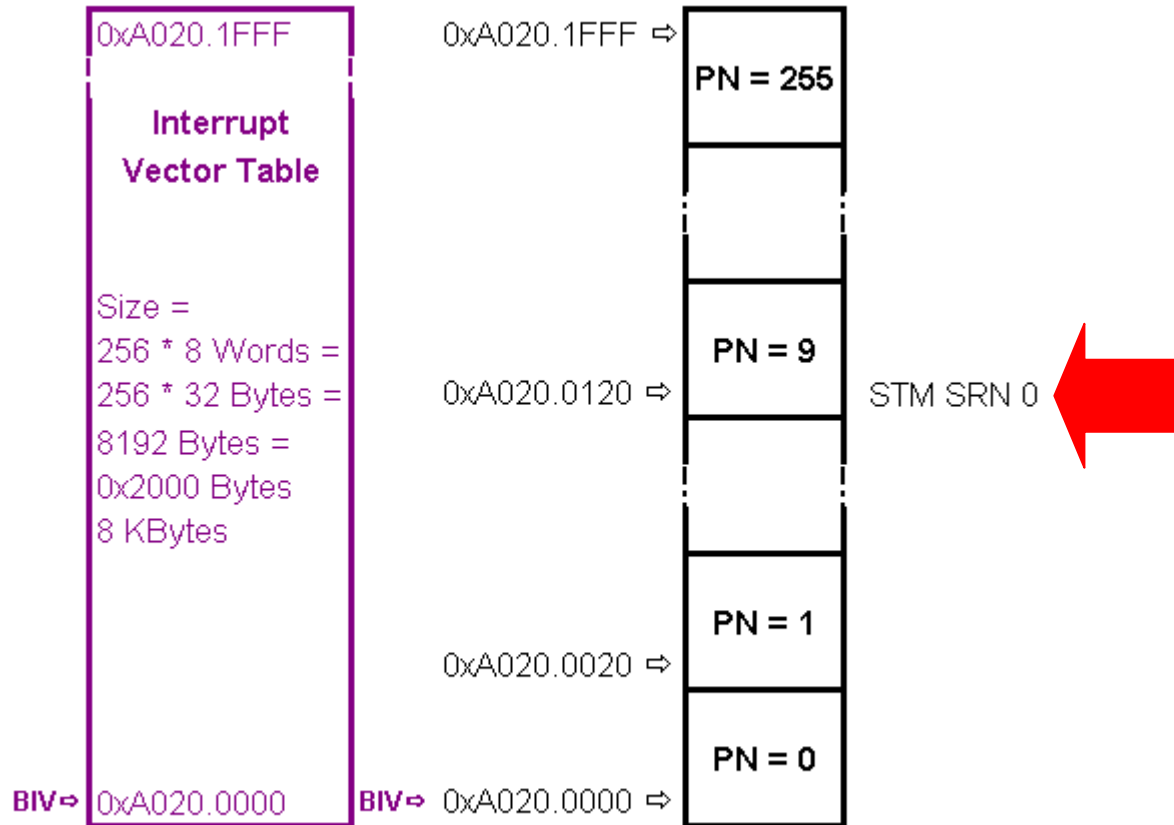
Note: To change the level and the group of an interrupt source, click on it, drag it to its new position and drop it. To set an interrupt source to the non interrupting level (Level 0) click on it, drag it to the 'Level 0' list and drop it.

TC1130 | C:\TC1130\TC1130.dav



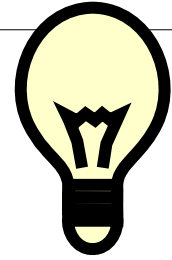
Additional Information:

Interrupt-Vector-Table:



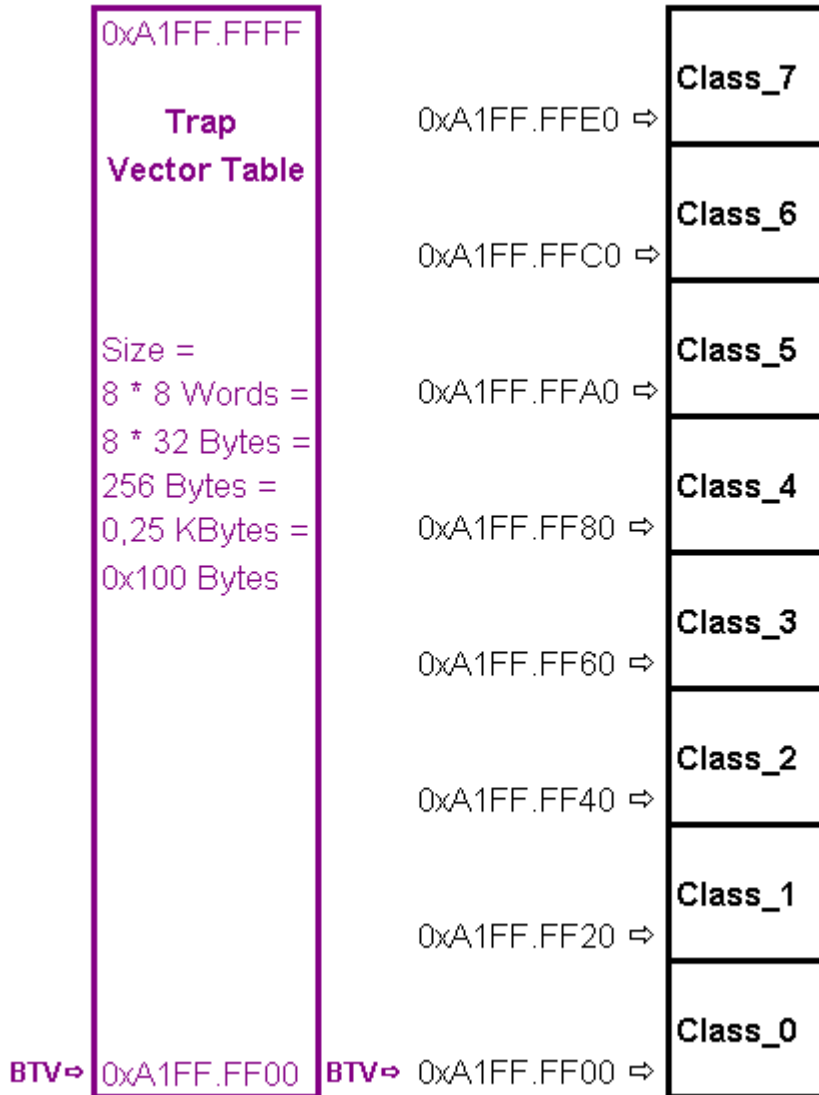
Note:
PN ... Priority Number

[Click here to see the Map File](#)



Additional Information:

TRAP-Vector-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.

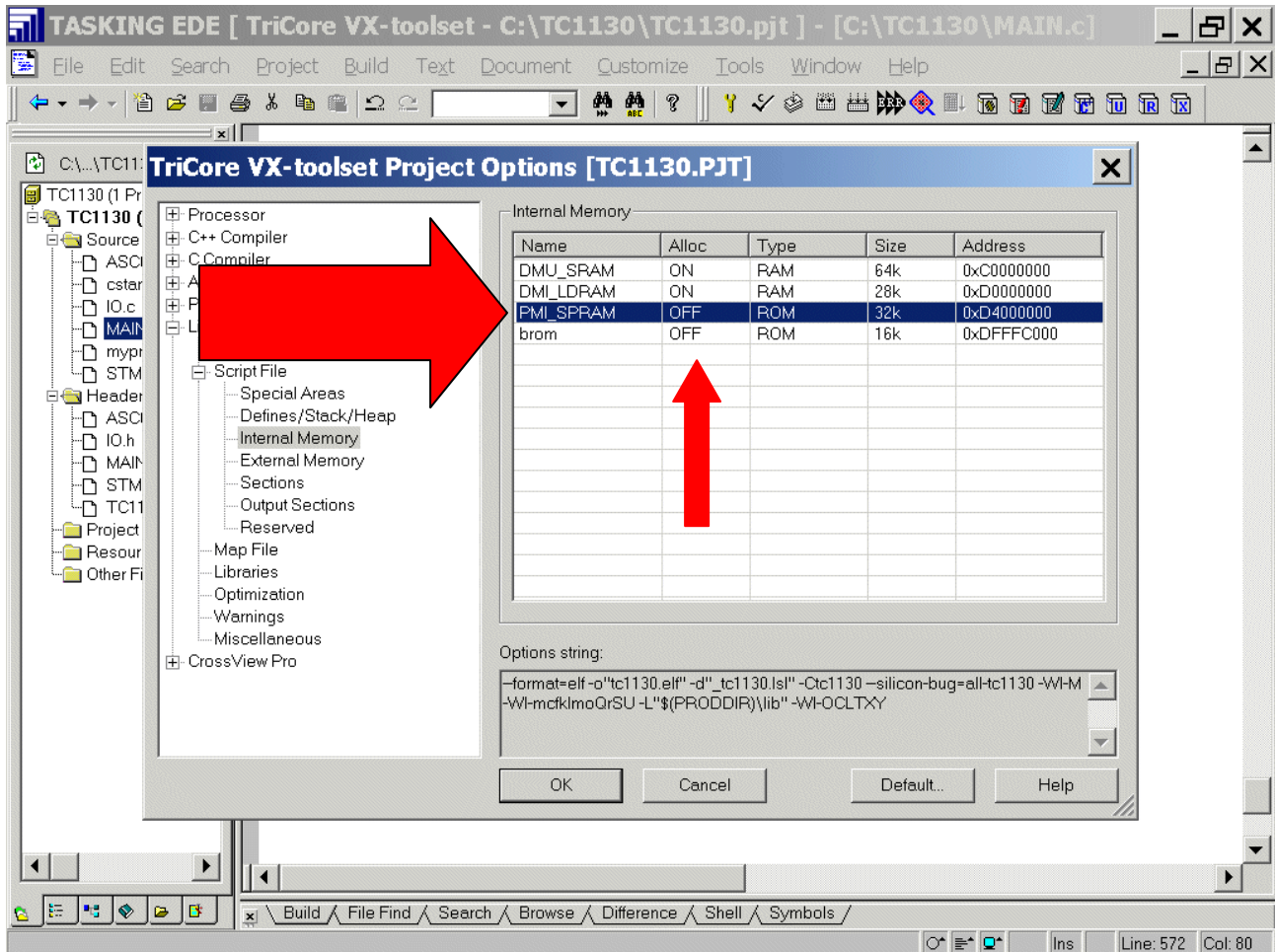
$$\text{BTV} = 0xA020.0000 = 2.686.451.712 / 8.192 = 327.936 \quad \checkmark$$

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).

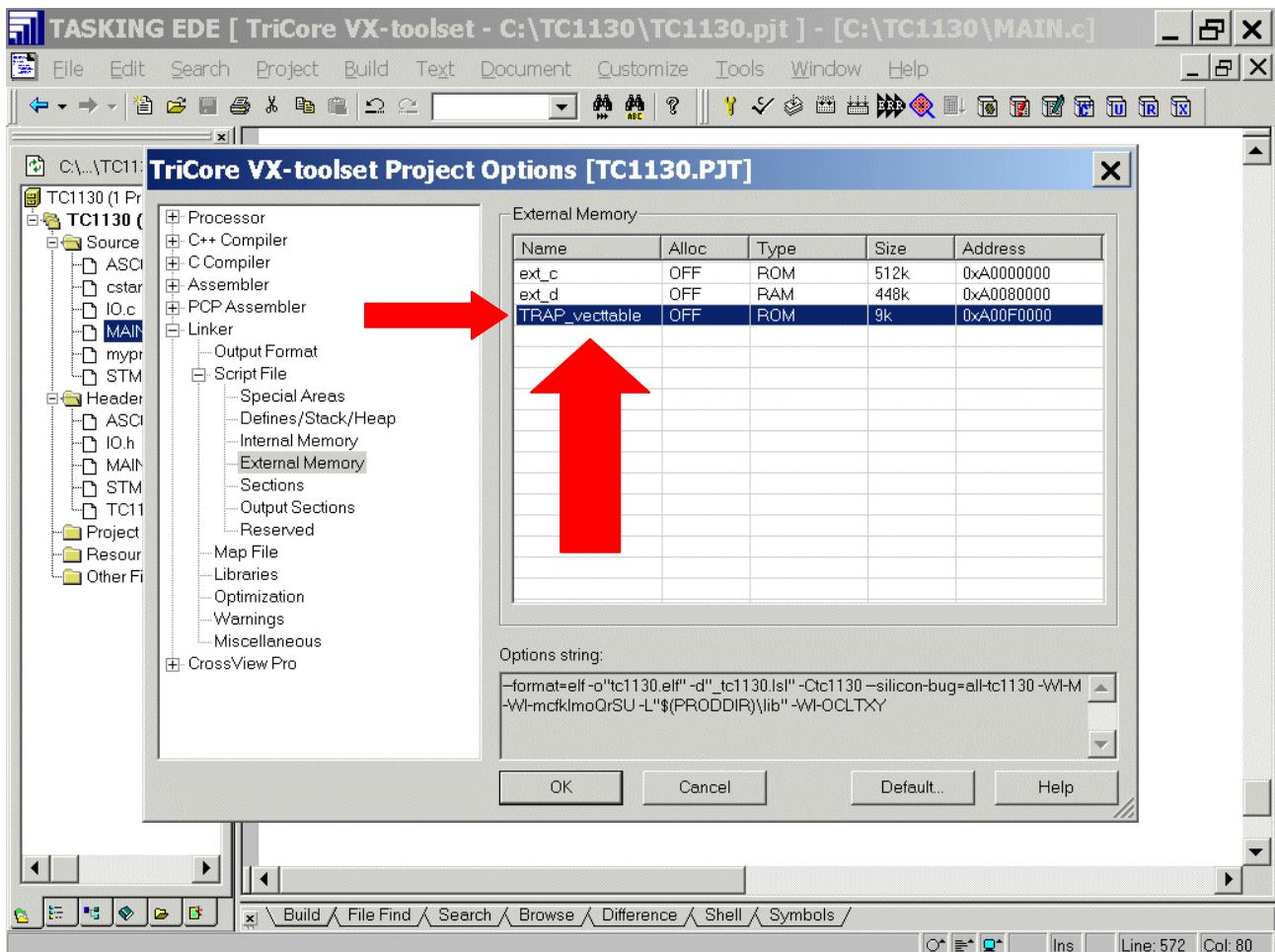
$$\text{BIV} = 0xA1FF.FF00 = 2.717.908.736 / 256 = 10.616.831 \quad \checkmark$$

Linker: Script File: Internal Memory: PMI_SPRAM: Alloc: **select OFF**

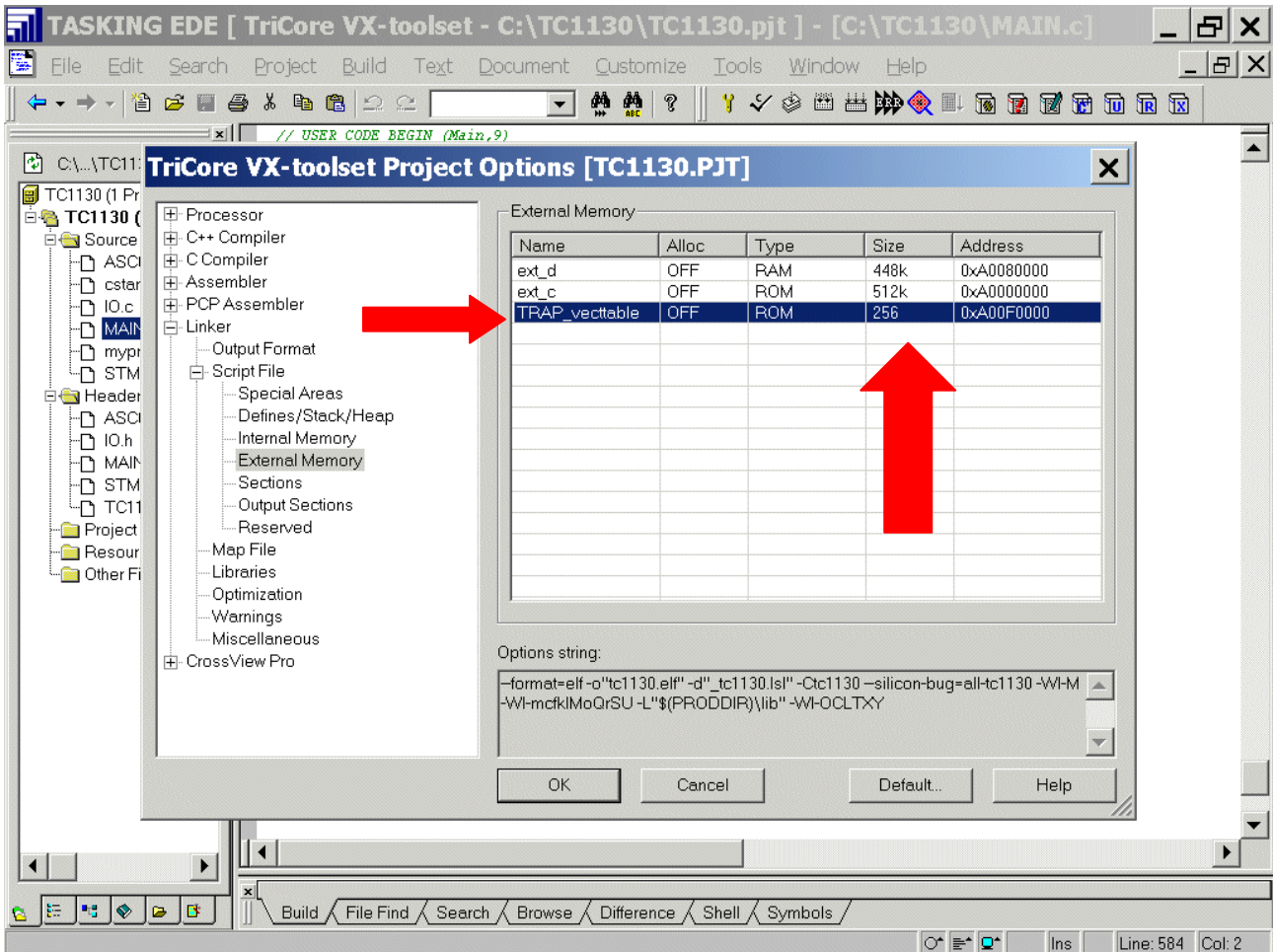


(Configuration of this dialog-window is now finished)

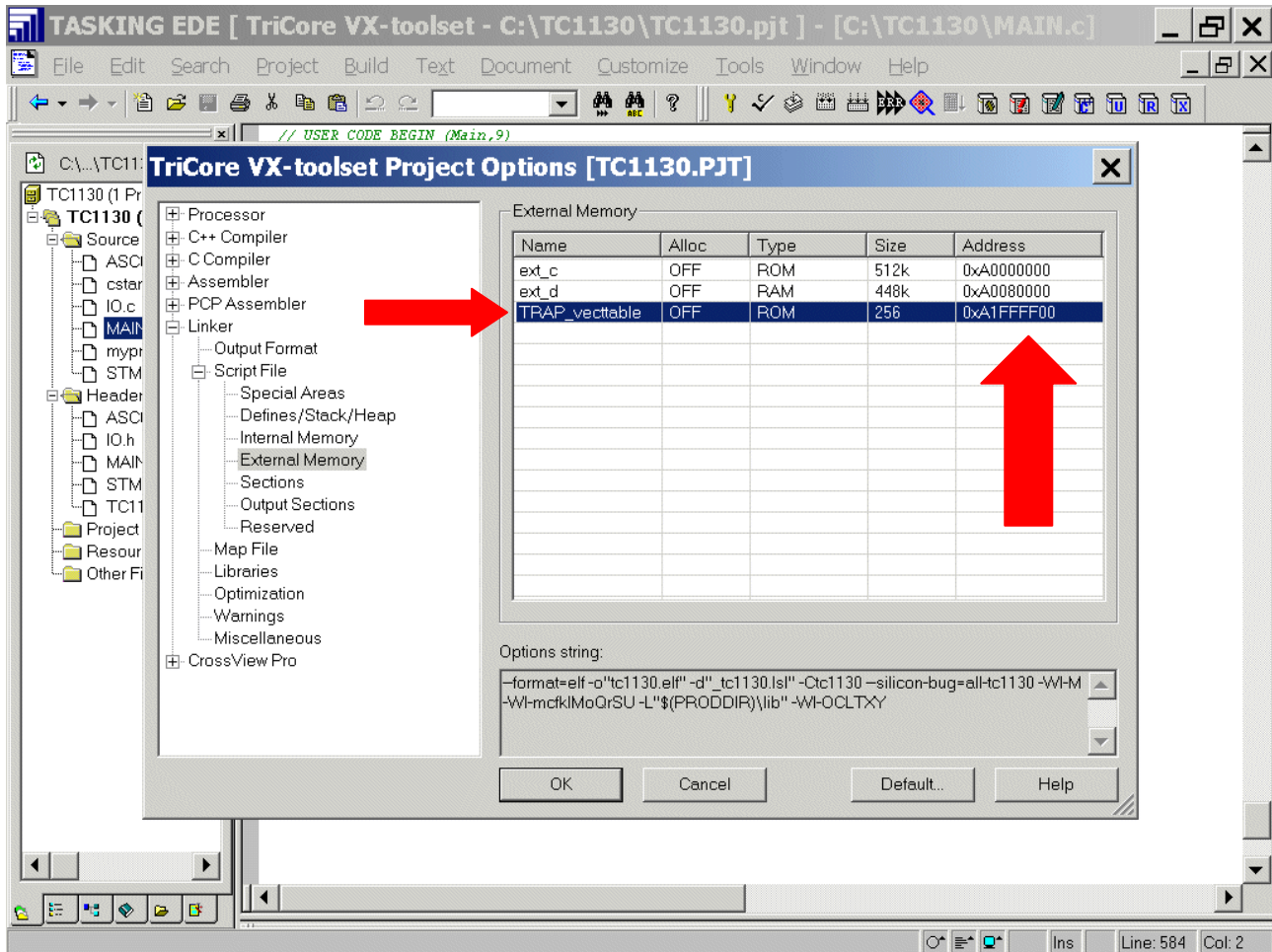
Linker: Script File: External Memory: Name: "vecttable" change to "TRAP_ vecttable"



Linker: Script File: External Memory: Size (TRAP_vectable): change to 256



Linker: Script File: External Memory: Address (TRAP_vecttable): change to 0xA1FFFF00



Note:

32 Mbytes OnBoardFlash
from A000.0000 to A1FF.FFFF – 256 Bytes “TRAP_vecttable” =
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"



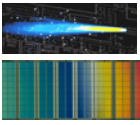
Linker: Script File: External Memory: Name=TRAP_vecttable: Alloc: select "ON"

TriCore VX-toolset Project Options [TC1130.PJT]

External Memory

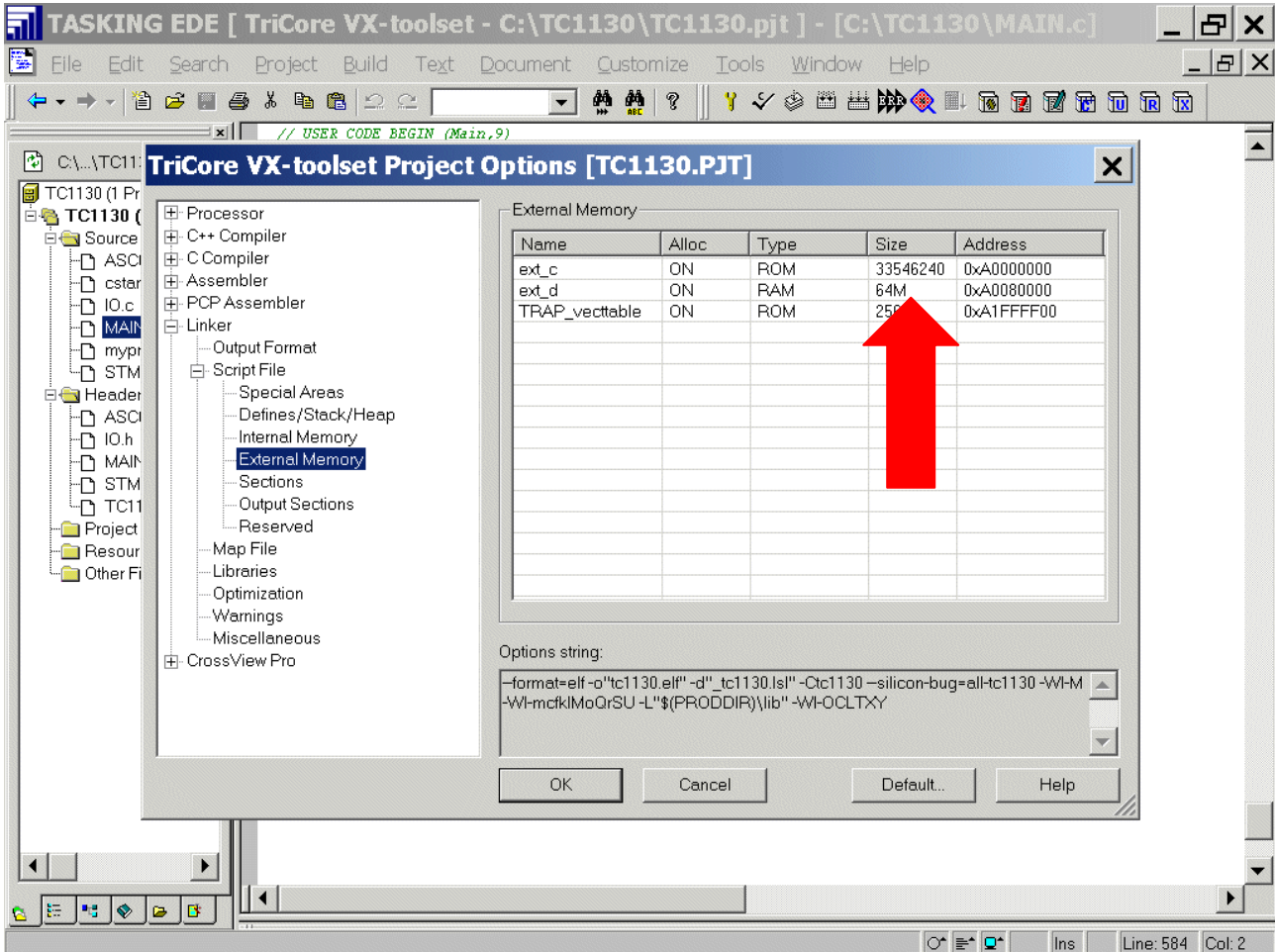
Name	Alloc	Type	Size	Address
ext_c	ON	ROM	512k	0xA0000000
ext_d	ON	RAM	448k	0xA0080000
TRAP_vecttable	ON	ROM	256	0xA1FFFF00

Options string:
 -format=elf-o"tc1130.elf"-d"tc1130.isl"-Ctc1130-silicon-bug=alHc1130-WM
 -W-mcflMoQrSU-L"\$(PRODDIR)\lib"-W-OCLTX



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 0xA1FFDFFF = 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)

TriCore VX-toolset Project Options [TC1130.PJT]

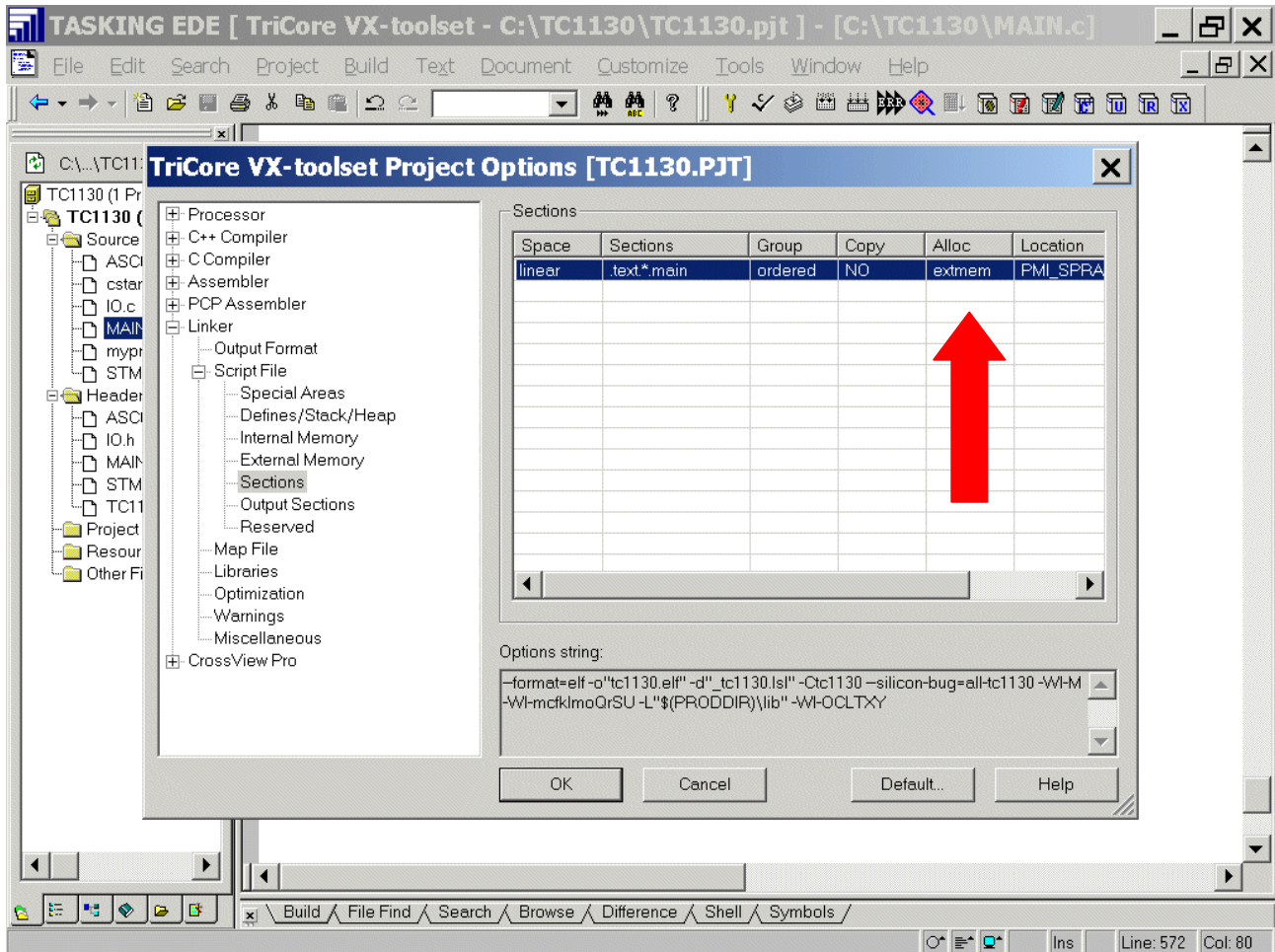
External Memory

Name	Alloc	Type	Size	Address
ext_c	ON	ROM	33546240	0xA0000000
ext_d	ON	RAM	64M	0xA4000000
TRAP_vecttable	ON	ROM	256	0xA1FFFF00

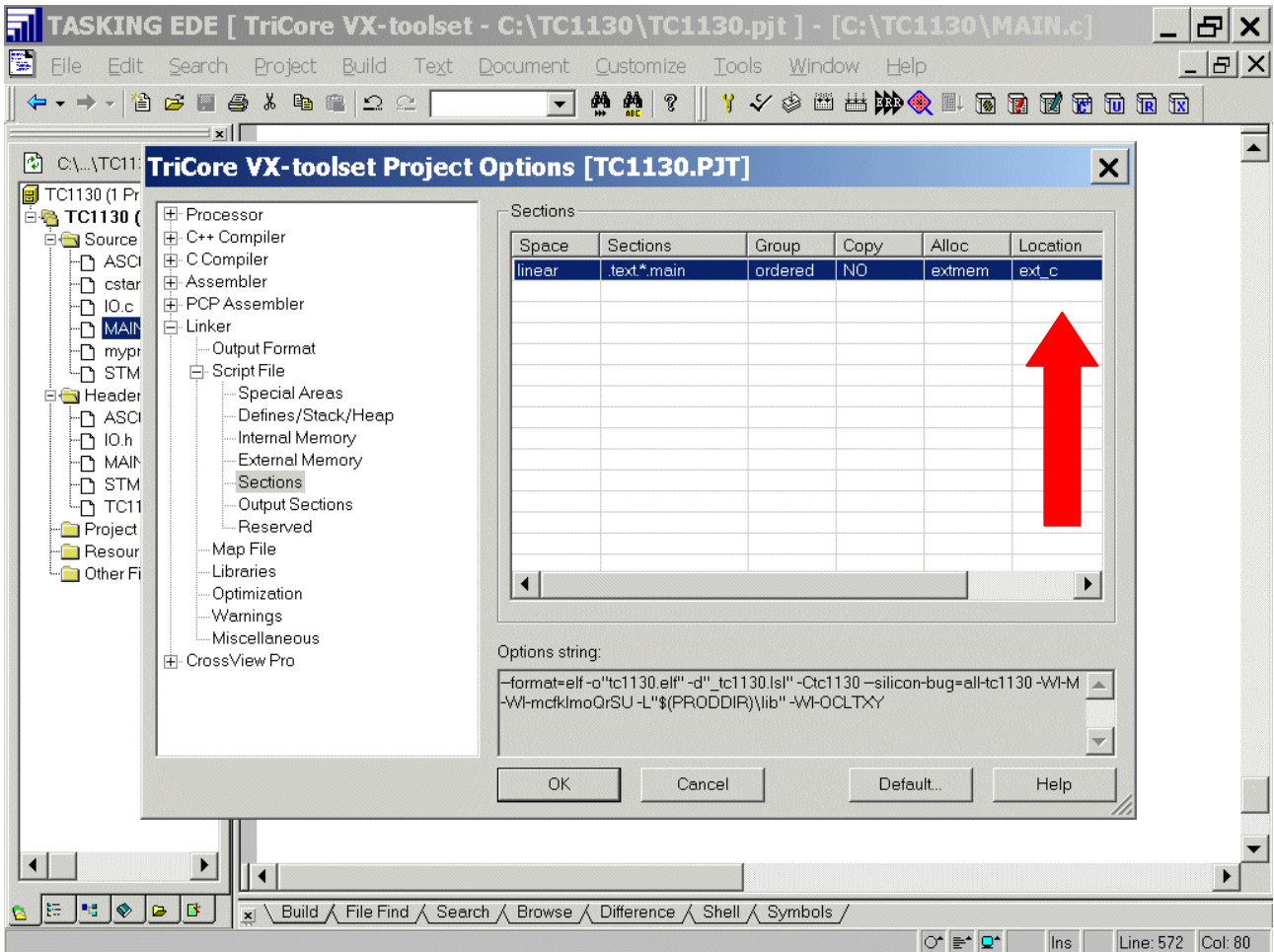
Options string:
-format=elf -o"tc1130.elf" -d"_tc1130.lis" -Ctc1130 -silicon-bug=alHc1130 -Wl-M -Wl-mcfkImoQrSU -L"\$(PRODDIR)\\lib" -Wl-OCLTX



Linker: Script File: Sections: Space=linear: Alloc: select "extmem"

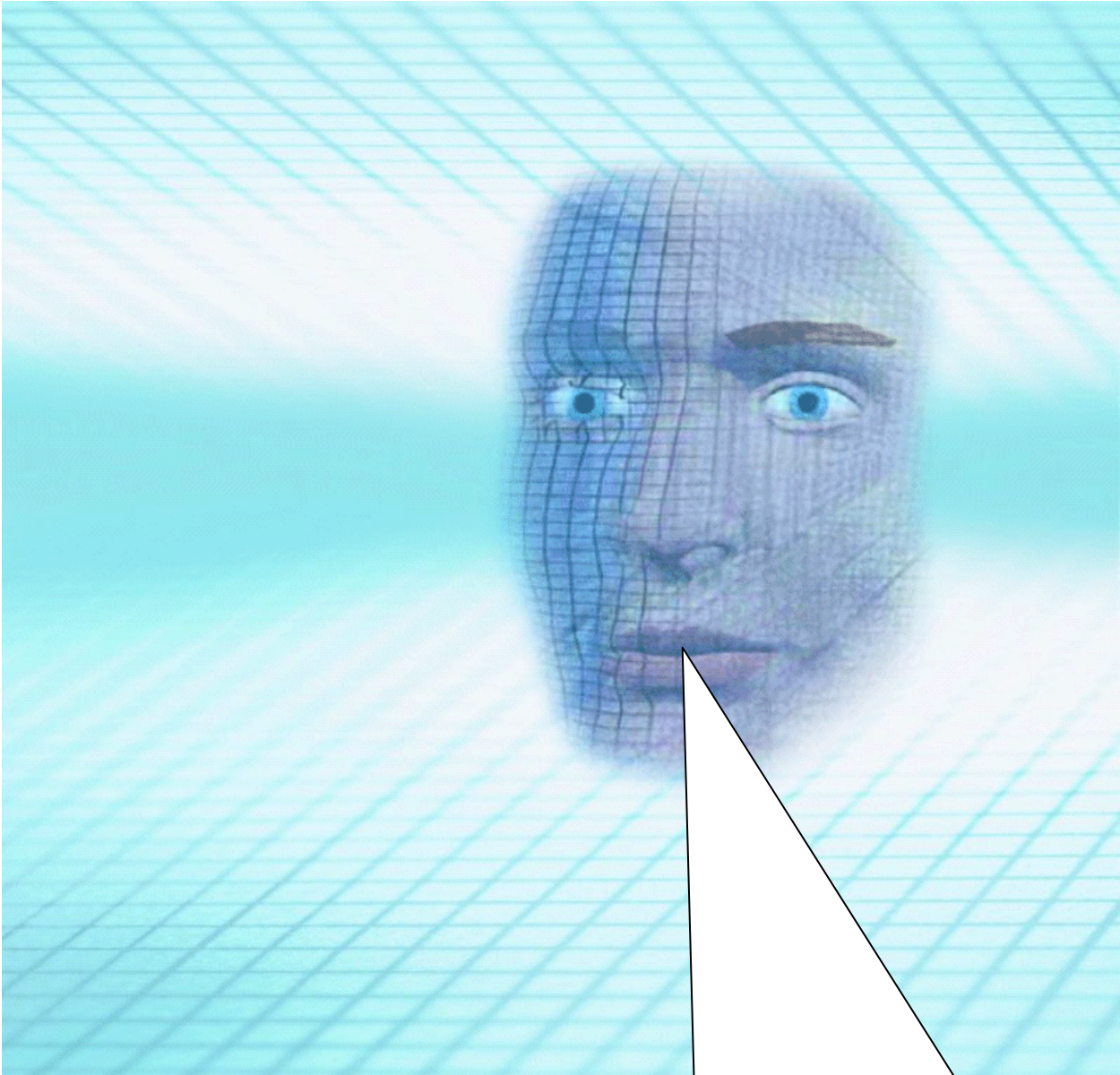


Linker: Script File: Sections: Space=linear: Location: insert "ext_c"



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`'.

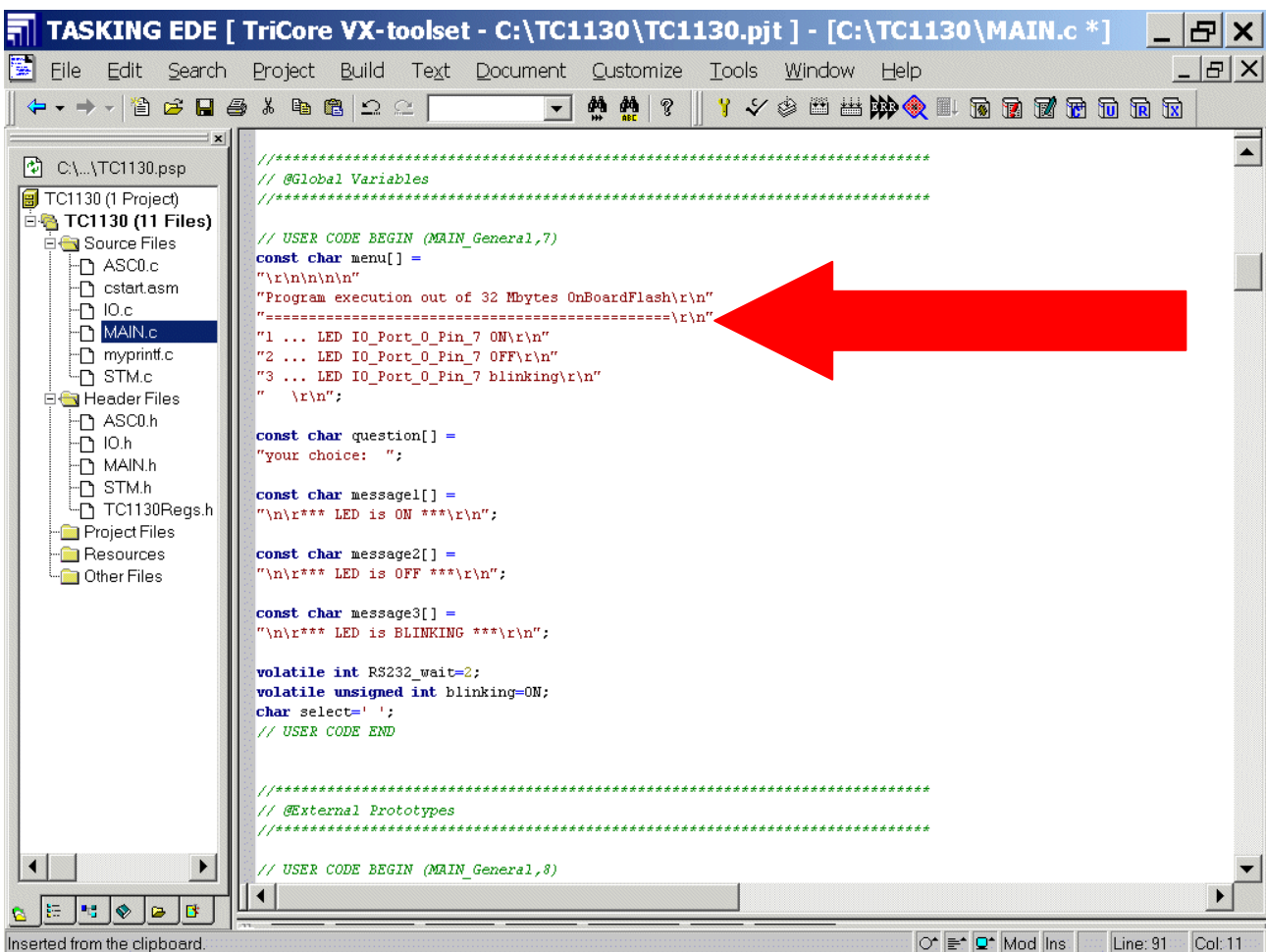
If you wish to change DAVe's generated code or add code outside these 'USER CODE' sections you will have to insert/modify your changes each time after letting DAVe regenerate code!

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

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

to

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



The screenshot shows the TASKING EDE IDE interface. The left pane displays the project structure for TC1130, with the 'MAIN.c' file selected. The main editor window shows the source code for 'MAIN.c'. A red arrow points to the updated 'const char menu[]' definition, which now includes '32 MBytes OnBoardFlash' instead of 'PMI_SPRAM'. The code also shows other global variables like 'question', 'message1', 'message2', and 'message3', and the start of the 'main' function.

```

//*****
// @Global Variables
//*****

// USER CODE BEGIN (MAIN_General,7)
const char menu[] =
"\r\n\r\n\r\n"
"Program execution out of 32 Mbytes OnBoardFlash\r\n\r\n"
"=====\r\n\r\n"
"1 ... LED IO_Port_0_Pin_7 ON\r\n\r\n"
"2 ... LED IO_Port_0_Pin_7 OFF\r\n\r\n"
"3 ... LED IO_Port_0_Pin_7 blinking\r\n\r\n"
" \r\n";

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

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

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

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

volatile int RS232_wait=2;
volatile unsigned int blinking=ON;
char select=' ';
// USER CODE END

//*****
// @External Prototypes
//*****

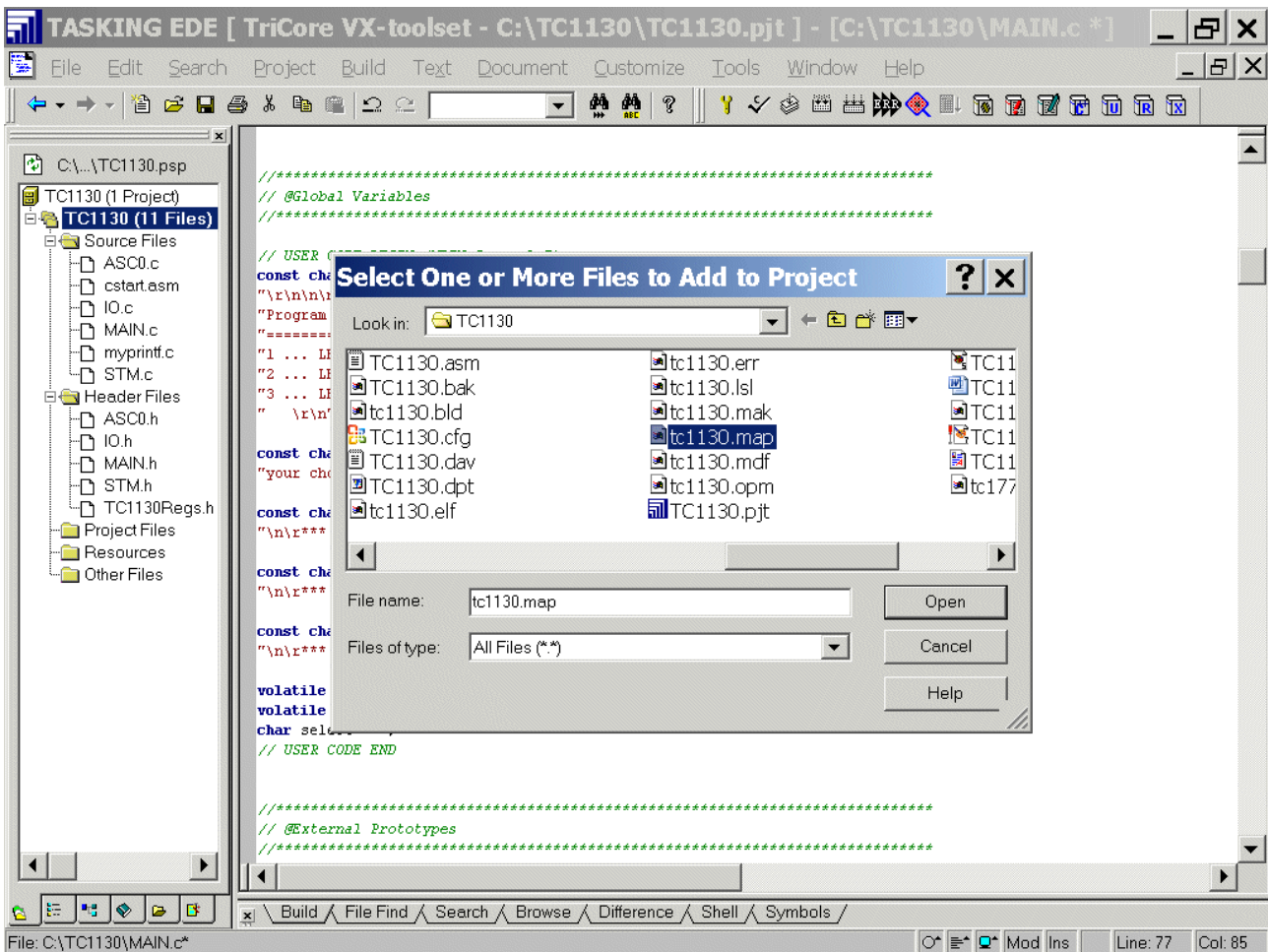
// USER CODE BEGIN (MAIN_General,8)

```

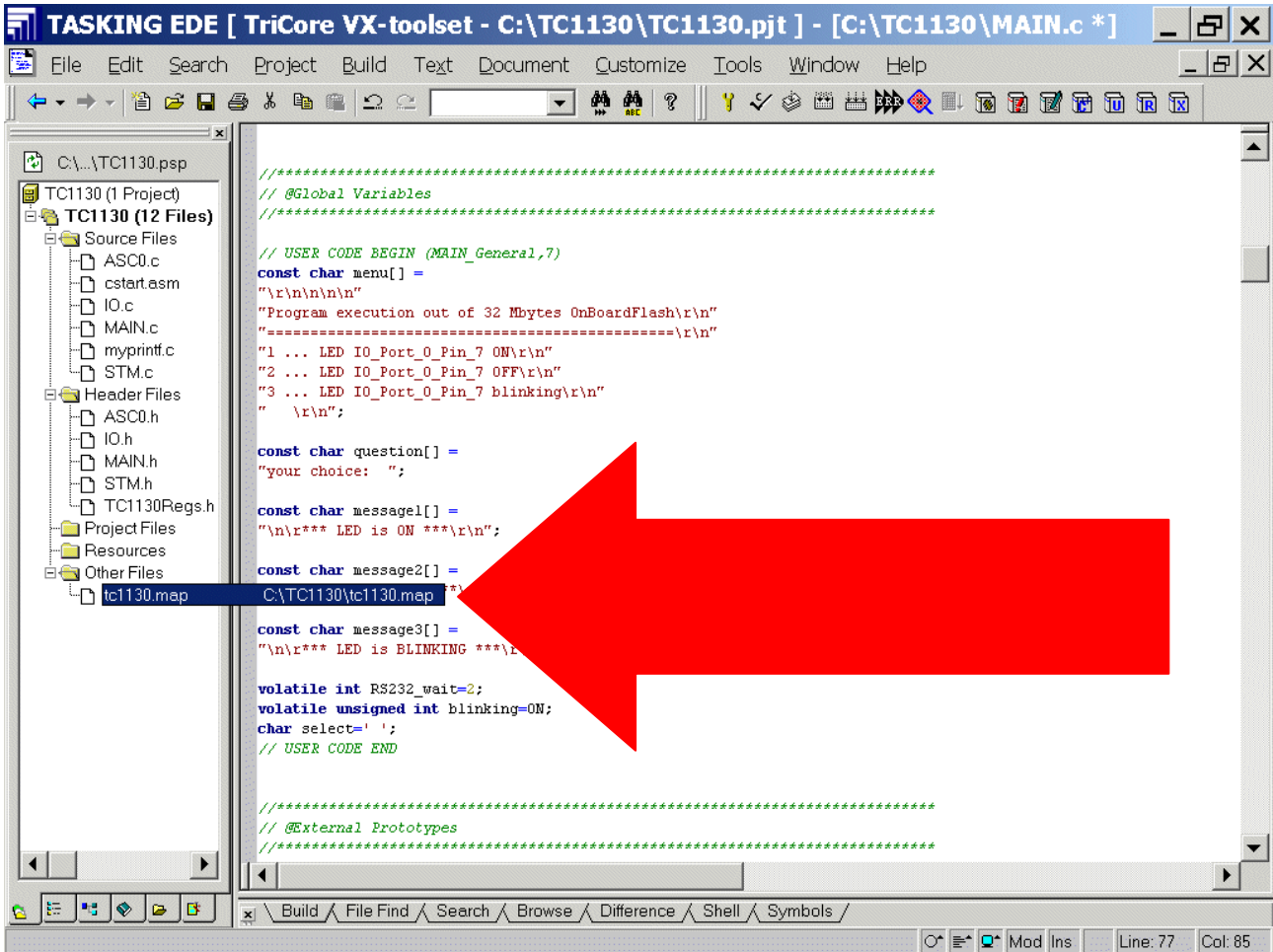

Insert Map File:

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

Select TC1130.map



Open - OK



The screenshot shows the TASKING EDE IDE interface. The title bar reads "TASKING EDE [TriCore VX-toolset - C:\TC1130\TC1130.pjt] - [C:\TC1130\MAIN.c *]". The menu bar includes File, Edit, Search, Project, Build, Text, Document, Customize, Tools, Window, and Help. The toolbar contains various icons for file operations and development. The left sidebar shows a project tree for "TC1130 (1 Project)" with 12 files, including source files (ASC0.c, cstart.asm, IO.c, MAIN.c, myprintf.c, STM.c), header files (ASC0.h, IO.h, MAIN.h, STM.h, TC1130Regs.h), project files, resources, and other files (tc1130.map). The main editor window displays the following C code:

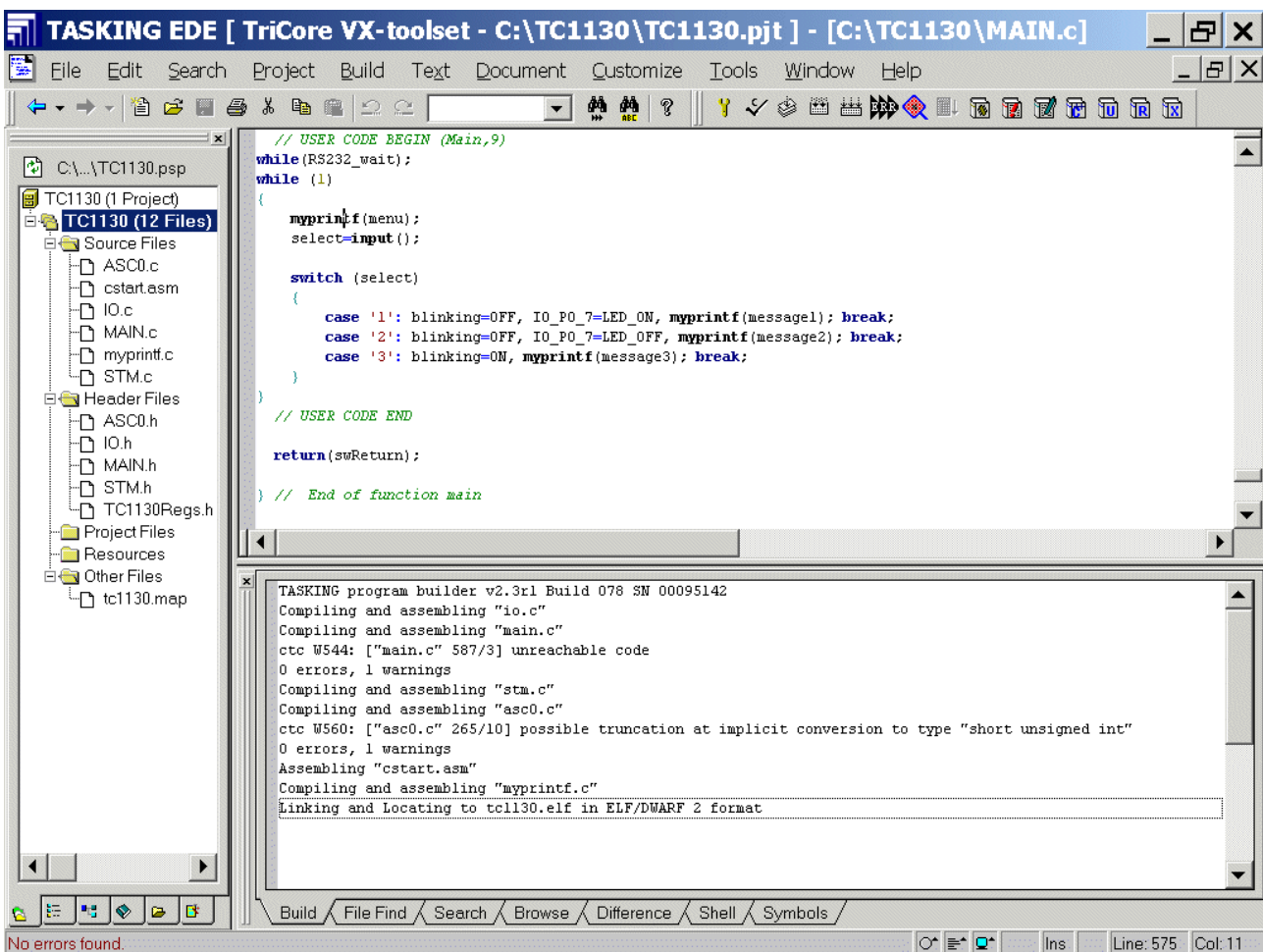
```
*****  
// @Global Variables  
*****  
  
// USER CODE BEGIN (MAIN_General,7)  
const char menu[] =  
"\r\n\r\n\r\n"  
"Program execution out of 32 Mbytes OnBoardFlash\r\n"  
"-----\r\n"  
"1 ... LED IO_Port_0_Pin_7 ON\r\n"  
"2 ... LED IO_Port_0_Pin_7 OFF\r\n"  
"3 ... LED IO_Port_0_Pin_7 blinking\r\n"  
"  \r\n";  
  
const char question[] =  
"your choice: ";  
  
const char message1[] =  
"\n\r*** LED is ON ***\r\n";  
  
const char message2[] =  
C:\TC1130\tc1130.map  
  
const char message3[] =  
"\n\r*** LED is BLINKING ***\r\n";  
  
volatile int RS232_wait=2;  
volatile unsigned int blinking=0N;  
char select=' ';  
// USER CODE END  
  
*****  
// @External Prototypes  
*****
```

A large red arrow points from the right side of the editor window to the line: `const char message2[] = C:\TC1130\tc1130.map`. The status bar at the bottom indicates "Line: 77 Col: 85".

Generate your application program:

Build
Rebuild

OR



See Map File:

Interrupt Vector Table:

Service Request Nodes		Interrupts	Func
CPU Interrupt (max.255)			
Level 16			
Level 15			
Level 14			
Level 13			
Level 12			
Level 11			
Level 10			
Level 9	STM SRN 0		
Level 8			

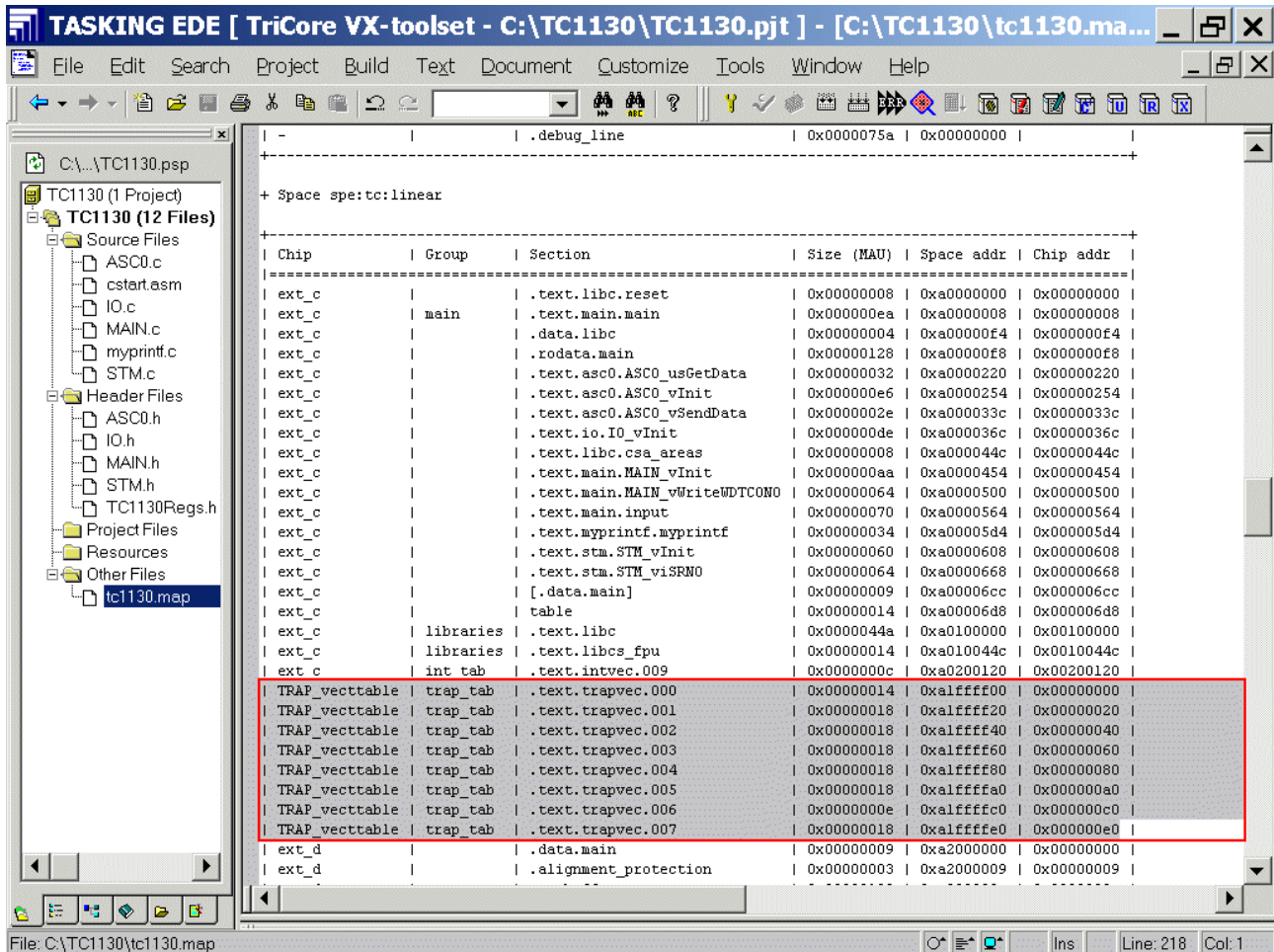
The screenshot shows the TASKING EDE interface with the memory map for TC1130. The 'int_tab' section is highlighted, showing the interrupt vector table entries. The table below is a representation of the data shown in the screenshot.

Chip	Group	Name	Size (MAU)	Space addr	Chip addr
		libc.reset	0x00000008	0xa0000000	0x00000000
	main	.main.main	0x000000ea	0xa0000008	0x00000008
		.libc	0x00000004	0xa00000f4	0x000000f4
		.data.main	0x00000128	0xa00000f8	0x000000f8
		ext.asc0.ASC0_usGetData	0x00000032	0xa0000220	0x00000220
		ext.asc0.ASC0_vInit	0x000000e6	0xa0000254	0x00000254
		ext.asc0.ASC0_vSendData	0x0000002e	0xa000033c	0x0000033c
		ext.io.IO_vInit	0x000000de	0xa000036c	0x0000036c
		ext.libc.csa_areas	0x00000008	0xa000044c	0x0000044c
		ext.main.MAIN_vInit	0x000000aa	0xa0000454	0x00000454
		ext.main.MAIN_vWriteWDTCON0	0x00000064	0xa0000500	0x00000500
		ext.main.input	0x00000070	0xa0000564	0x00000564
		ext.myprintf.myprintf	0x00000034	0xa00005d4	0x000005d4
		ext.stm.STM_vInit	0x00000060	0xa0000608	0x00000608
		ext.stm.STM_vISRNO	0x00000064	0xa0000668	0x00000668
		[.data.main]	0x00000009	0xa00006cc	0x000006cc
		table	0x00000014	0xa00006d8	0x000006d8
	libraries	.text.libc	0x0000044a	0xa0100000	0x00100000
	libraries	.text.libcs_fpu	0x00000014	0xa010044c	0x0010044c
	int_tab	.text.intvec.009	0x0000000c	0xa0200120	0x00200120
	TRAP_vecttable	trap_tab .text.trapvec.000	0x00000014	0xa1ffff00	0x00000000
	TRAP_vecttable	trap_tab .text.trapvec.001	0x00000018	0xa1ffff20	0x00000020
	TRAP_vecttable	trap_tab .text.trapvec.002	0x00000018	0xa1ffff40	0x00000040
	TRAP_vecttable	trap_tab .text.trapvec.003	0x00000018	0xa1ffff60	0x00000060
	TRAP_vecttable	trap_tab .text.trapvec.004	0x00000018	0xa1ffff80	0x00000080
	TRAP_vecttable	trap_tab .text.trapvec.005	0x00000018	0xa1ffffa0	0x000000a0
	TRAP_vecttable	trap_tab .text.trapvec.006	0x0000000e	0xa1ffffc0	0x000000c0
	TRAP_vecttable	trap_tab .text.trapvec.007	0x00000018	0xa1ffffe0	0x000000e0
	ext_d	.data.main	0x00000009	0xa2000000	0x00000000
	ext_d	.alignment_protection	0x00000003	0xa2000009	0x00000009

[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:

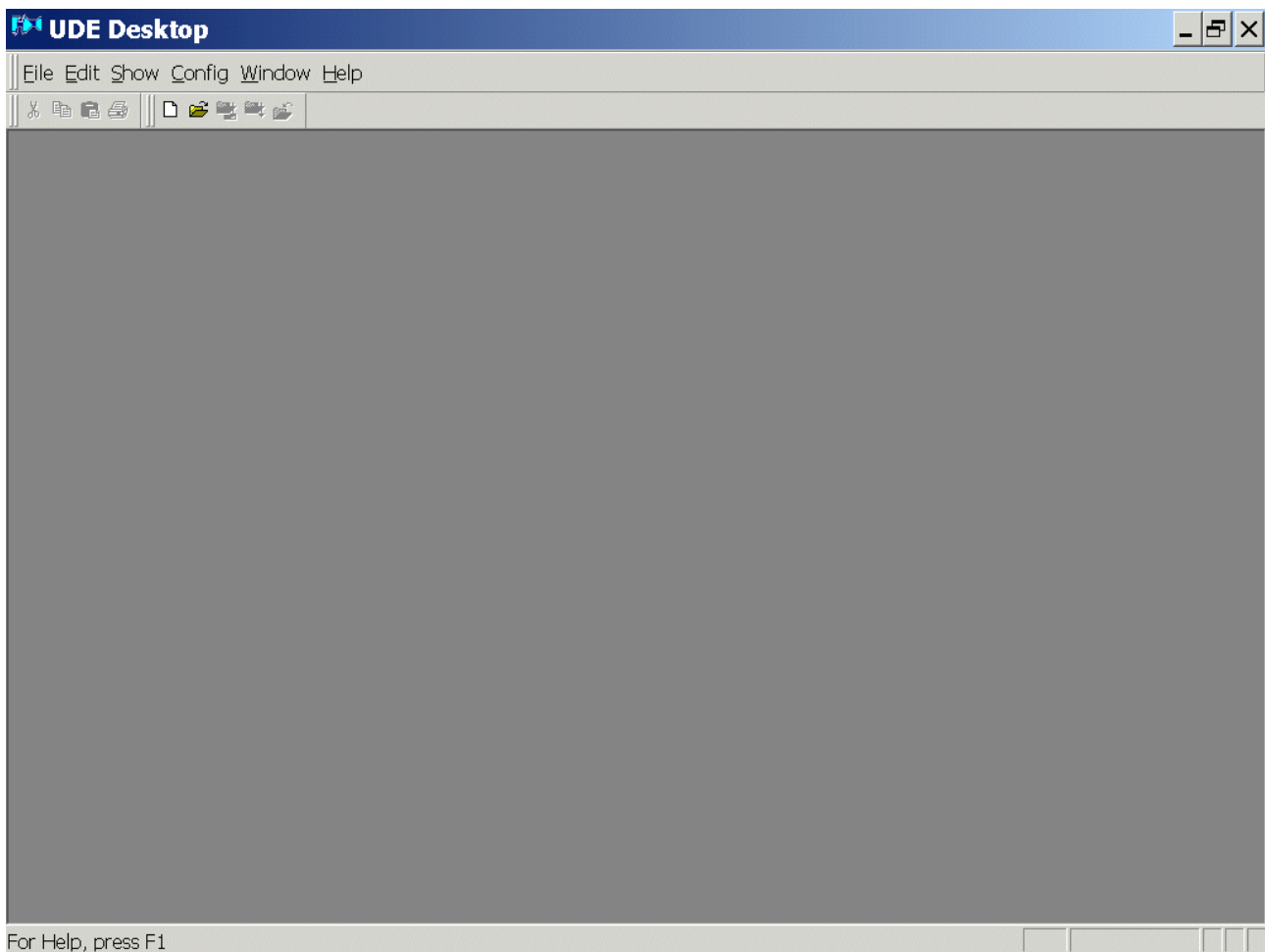
File - Close Project Space

File - Exit

Programming is now complete. You can now **load** and **run** your program:



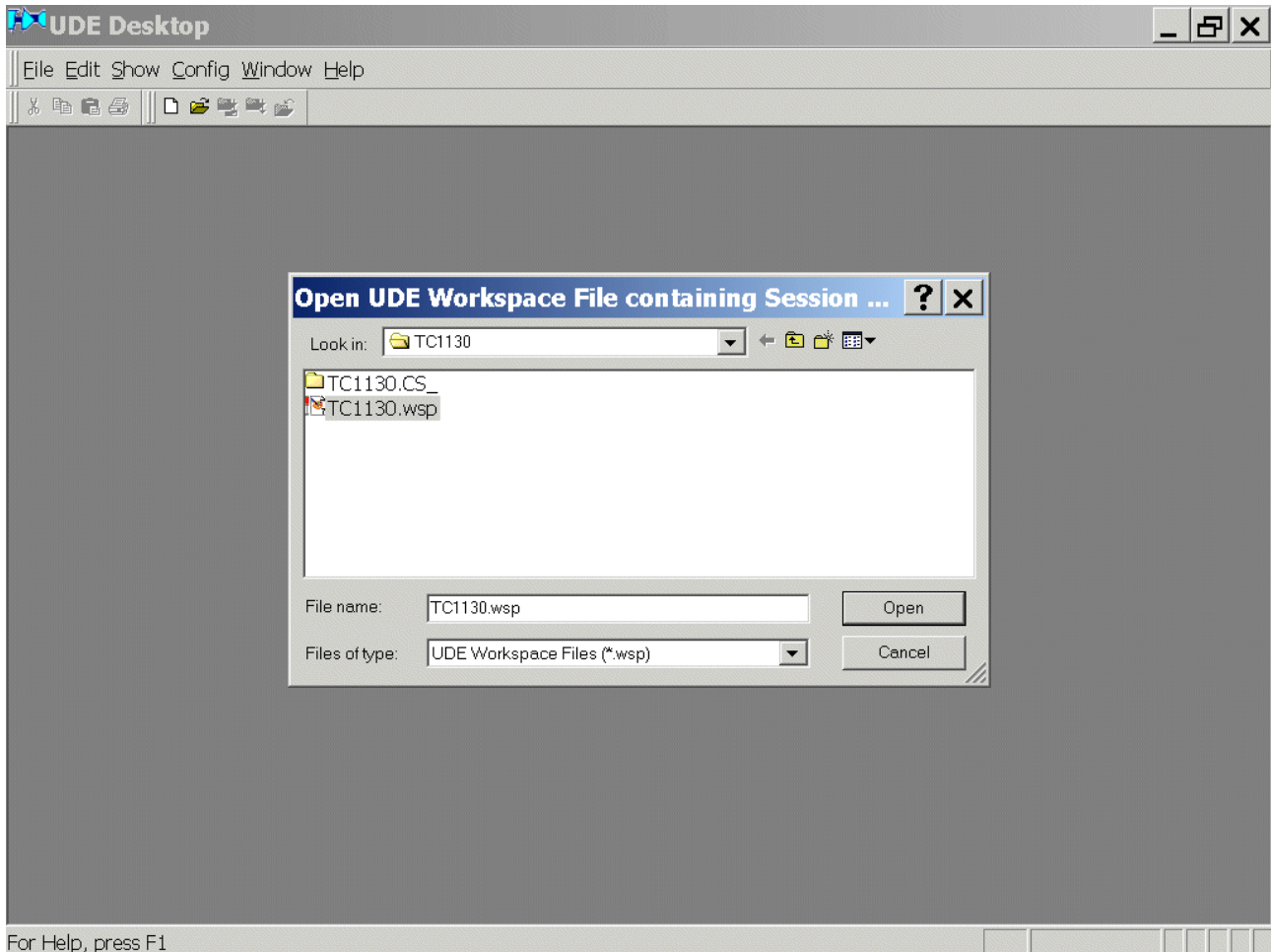
Start pls-Debugger



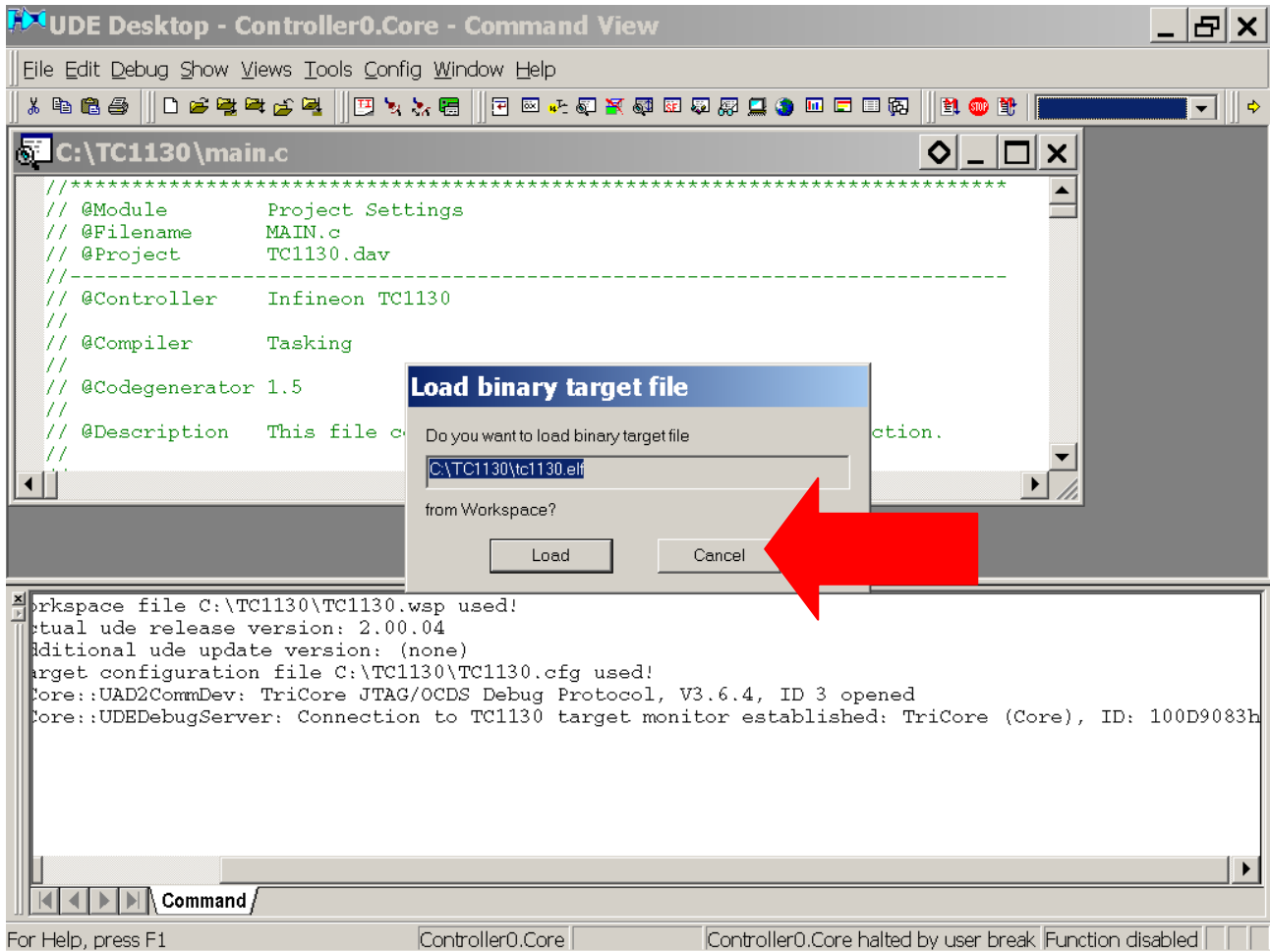
File – Open Workspace

Look in: select C:\TC1130

File name: select TC1130.wsp

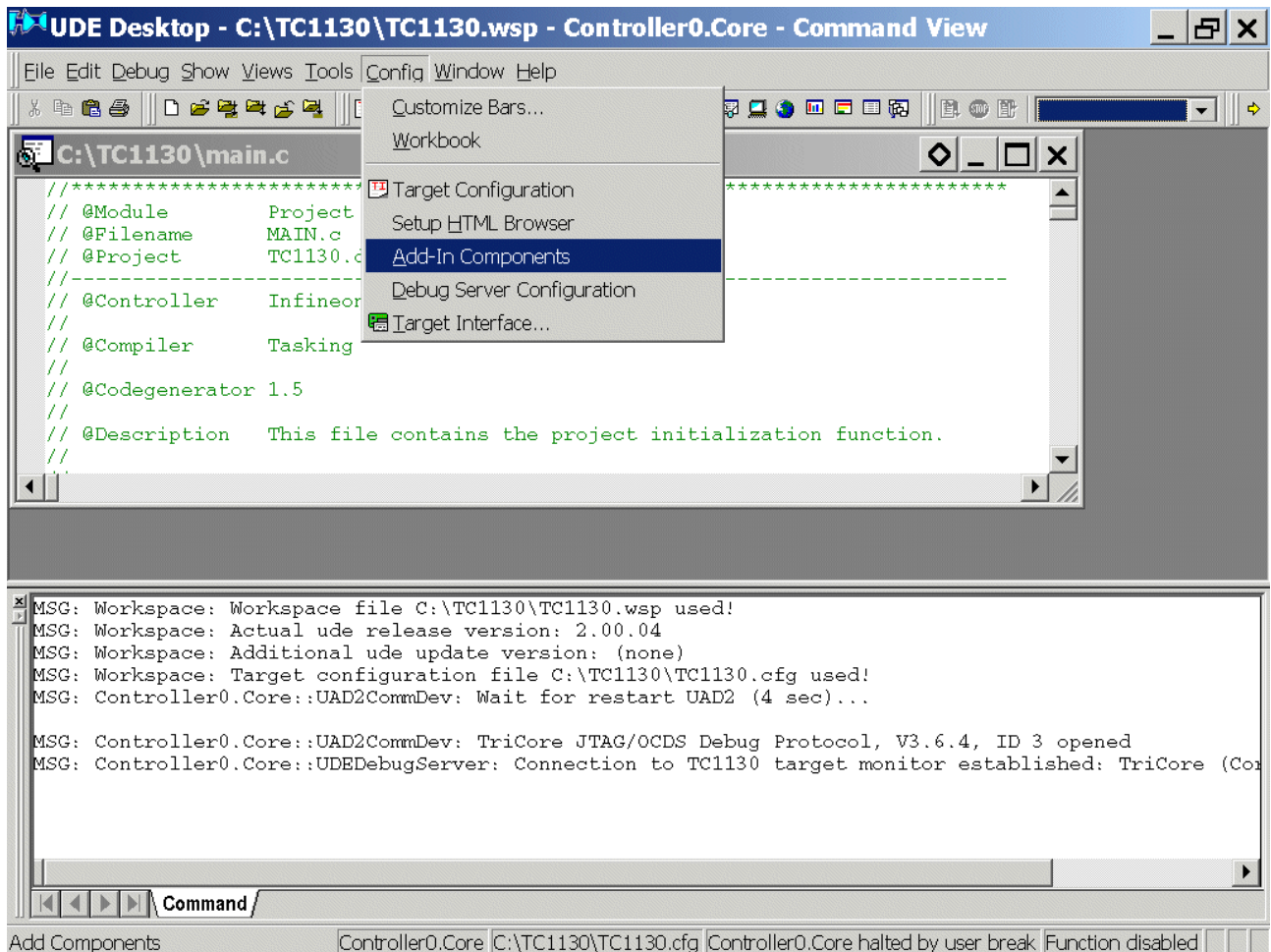


Open

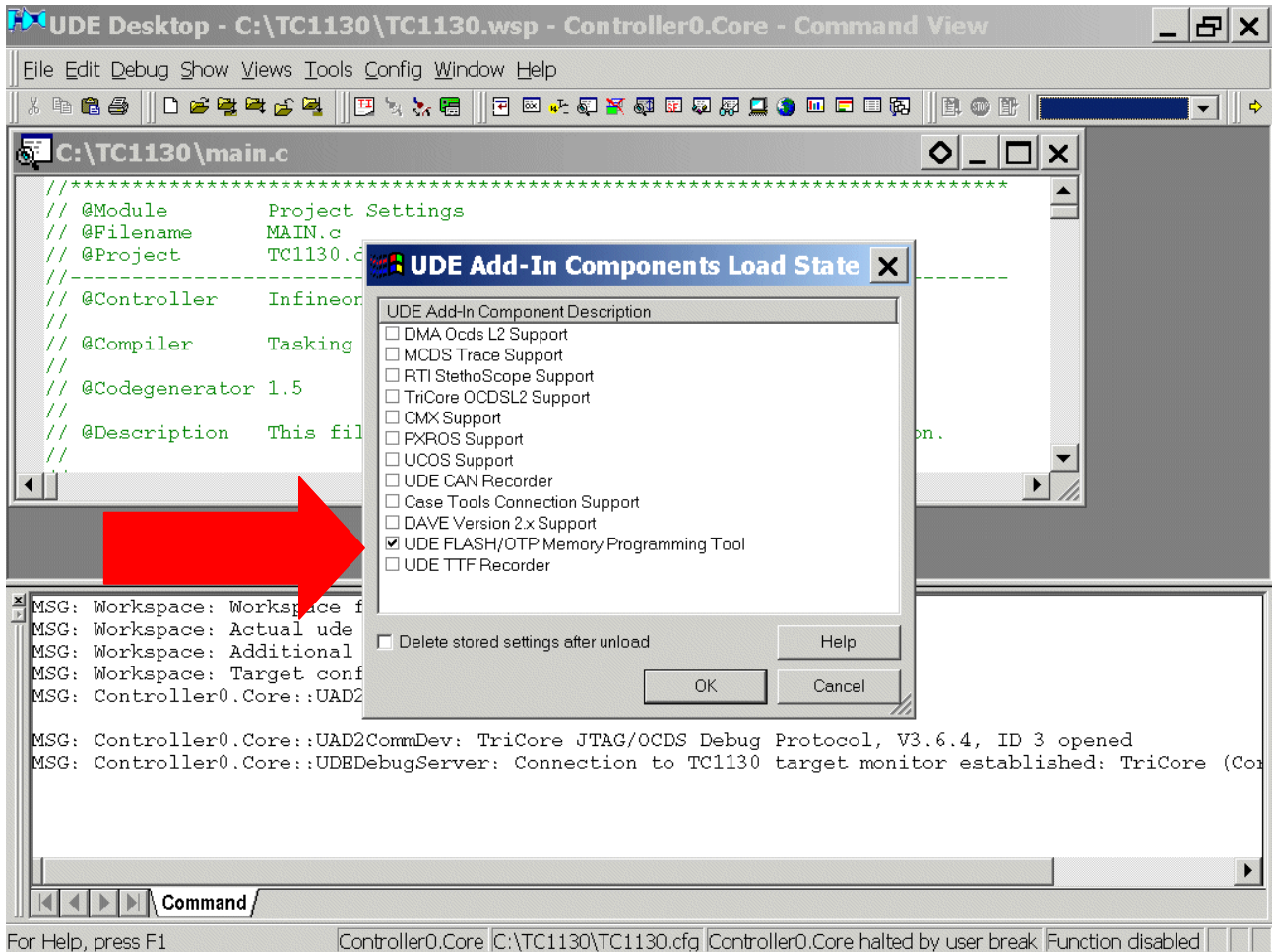


Cancel

Config – Add-In Components

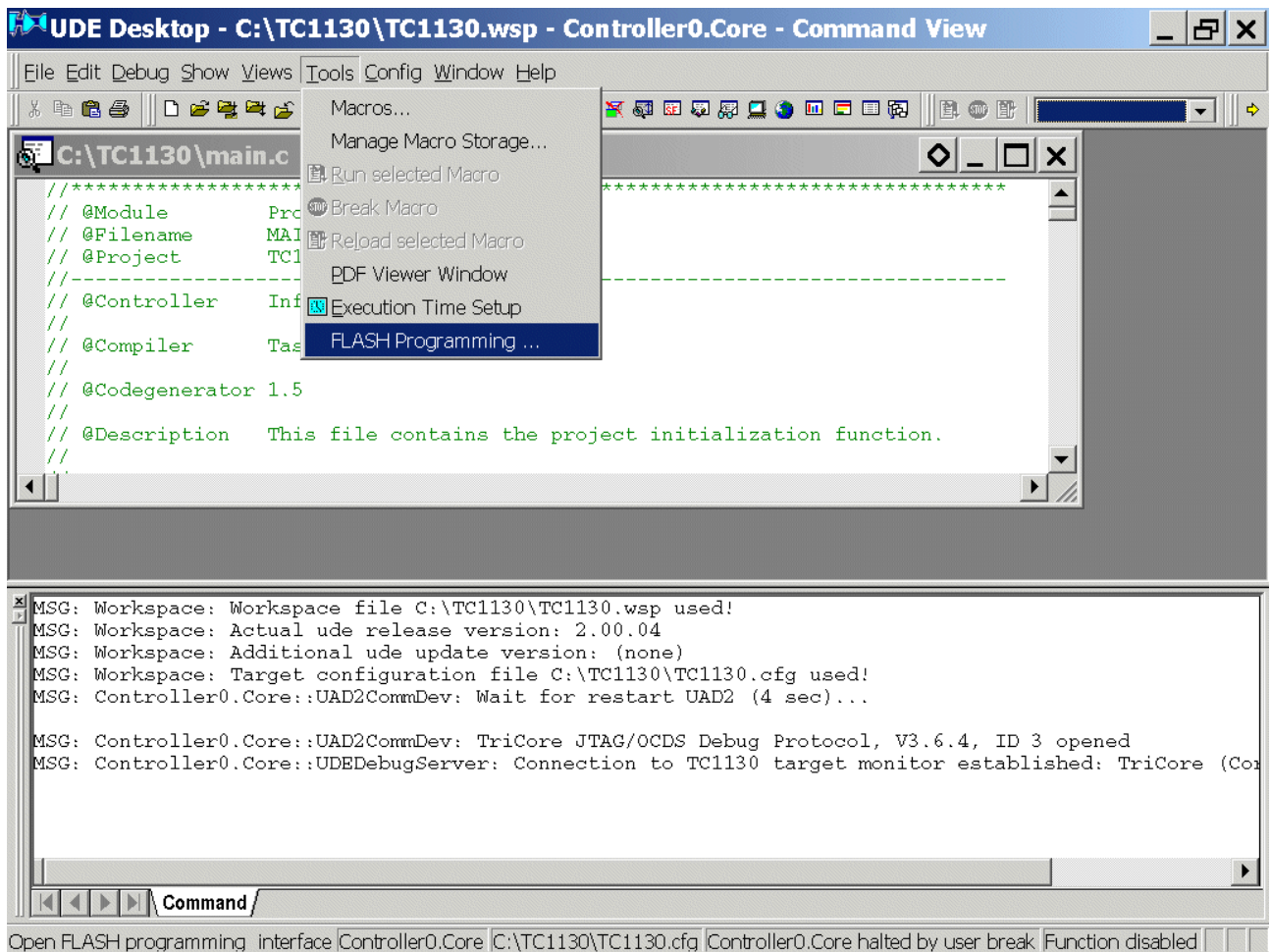


Click ✓ UDE FLASH/OTP Memory Programming Tool



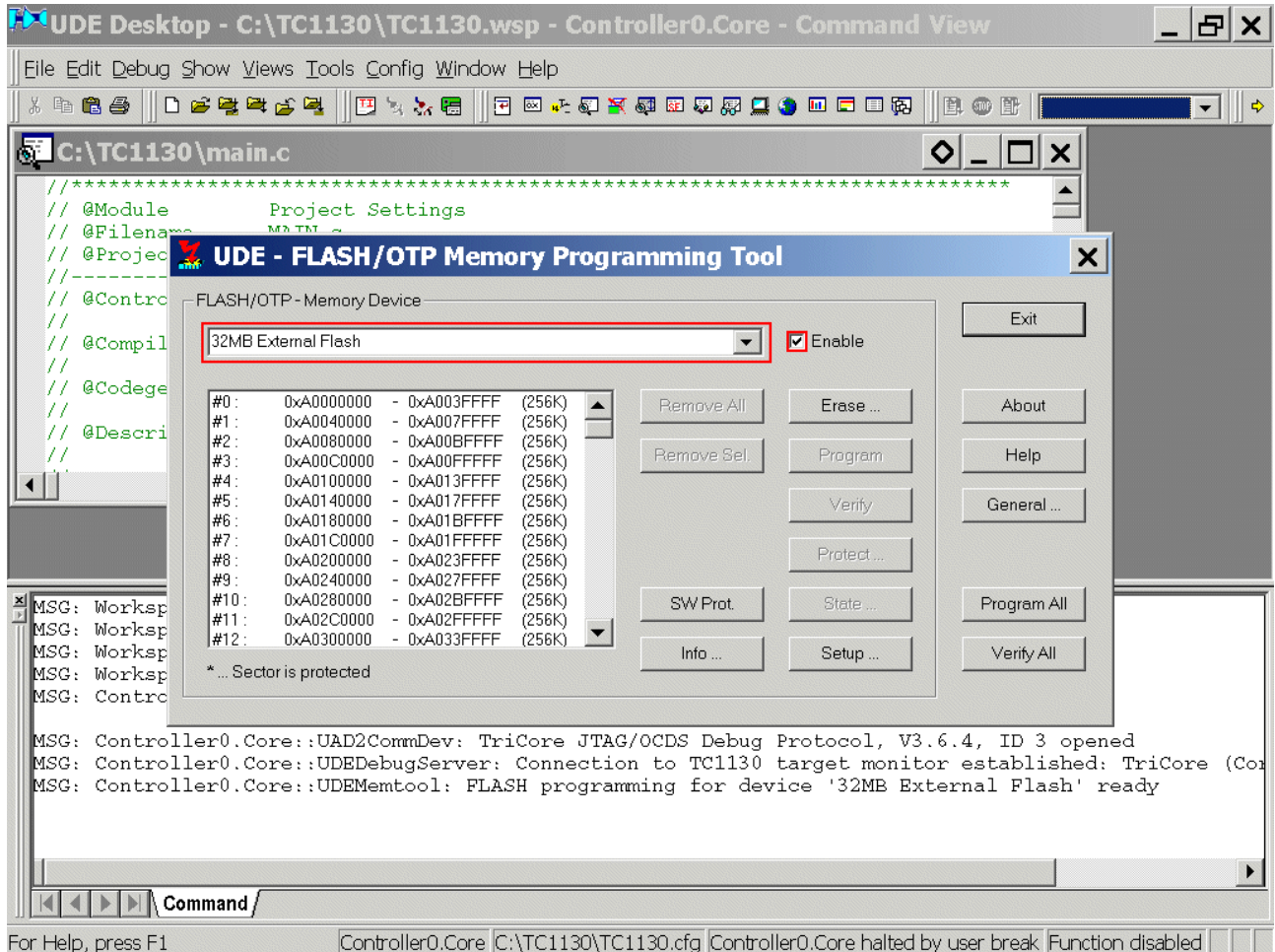
OK

Tools – FLASH Programming ...



FLASH/OTP – Memory Device: **select** 32MB External Flash (not ready)

FLASH/OTP – Memory Device: **click** ✓ Enable



The screenshot shows the UDE Desktop interface. The main window is titled "UDE Desktop - C:\TC1130\TC1130.wsp - Controller0.Core - Command View". A code editor in the background shows C code for "main.c". Overlaid on the code editor is the "UDE - FLASH/OTP Memory Programming Tool" dialog box. The dialog box has a title bar with a close button. Inside, there is a section "FLASH/OTP - Memory Device" with a dropdown menu currently set to "32MB External Flash" (highlighted with a red box) and a checked "Enable" checkbox. Below this is a list of memory sectors:

Index	Start Address	End Address	Size
#0:	0xA0000000	0xA003FFFF	(256K)
#1:	0xA0040000	0xA007FFFF	(256K)
#2:	0xA0080000	0xA00BFFFF	(256K)
#3:	0xA00C0000	0xA00FFFFF	(256K)
#4:	0xA0100000	0xA013FFFF	(256K)
#5:	0xA0140000	0xA017FFFF	(256K)
#6:	0xA0180000	0xA01BFFFF	(256K)
#7:	0xA01C0000	0xA01FFFFF	(256K)
#8:	0xA0200000	0xA023FFFF	(256K)
#9:	0xA0240000	0xA027FFFF	(256K)
#10:	0xA0280000	0xA02BFFFF	(256K)
#11:	0xA02C0000	0xA02FFFFF	(256K)
#12:	0xA0300000	0xA033FFFF	(256K)

Below the list, there is a note: "* ... Sector is protected". The dialog box also contains several buttons: "Remove All", "Erase ...", "Remove Sel.", "Program", "Verify", "Protect ...", "SW Prot.", "State ...", "Info ...", "Setup ...", "Exit", "About", "Help", "General ...", "Program All", and "Verify All". The background code editor shows the following code:

```

// *****
// @Module      Project Settings
// @Filename    MD_TM_...
// @Project     ...
// @Control    ...
// @Compiler   ...
// @Codegen    ...
// @Descriptor ...
// *****

```

The command window at the bottom shows the following messages:

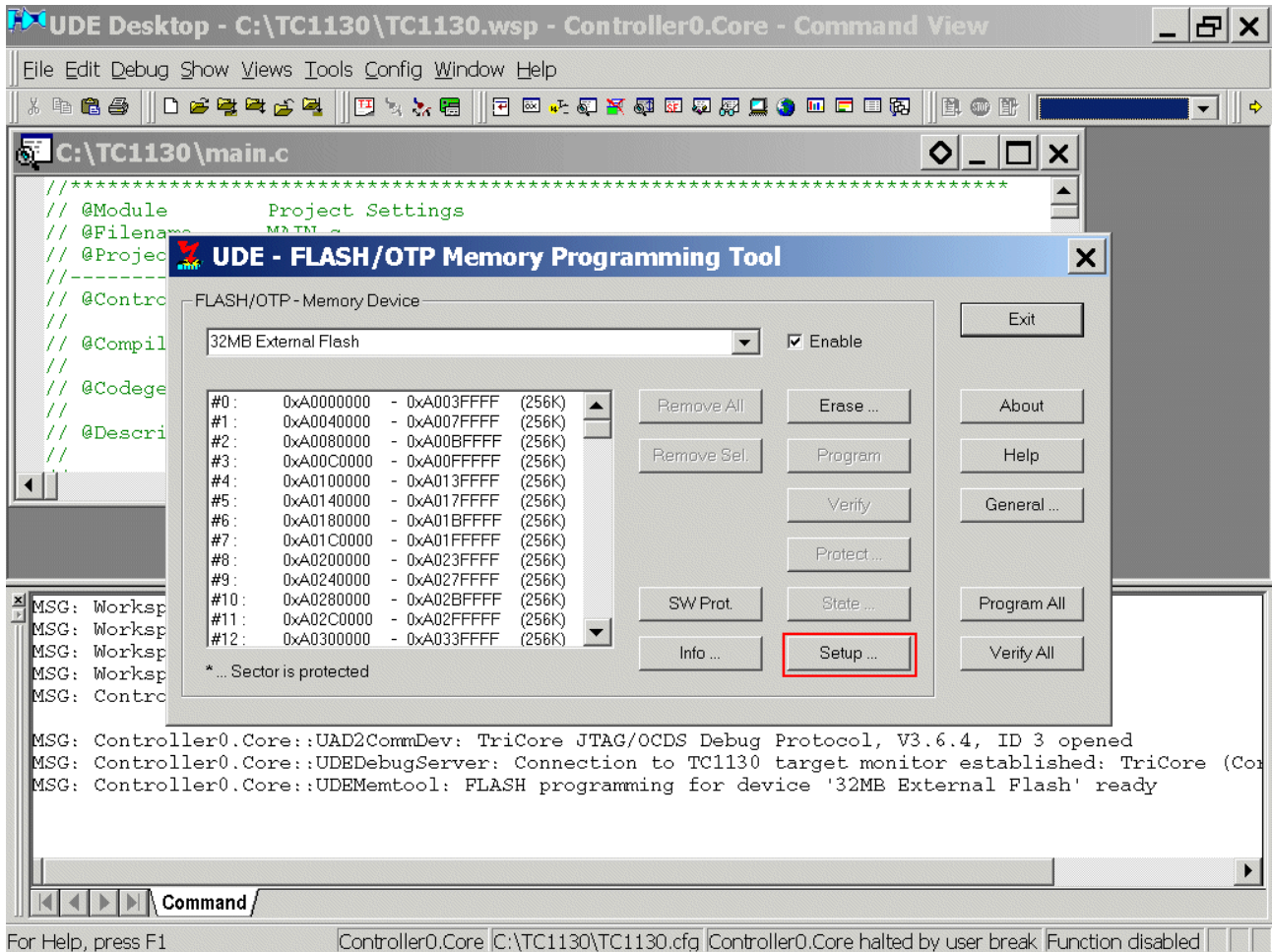
```

MSG: Worksp
MSG: Worksp
MSG: Worksp
MSG: Worksp
MSG: Contro
MSG: Controller0.Core::UAD2CommDev: TriCore JTAG/OCDS Debug Protocol, V3.6.4, ID 3 opened
MSG: Controller0.Core::UDEDebugServer: Connection to TC1130 target monitor established: TriCore (Co
MSG: Controller0.Core::UDEMemtool: FLASH programming for device '32MB External Flash' ready

```

The status bar at the bottom of the window displays: "For Help, press F1 | Controller0.Core C:\TC1130\TC1130.cfg | Controller0.Core halted by user break | Function disabled |"

click Setup ...



The screenshot shows the UDE Desktop interface with the 'UDE - FLASH/OTP Memory Programming Tool' dialog box open. The dialog box displays a list of memory sectors for a '32MB External Flash' device. The 'Setup ...' button is highlighted with a red box.

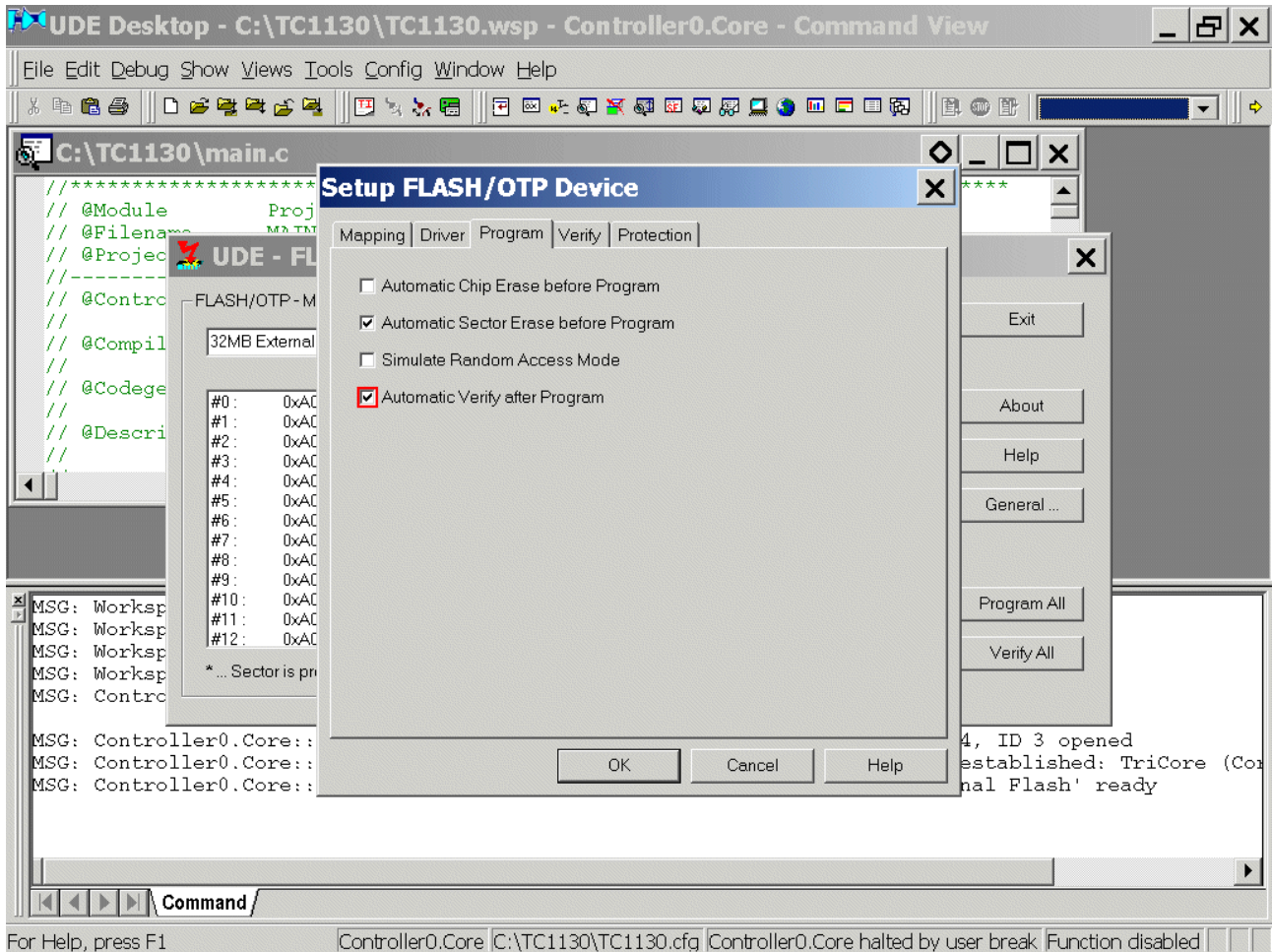
Address	Start	End	Size
#0:	0xA0000000	0xA003FFFF	(256K)
#1:	0xA0040000	0xA007FFFF	(256K)
#2:	0xA0080000	0xA00BFFFF	(256K)
#3:	0xA00C0000	0xA00FFFFF	(256K)
#4:	0xA0100000	0xA013FFFF	(256K)
#5:	0xA0140000	0xA017FFFF	(256K)
#6:	0xA0180000	0xA01BFFFF	(256K)
#7:	0xA01C0000	0xA01FFFFF	(256K)
#8:	0xA0200000	0xA023FFFF	(256K)
#9:	0xA0240000	0xA027FFFF	(256K)
#10:	0xA0280000	0xA02BFFFF	(256K)
#11:	0xA02C0000	0xA02FFFFF	(256K)
#12:	0xA0300000	0xA033FFFF	(256K)

Messages in the background:

```

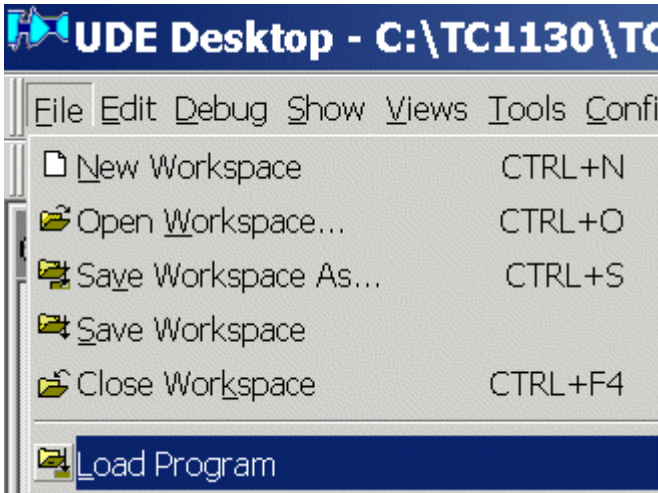
MSG: Worksp
MSG: Worksp
MSG: Worksp
MSG: Worksp
MSG: Contro
MSG: Controller0.Core::UAD2CommDev: TriCore JTAG/OCDS Debug Protocol, V3.6.4, ID 3 opened
MSG: Controller0.Core::UDEDebugServer: Connection to TC1130 target monitor established: TriCore (Co
MSG: Controller0.Core::UDEMentool: FLASH programming for device '32MB External Flash' ready
  
```

Program: **click** ✓ Automatic Verify after Program



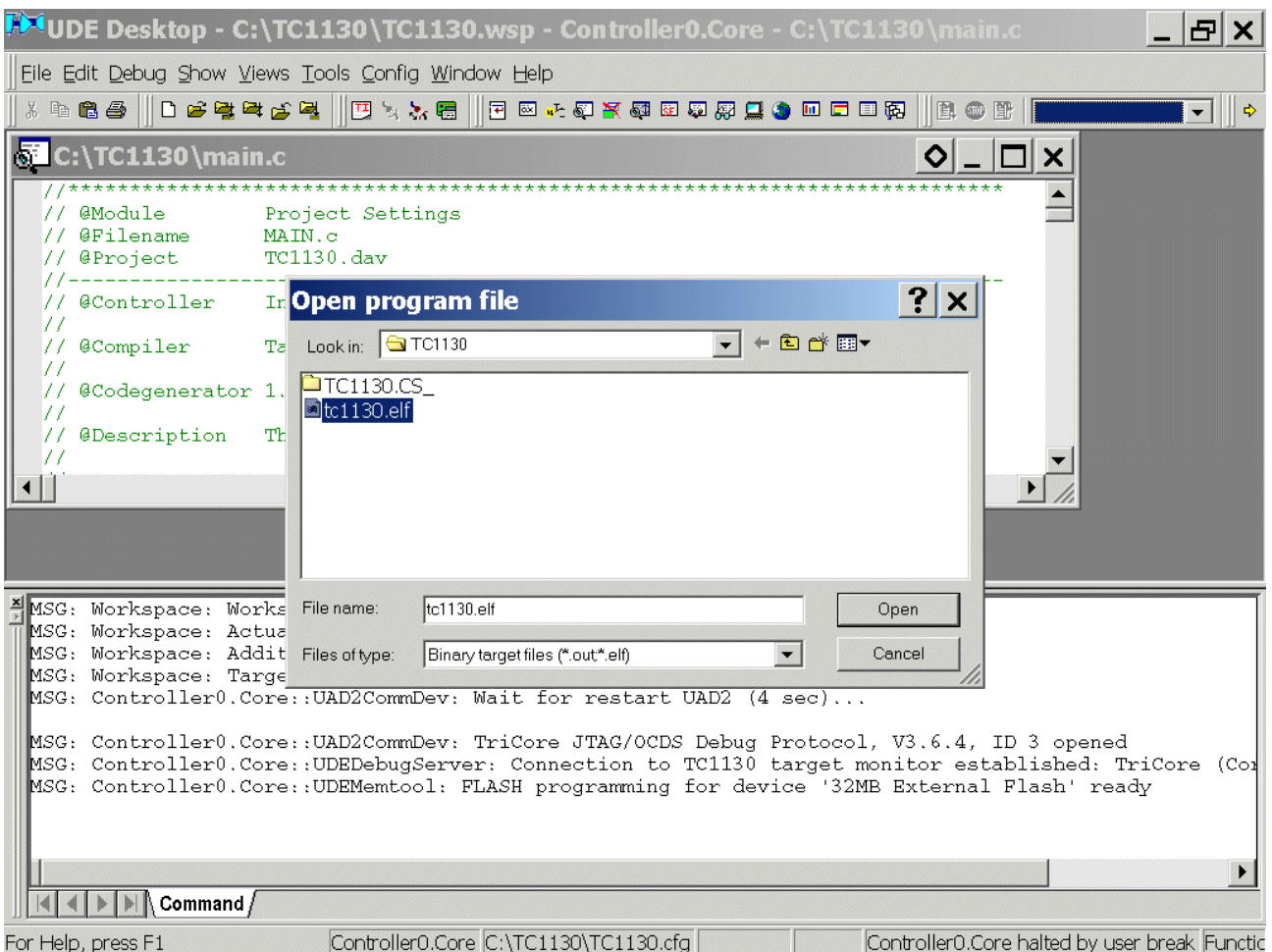
OK
Exit

File – Load Program



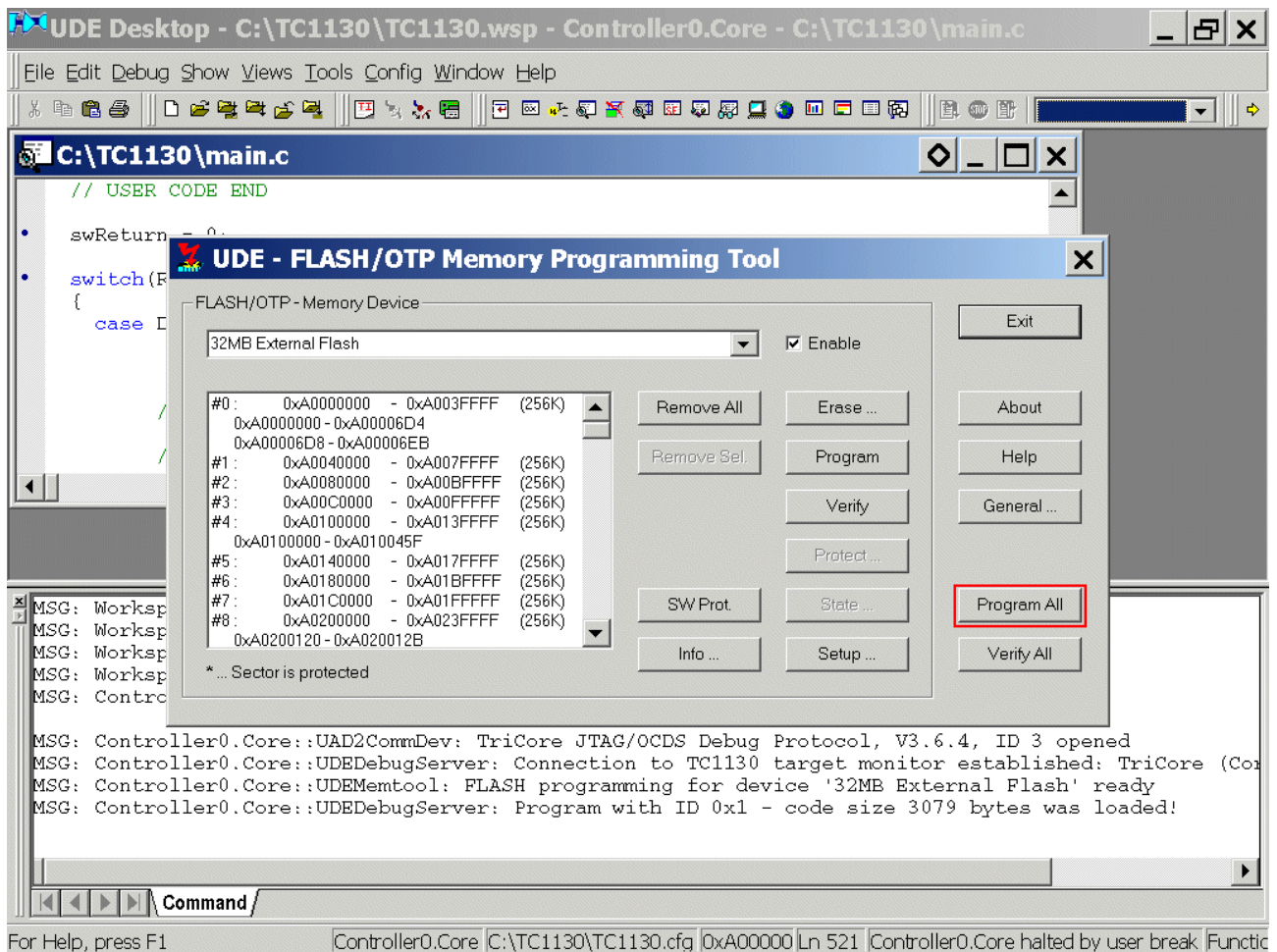
Look in: select TC1130

File name: select TC1130.elf



Open

Click Program All

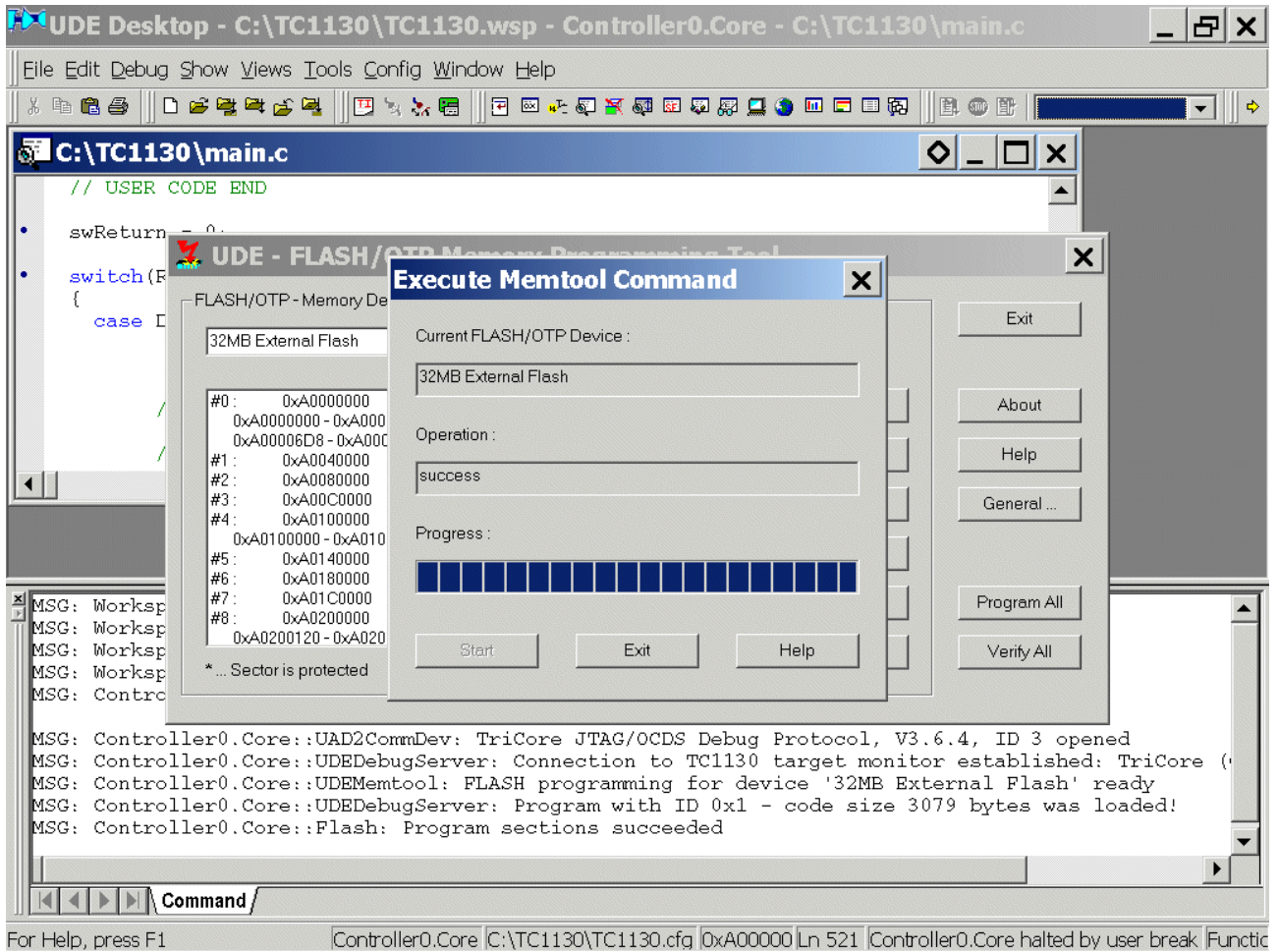


The screenshot shows the UDE Desktop environment. A dialog box titled "UDE - FLASH/OTP Memory Programming Tool" is open, displaying a list of memory sectors for a "32MB External Flash" device. The "Program All" button is highlighted with a red box. The background shows a code editor with C code and a command window with the following messages:

```

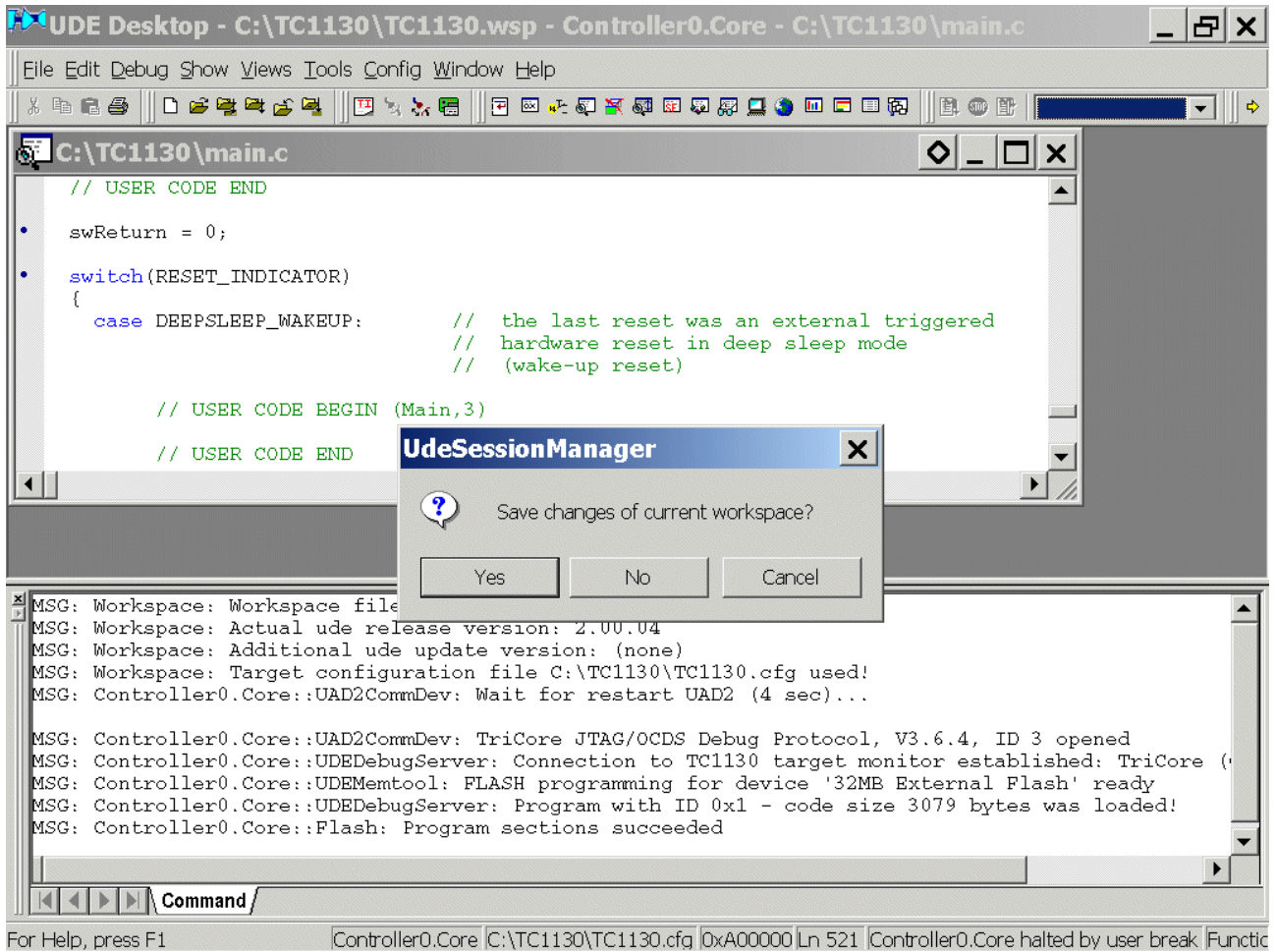
MSG: Worksp
MSG: Worksp
MSG: Worksp
MSG: Worksp
MSG: Contro
MSG: Controller0.Core::UAD2CommDev: TriCore JTAG/OCDS Debug Protocol, V3.6.4, ID 3 opened
MSG: Controller0.Core::UDEDebugServer: Connection to TC1130 target monitor established: TriCore (Co
MSG: Controller0.Core::UDEMentool: FLASH programming for device '32MB External Flash' ready
MSG: Controller0.Core::UDEDebugServer: Program with ID 0x1 - code size 3079 bytes was loaded!
  
```

At the bottom of the window, the status bar shows: "For Help, press F1 Controller0.Core C:\TC1130\TC1130.cfg 0xA00000 Ln 521 Controller0.Core halted by user break Functio



Exit
Exit

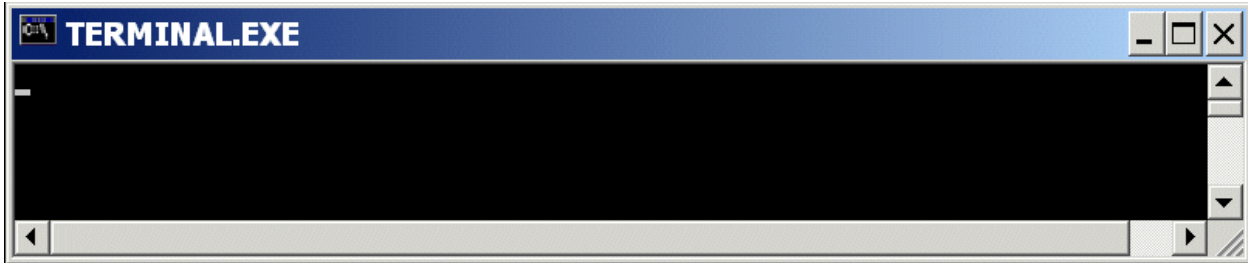
File – Close Workspace



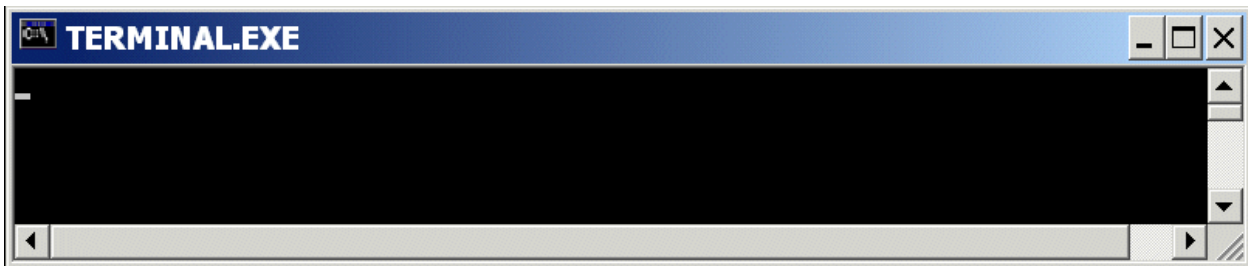
Yes

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 (0x000004) of the memory device attached to CS0.

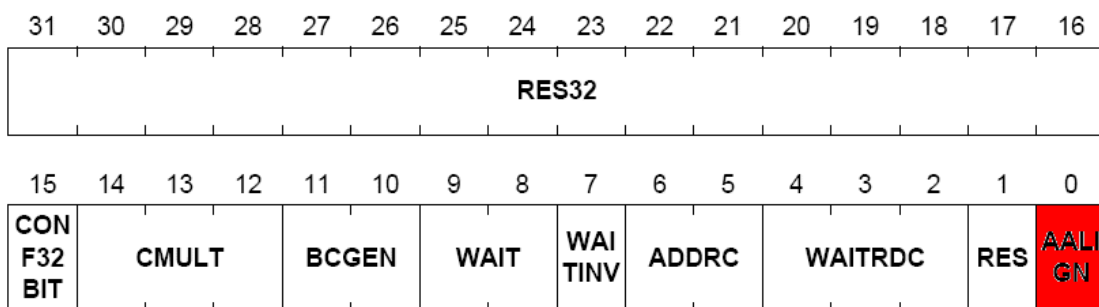
The result of this read-access is used during "Power-ON" to configure the EBU.

[To see the configuration value click here.](#)

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

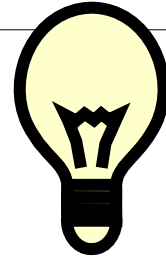
Bit#	Description	Value
0	AALIGN: Address alignment	ON
2-4	WAITRDC: Read access wait states	3 wait states
5-6	ADDRC: Address cycles	3 cycles
7	WAITINV: WAIT level	Active low
8-9	WAIT: Variable wait-state insertion	Disabled
10-11	BCGEN: Byte control signal timing	Chip select mode
12-14	CMULT: Wait cycle multiplier	1
15	CFG: Boot memory data width	32 bit

EBU Boot Configuration Value



Field	Bits	Description
AALIGN	0	Address Alignment Loaded into BUSCON0.AALIGN (see Section 14.12.3).

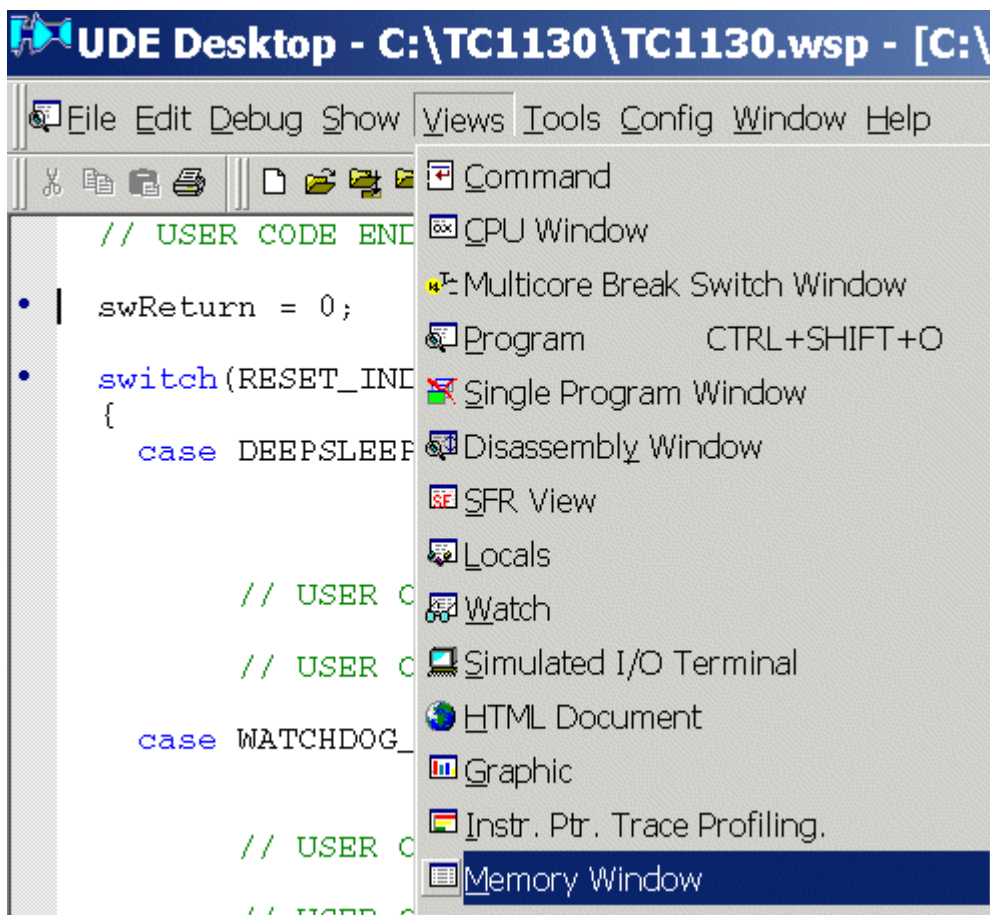
"external-boot-memory-configuration-word" at address 0xA0000010
(0xA0000004 << 2 = 0xA0000010 , due to 32 bit alignment)



To check this we use the pls-debugger:



Views – Memory Window



Insert 0xA0000000 <ENTER>



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

0xA000.0000 0xA000.0004 0xA000.0008 0xA000.000C

The screenshot shows the UDE Desktop software interface. The main window displays a memory map for 'Controller0.Core' with addresses from 0x0 to 0x14. The address 0xA000.0004 is highlighted in red. A red arrow points from the text 'Insert 0xA0000000 <ENTER>' to the address 0xA000.0000 in the memory map. Another red arrow points from the text '0xA000.0010' to the address 0xA000.0010 in the memory map. A red thought bubble is positioned below the 0xA000.0010 address, containing the text: 'External Boot Memory Configuration Word @ 0xA000.0010 Address Alignment = 1'. The bottom window shows a command window with the following text:

```

MSG: Workspace: Additional ude update version: (none)
MSG: Workspace: Target configuration file C:\TC1130\TC1130.cfg
MSG: Controller0.Core::UAD2CommDev: TriCore JTAG/OCDS D
MSG: Controller0.Core::UDEMentool: FLASH programming for de
MSG: Controller0.Core::UDEDebugServer: Connection to TC1130 ta
MSG: Controller0.Core::UDEDebugServer: Program with ID 0x1 - code size 3
MSG: Controller0.Core::Flash: Program sections succeeded
    
```

At the bottom of the screenshot, the status bar shows: 'For Help, press F1 Controller0.Core C:\TC1130\TC1130.cfg |Address:0x0 Length:0x14F Controller0.Core halted by user bre'



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)

The screenshot shows the UDE Designer interface. The main window displays a memory dump for 'Controller0.Core - Memory from 0xA0000000...'. The address 0xA0000010 is highlighted in red, and the value 0000806D is highlighted in green. A red callout bubble points to this location, stating 'Wrong Value (0x.....) Correct Location (0xA0000010)'. Another red callout bubble points to the value 0000806D at address 0xA0000004, stating 'Correct Value (0x0000806D) Wrong Location (0xA0000004)'. The command window at the bottom shows the following messages:

```

MSG: Controller0.Core::UAD2CommDev: TriCore JTAG/OCDS Debug Protocol, V3.6.4, ID 3 opened
MSG: Controller0.Core::UDEMemtool: 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 3091 bytes was loaded!
MSG: Controller0.Core::Flash: Program sections succeeded
MSG: Controller0.Core::UDEDebugServer: Program with ID 0x1 - code size 3079 bytes was loaded!
MSG: Controller0.Core::Flash: Program sections succeeded
  
```

→ 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

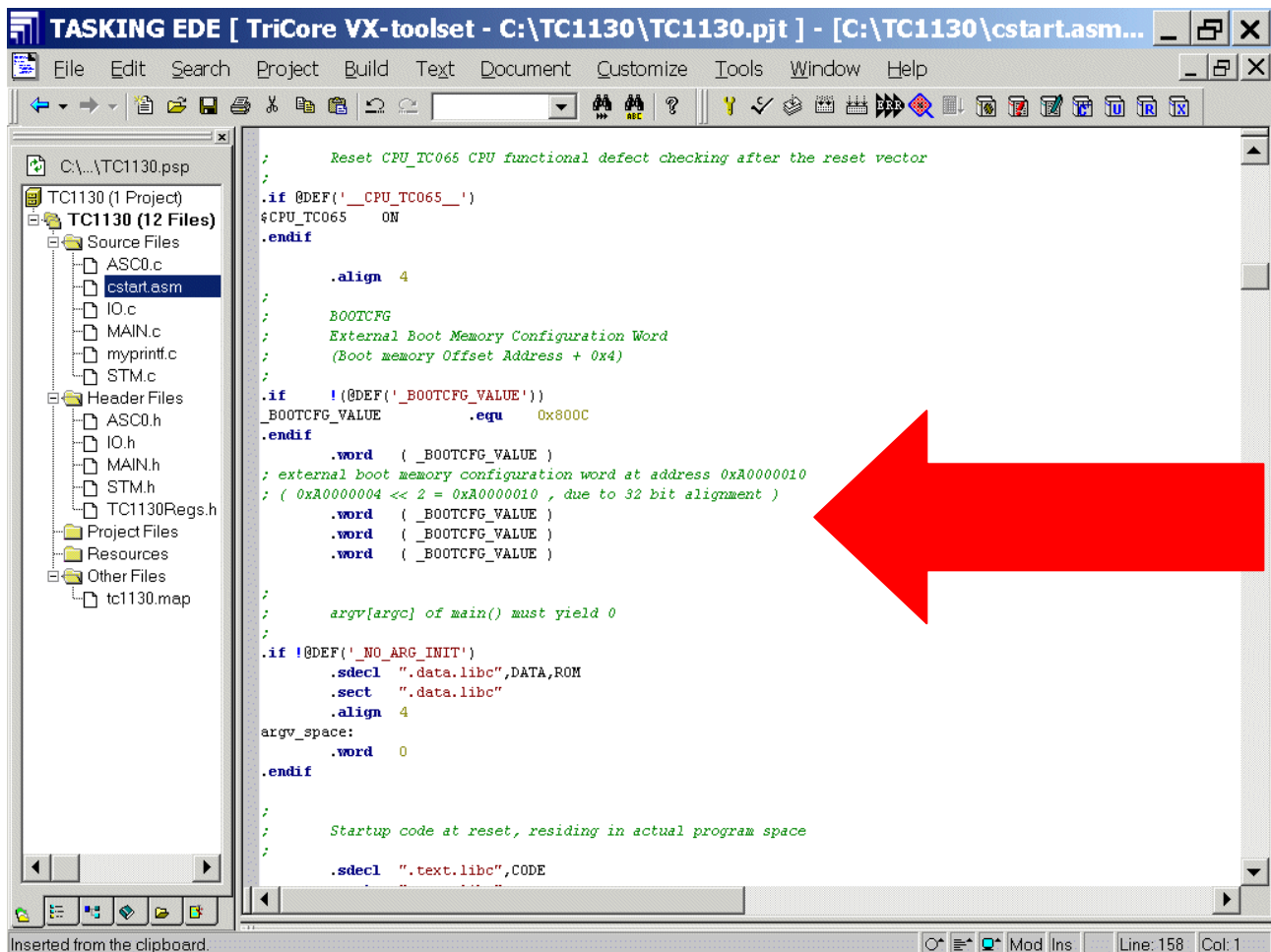
Insert your application specific program:

Double click: `cstart.asm` and insert code:

```

; 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 )

```



The screenshot shows the TASKING EDE IDE interface. The left pane displays the project structure for TC1130, with 'cstart.asm' selected under 'Source Files'. The main editor window shows the assembly code for 'cstart.asm'. A red arrow points to the configuration word definition section of the code.

```

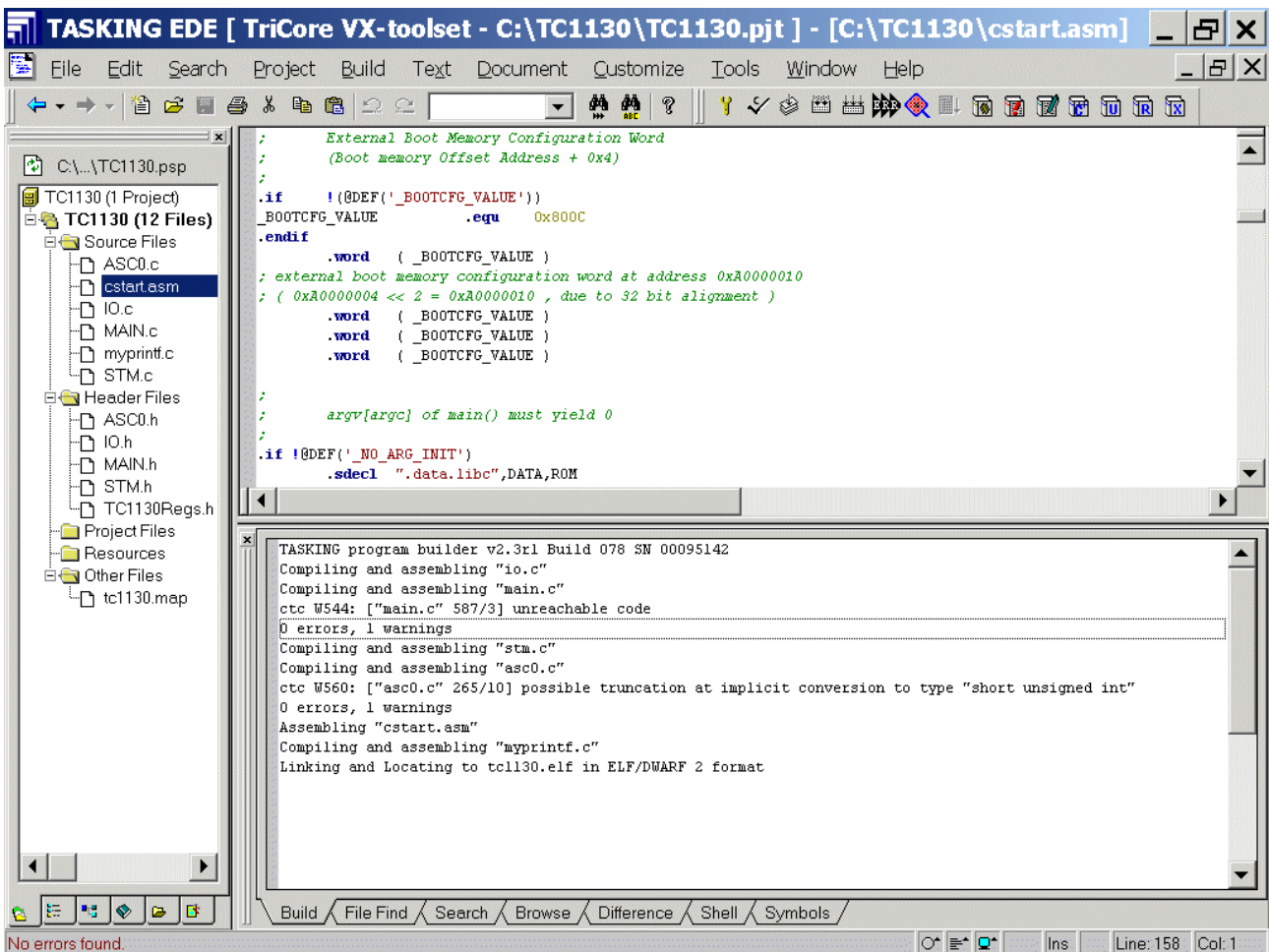
; Reset CPU_TC065 CPU functional defect checking after the reset vector
;
; .if @DEF('_CPU_TC065_')
; CPU_TC065 ON
; .endif
;
; .align 4
;
; BOOTCFG
; External Boot Memory Configuration Word
; (Boot memory Offset Address + 0x4)
;
; .if !(@DEF('_BOOTCFG_VALUE'))
; _BOOTCFG_VALUE .equ 0x800C
; .endif
;
; .word ( _BOOTCFG_VALUE )
; 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 )
;
; argv[argc] of main() must yield 0
;
; .if !@DEF('_NO_ARG_INIT')
; .sdecl ".data.libc",DATA,ROM
; .sect ".data.libc"
; .align 4
; argv_space:
; .word 0
; .endif
;
; Startup code at reset, residing in actual program space
;
; .sdecl ".text.libc",CODE

```

Generate your application program:

Build
Rebuild

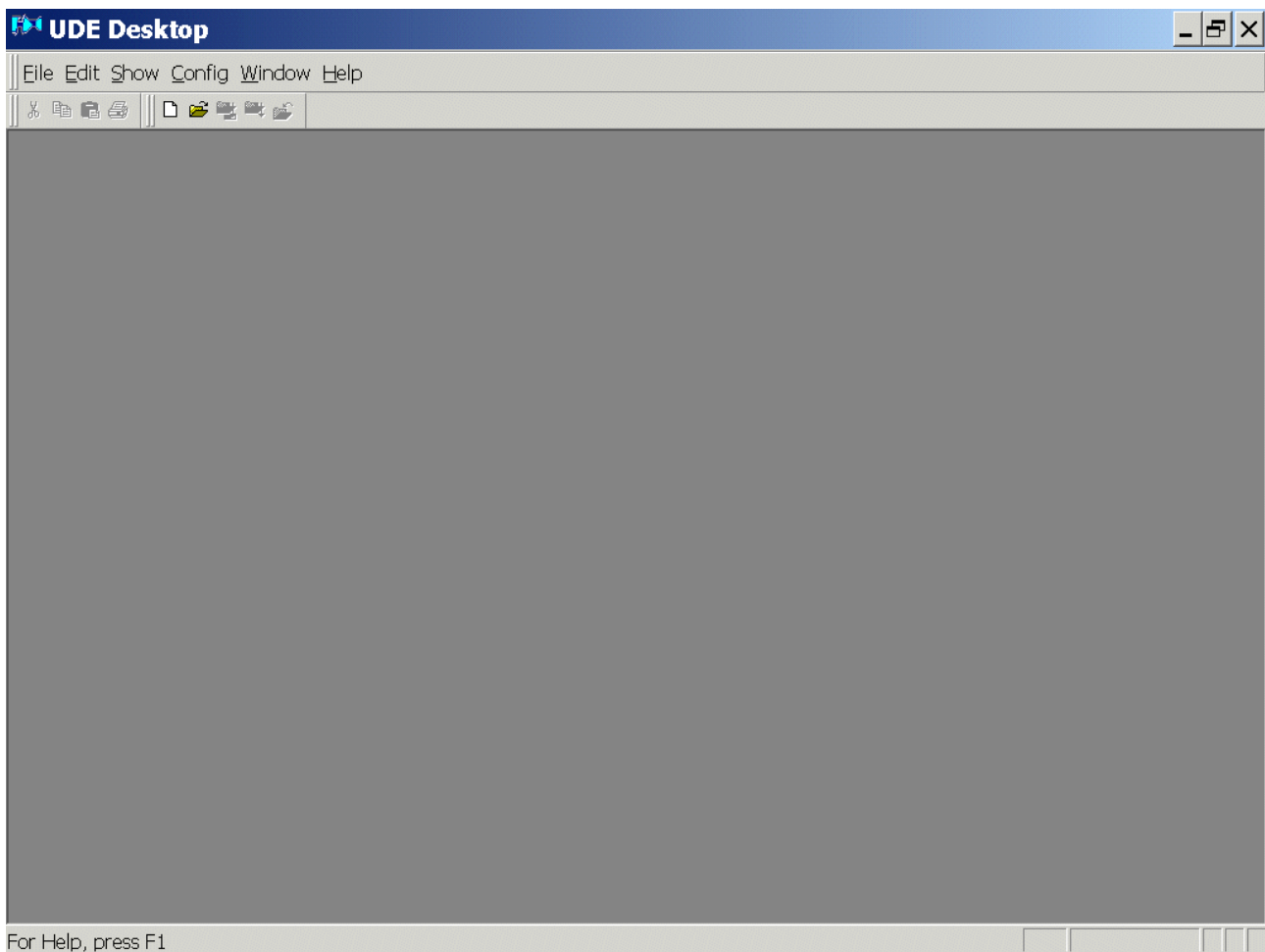
or



Programming and Troubleshooting is now complete. You can now **load** and **run** your program:



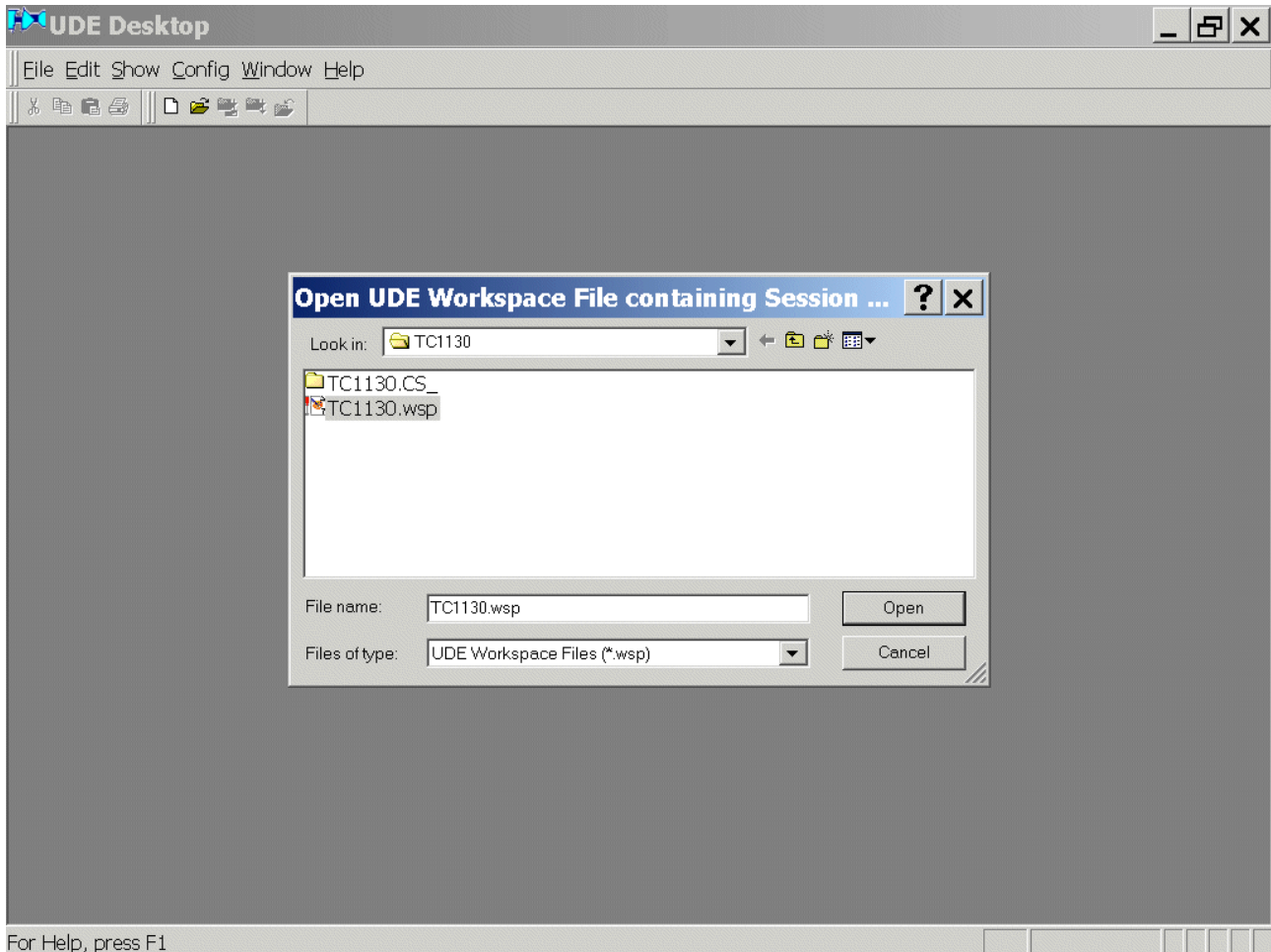
Start pls-Debugger



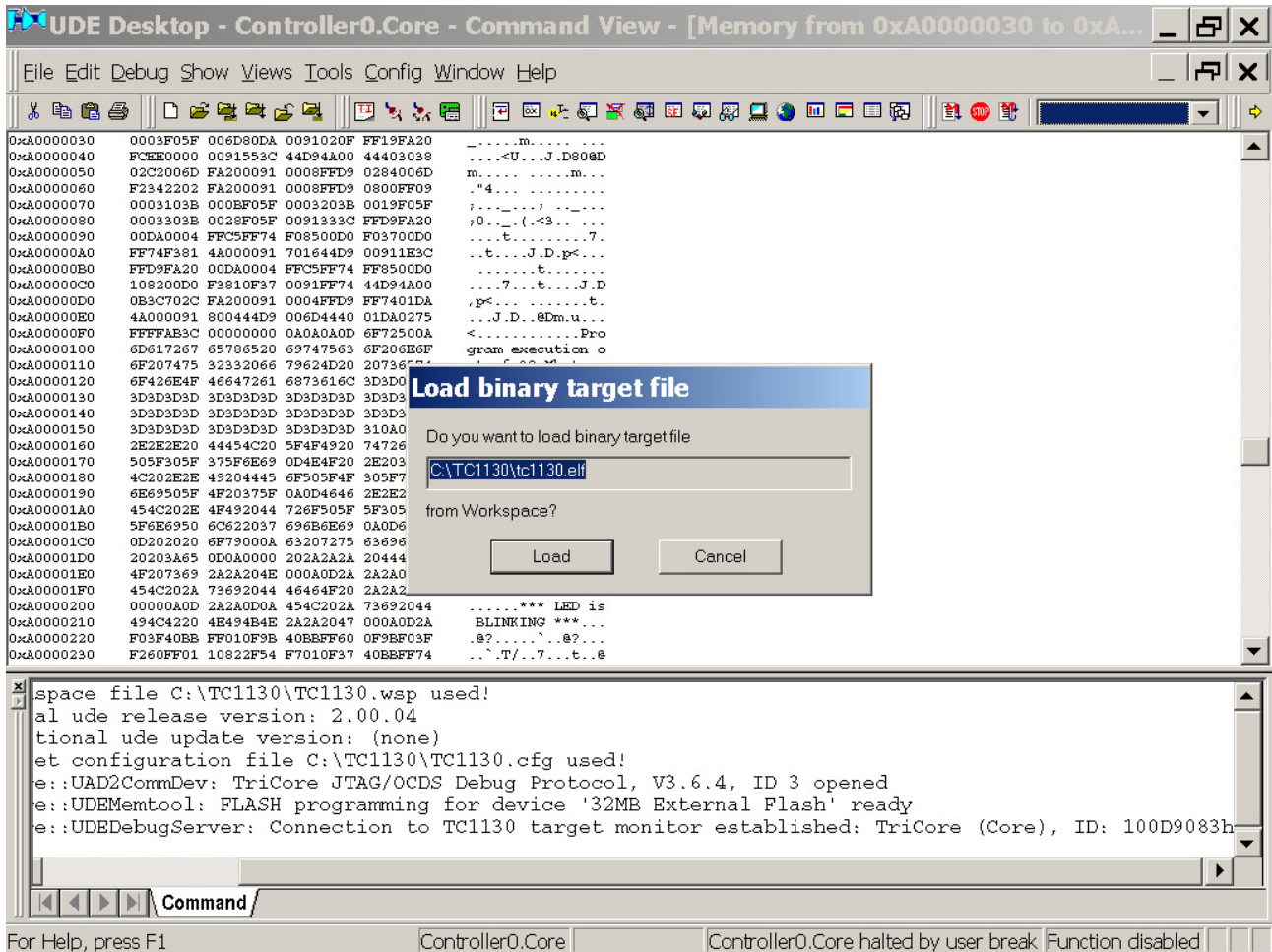
File – Open Workspace

Look in: select C:\TC1130

File name: select TC1130.wsp



Open



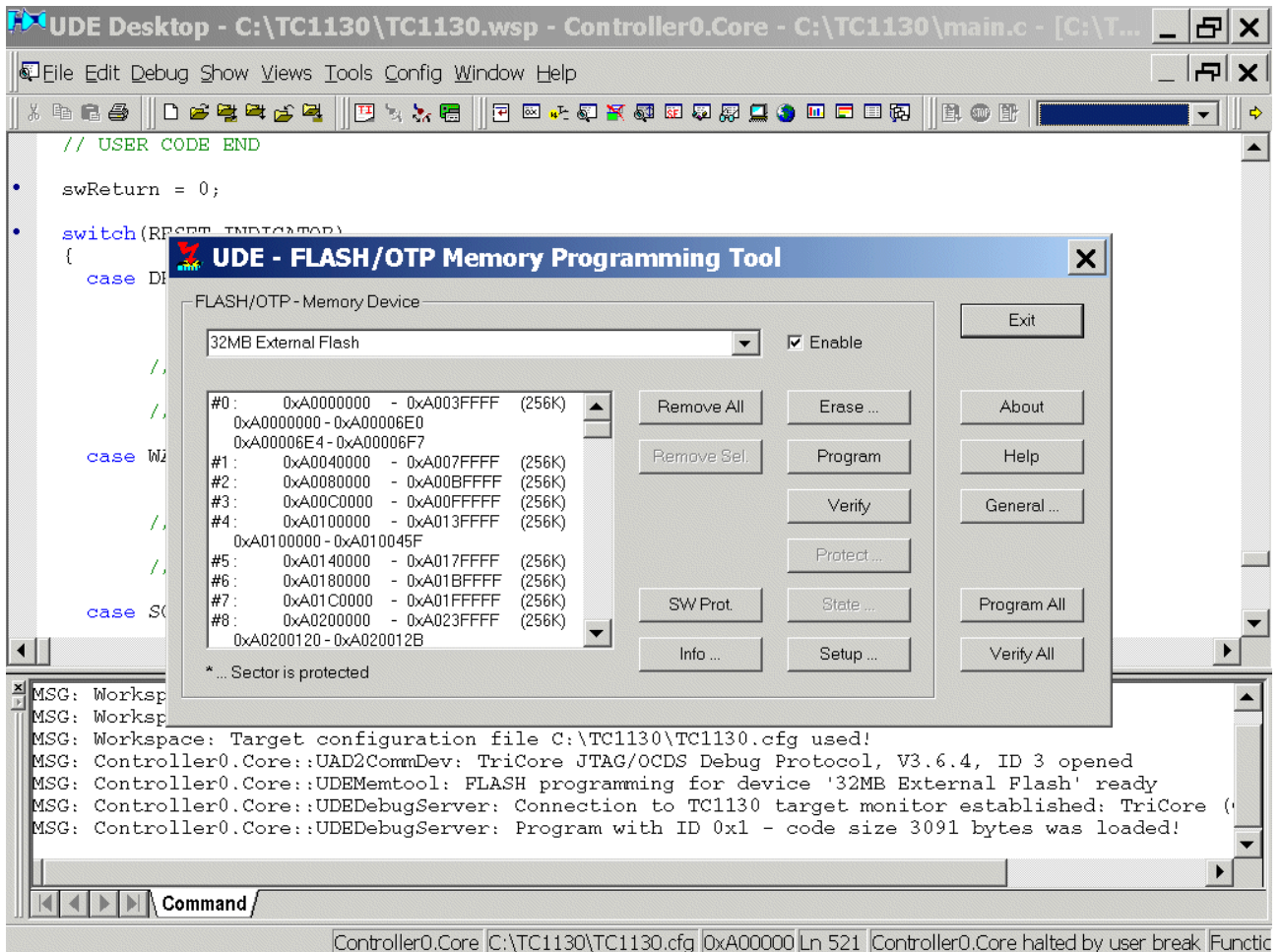
The screenshot shows the UDE Desktop interface for Controller0.Core. The main window displays a memory dump with addresses from 0xA0000030 to 0xA0000230. A dialog box titled "Load binary target file" is open, asking "Do you want to load binary target file" and showing the file path "C:\TC1130\TC1130.elf". Below the dialog, the command view shows the following text:

```

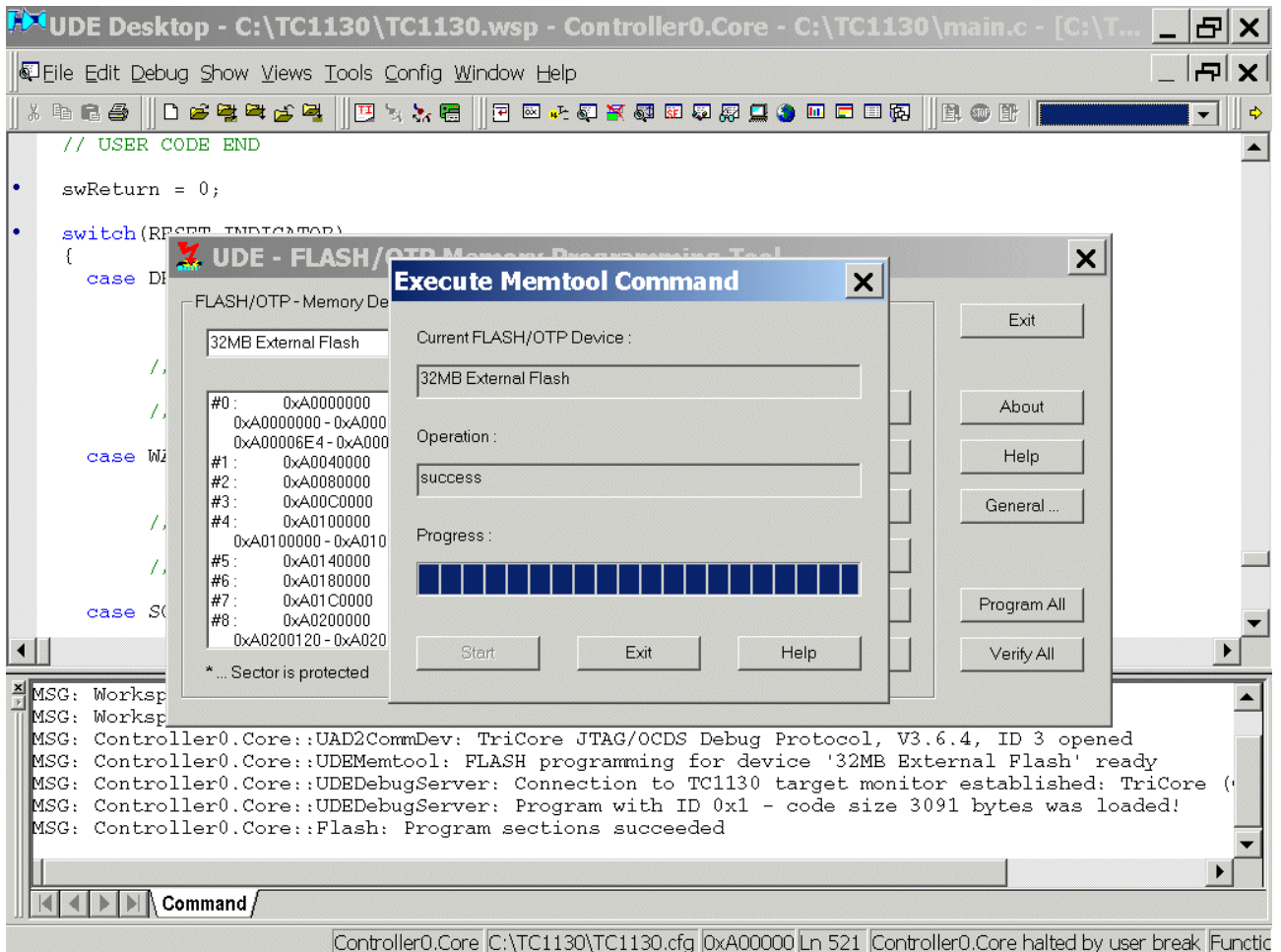
space file C:\TC1130\TC1130.wsp used!
al ude release version: 2.00.04
tional ude update version: (none)
et configuration file C:\TC1130\TC1130.cfg used!
e.:UAD2CommDev: TriCore JTAG/OCDS Debug Protocol, V3.6.4, ID 3 opened
e.:UDEMentool: FLASH programming for device '32MB External Flash' ready
e.:UDEDebugServer: Connection to TC1130 target monitor established: TriCore (Core), ID: 100D9083h
  
```

The status bar at the bottom indicates "Controller0.Core" and "Controller0.Core halted by user break: Function disabled".

Load



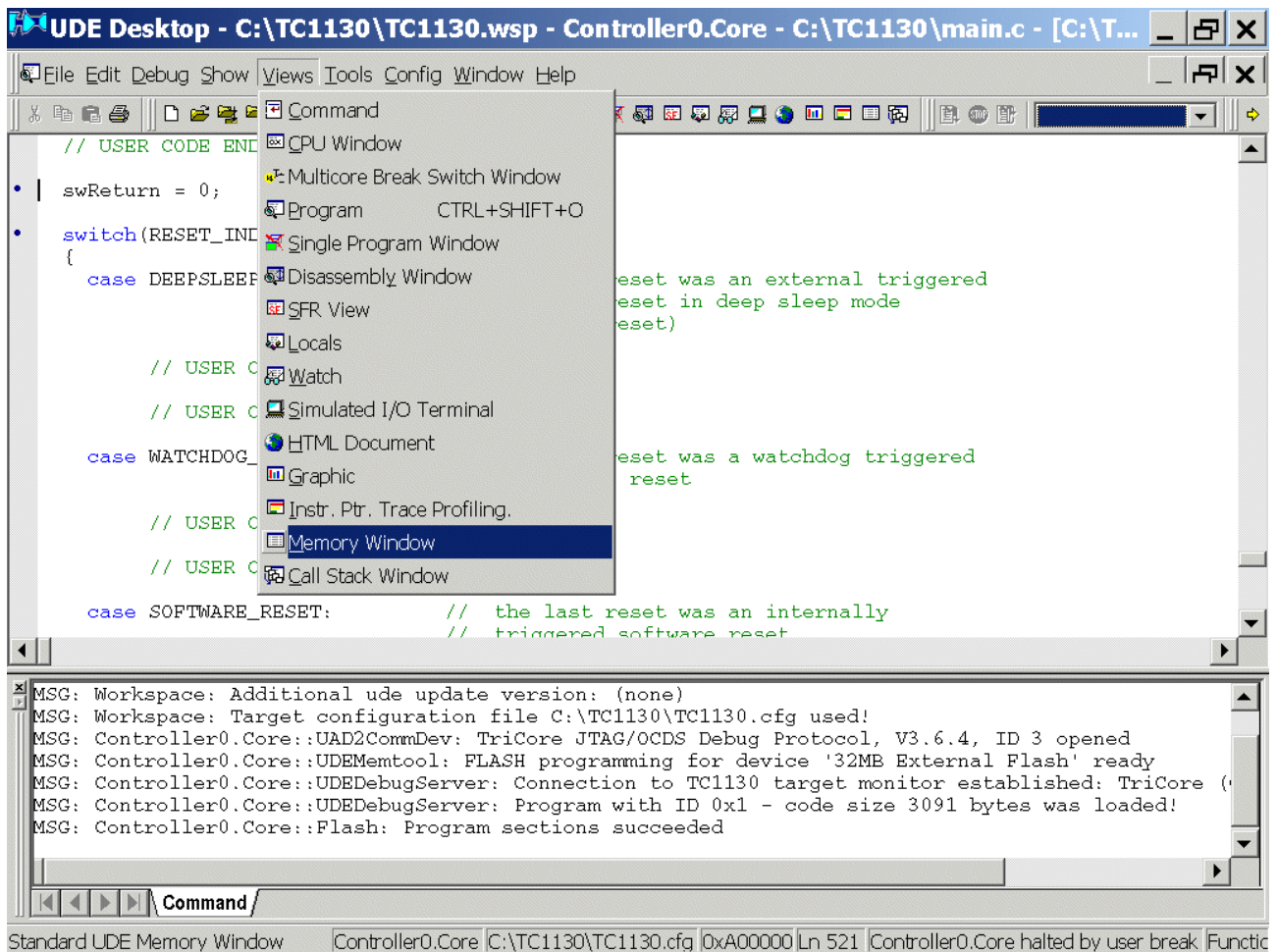
Program All



Exit

Exit

Views – Memory Window

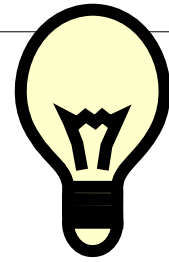


The screenshot shows the UDE Desktop IDE interface. The main window displays a C program with a menu open over it. The menu includes options like CPU Window, Program, Single Program Window, Disassembly Window, SFR View, Locals, Watch, Simulated I/O Terminal, HTML Document, Graphic, Instr. Ptr. Trace Profiling, Memory Window (highlighted), and Call Stack Window. The console at the bottom shows the following messages:

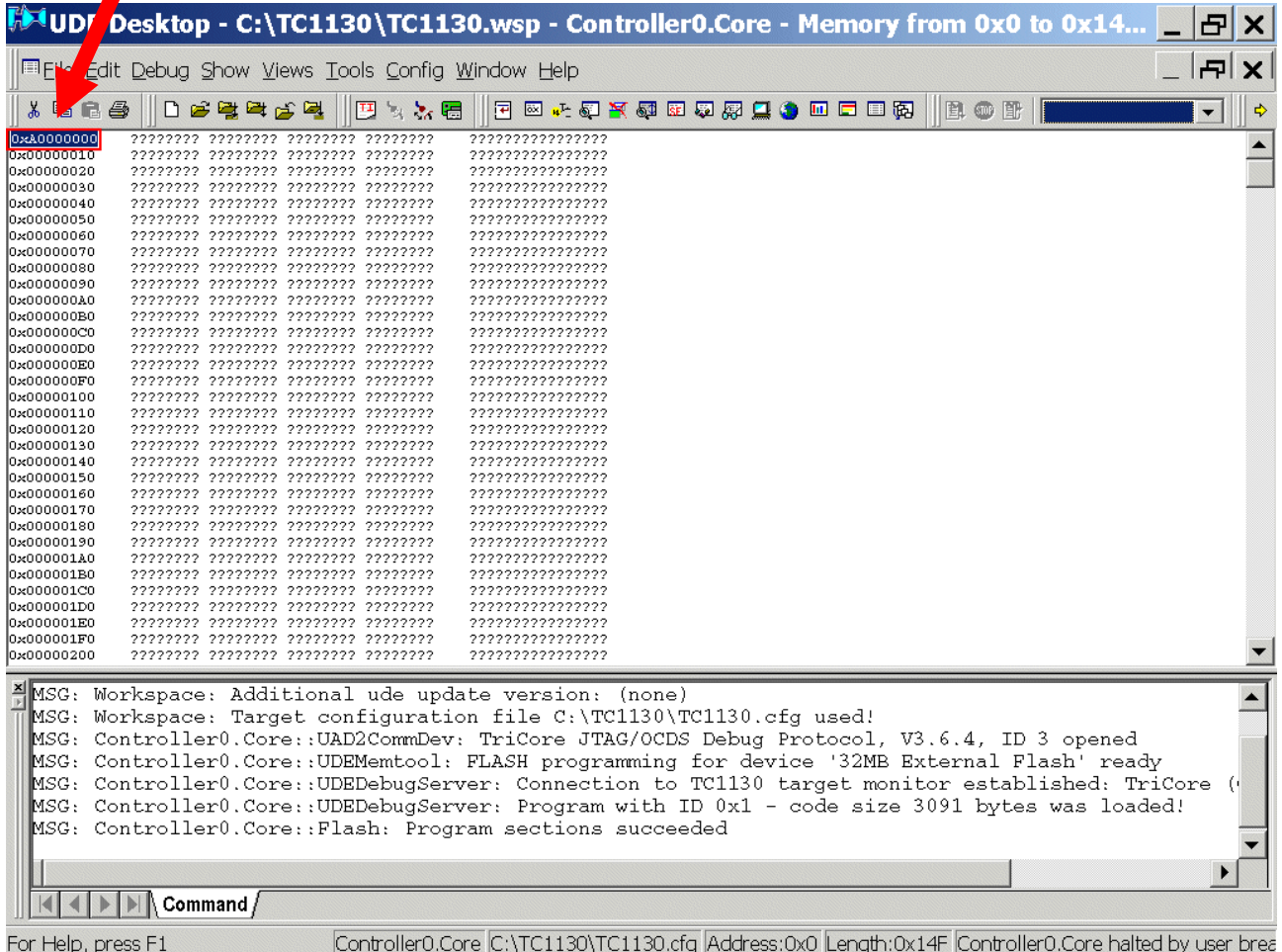
```

MSG: Workspace: Additional ude update version: (none)
MSG: Workspace: Target configuration file C:\TC1130\TC1130.cfg used!
MSG: Controller0.Core::UAD2CommDev: TriCore JTAG/OCDS Debug Protocol, V3.6.4, ID 3 opened
MSG: Controller0.Core::UDEMentool: 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 3091 bytes was loaded!
MSG: Controller0.Core::Flash: Program sections succeeded
  
```

The status bar at the bottom indicates: Standard UDE Memory Window | Controller0.Core | C:\TC1130\TC1130.cfg | 0xA00000 Ln 521 | Controller0.Core halted by user break | Functio



Insert 0xA0000000 <ENTER>

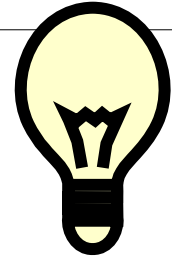


The screenshot shows the UDE Desktop application window titled "C:\TC1130\TC1130.wsp - Controller0.Core - Memory from 0x0 to 0x14...". The main display area shows a list of memory addresses from 0x00000000 to 0x00000200, each followed by several columns of question marks. A red arrow points to the first address, 0xA0000000, which is highlighted in red. Below the memory display is a command window with the following text:

```

MSG: Workspace: Additional ude update version: (none)
MSG: Workspace: Target configuration file C:\TC1130\TC1130.cfg used!
MSG: Controller0.Core::UAD2CommDev: TriCore JTAG/OCDS Debug Protocol, V3.6.4, ID 3 opened
MSG: Controller0.Core::UDEMentool: 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 3091 bytes was loaded!
MSG: Controller0.Core::Flash: Program sections succeeded
  
```

At the bottom of the window, a status bar displays: "For Help, press F1 Controller0.Core C:\TC1130\TC1130.cfg Address:0x0 Length:0x14F Controller0.Core halted by user bree".



As you can see:

The “external-boot-memory-configuration-word” value (0x0000806D) is now @ 0xA0000010:

Correct Value (0x0000806D)
Correct Location (0xA0000010)

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

```

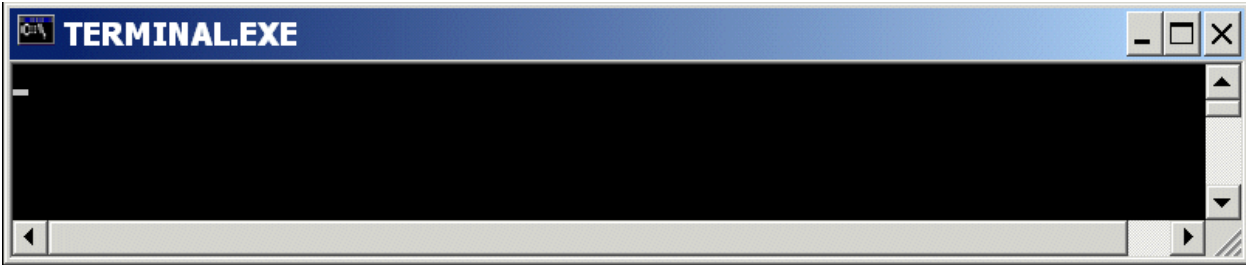
0xA0000000 000081D 0000806D 0000806D 0000806D .....m.m.m.m...
0xA0000010 0000806D FF8500DA 8F8F0014 FF16F01E m.....
0xA0000020 0000803B 000FF05F 0001003B 000BF05F ?.....?.....
0xA0000030 0002003B 0007F05F 0004003B 0003F05F ?.....?.....
0xA0000040 006D80DA 0091020F FF19FA20 FC0E0000 .....m.....
0xA0000050 0091553C 44D94A00 44404004 02C2006D <U...J.D.@Dm...
0xA0000060 FA200091 0008FFD9 0284006D F2342202 .....m....."4.
0xA0000070 FA200091 0008FFD9 0800FF09 0003103B .....
0xA0000080 000BF05F 0003203B 0019F05F 0003303B .....?.....0...
0xA0000090 0028F05F 0091333C FFD9FA20 00DA0004 .....{.<3.....
0xA00000A0 FFC5FF74 F08500D0 F03700D0 FF74F381 .....t.....7...t.
0xA00000B0 4A000091 702244D9 00911E3C FFD9FA20 .....J.D"pc.....
0xA00000C0 00DA0004 FFC5FF74 FF8500D0 108200D0 .....t.....
0xA00000D0 F3810F37 0091FF74 44D94A00 0B3C7038 7...t...J.D8pc.
0xA00000E0 FA200091 0004FFD9 FF7401DA 4A000091 .....t.....J
0xA00000F0 801044D9 006D4440 01DA0275 FFFFAB3C .....D...@Dm.u...<...
0xA0000100 00000000 0A0A0A0D 6F72500A 6D617267 .....Program
0xA0000110 65786520 69747563 6F206E6F 6F207475 execution out o
0xA0000120 32332066 79624D20 20736574 6F426E4F f 32 Mbytes OnBo
0xA0000130 46647261 6873616C 3D3D0A0D 3D3D3D3D ardFlash..=====
0xA0000140 3D3D3D3D 3D3D3D3D 3D3D3D3D 3D3D3D3D =====
0xA0000150 3D3D3D3D 3D3D3D3D 3D3D3D3D 3D3D3D3D =====
0xA0000160 3D3D3D3D 3D3D3D3D 310A0D3D 2E2E2E20 .....1...
0xA0000170 44454C20 5F4F4920 74726F50 505F305F LED IO_Port_0_P
0xA0000180 375F6E69 0D4E4F20 2E20320A 4C202E2E in_7 ON..2...L
0xA0000190 49204445 6F505F4F 305F7472 6E69505F ED IO_Port_0_Pin
0xA00001A0 4F20375F 0A0D4646 2E2E2033 454C202E _7 OFF..3...LE
0xA00001B0 4F492044 726F505F 5F305F74 5F6E6950 D IO_Port_0_Pin_
0xA00001C0 6C622037 696B6E69 0A0D676E 0D202020 7 blinking...
0xA00001D0 6F79000A 63207275 63696F68 20203A65 ..your choice:
0xA00001E0 0D0A0000 202A2A2A 2044454C 4F207369 .....** LED is O
0xA00001F0 2A2A204E 000A0D2A 2A2A0D0A 454C202A N ***.....** LE
0xA0000200 73692044 46464F20 2A2A2A20 0000A0AD D is OFF ***...
  
```

```

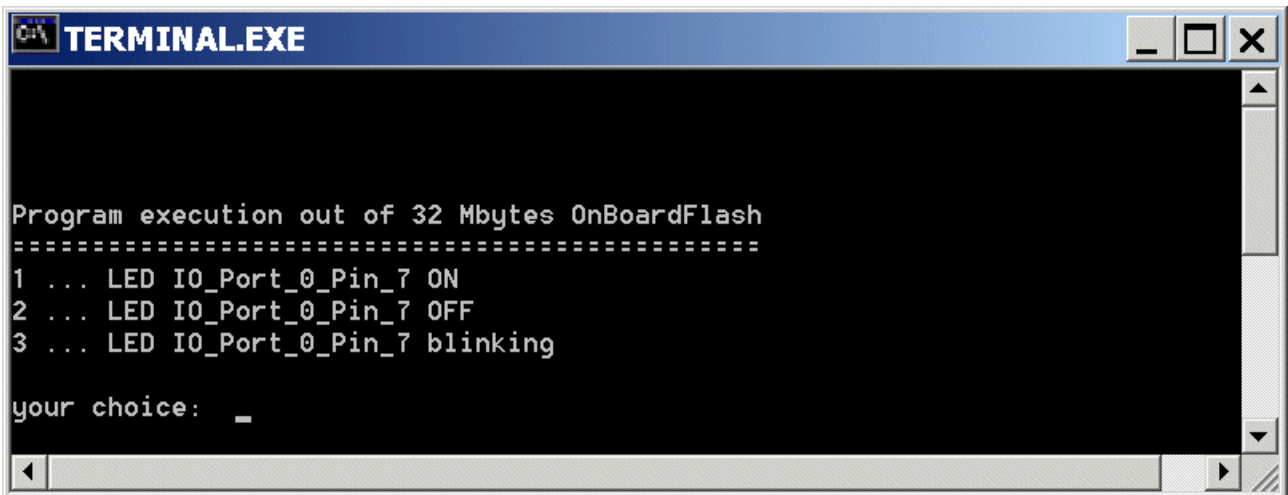
MSG: Workspace: Additional ude update version: (none)
MSG: Workspace: Target configuration file C:\TC1130\TC1130.cfg used!
MSG: Controller0.Core::UAD2CommDev: TriCore JTAG/OCDS Debug Protocol, V3.6.4, ID 3 opened
MSG: Controller0.Core::UDEMentool: FLASH programming for device '32MB External Flash' ready
MSG: Controller0.Core::UDEDebugServer: Connection to TC1130 target monitor established: TriCore (
MSG: Controller0.Core::UDEDebugServer: Program with ID 0x1 - code size 3091 bytes was loaded!
MSG: Controller0.Core::Flash: Program sections succeeded
  
```

→ 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):



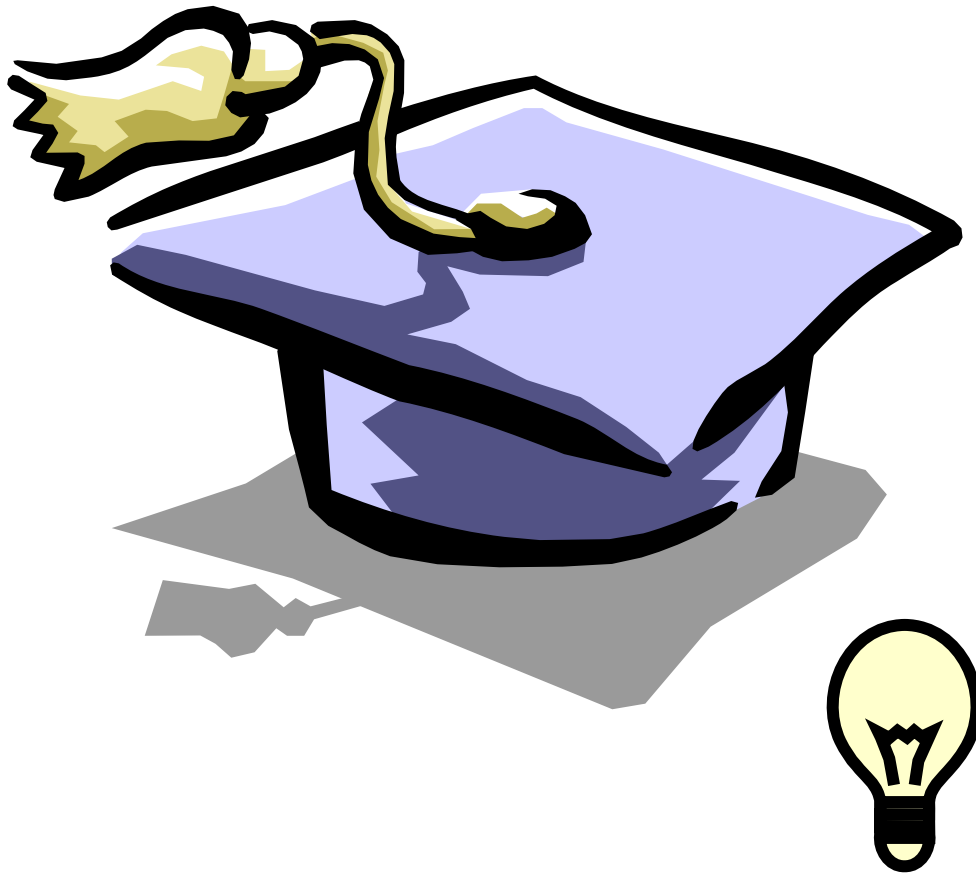
Power-On the Board and see the result:



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

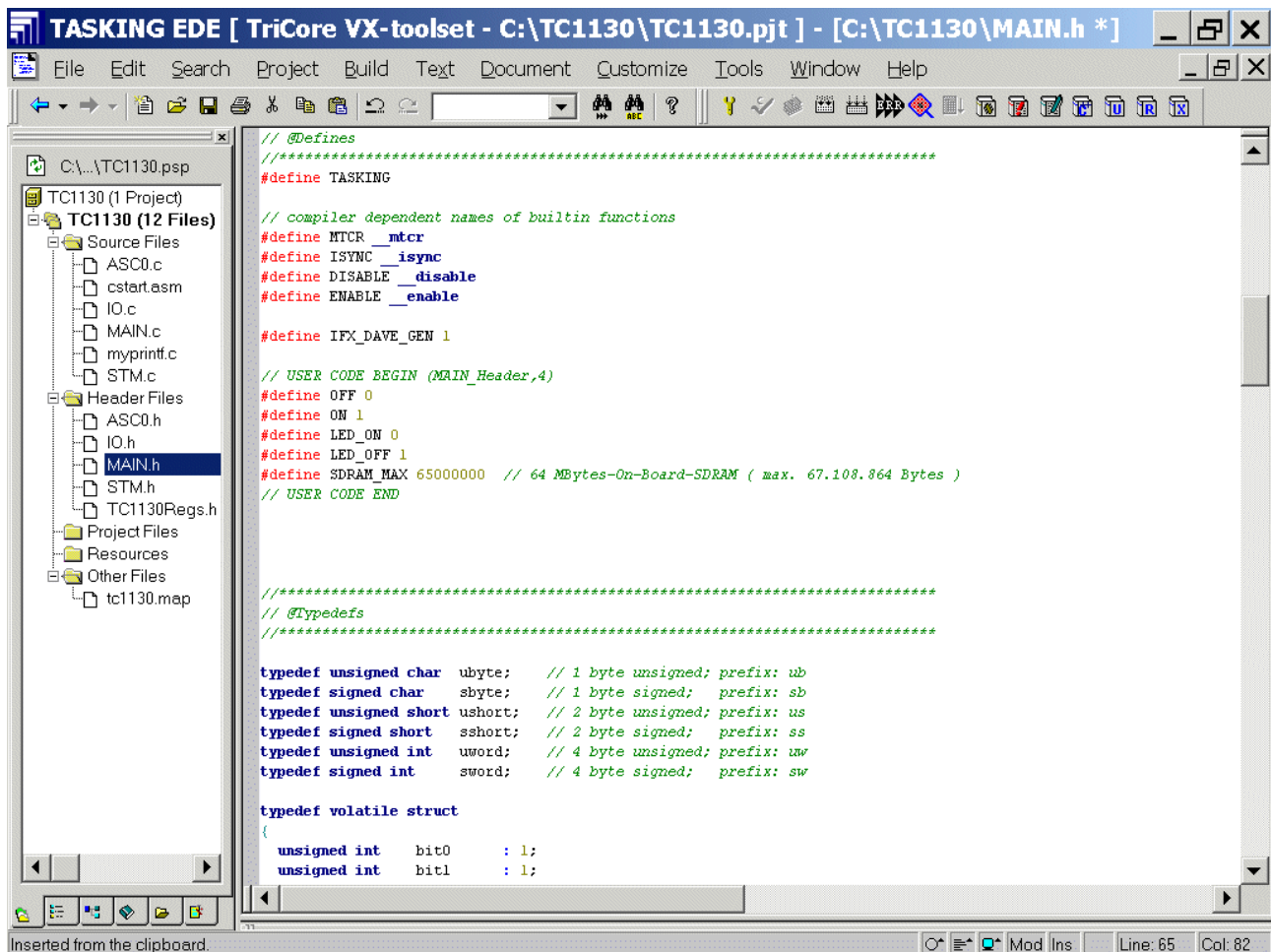
File name: select TC1130.psp

Open

Insert your application specific program:

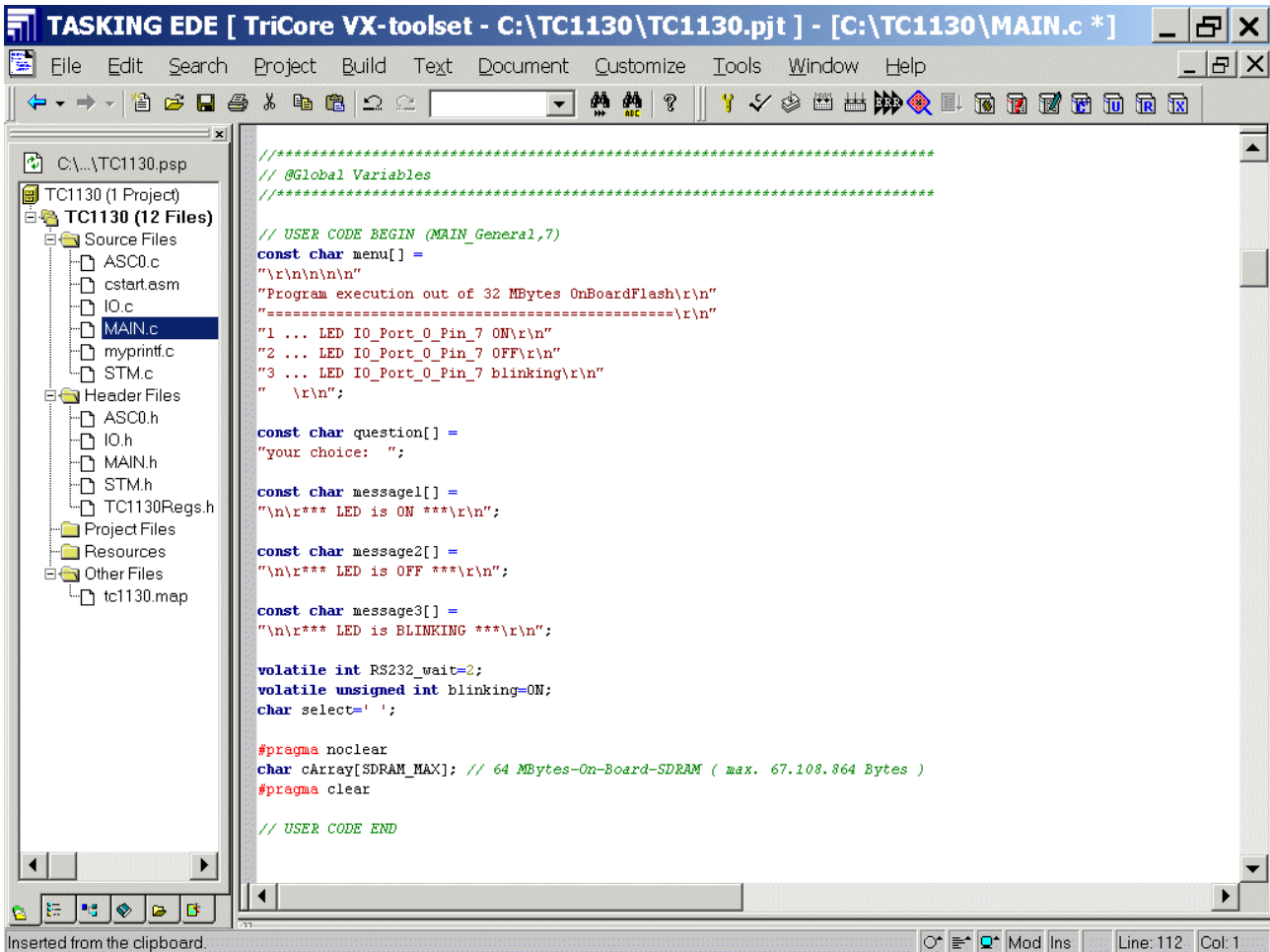
Double click: Main.h and insert the following Defines:

```
#define SDRAM_MAX 65000000 // 64 MBytes-On-Board-SDRAM ( max. 67.108.864 Bytes )
```



Double click: **MAIN.C** and insert Global Variables:

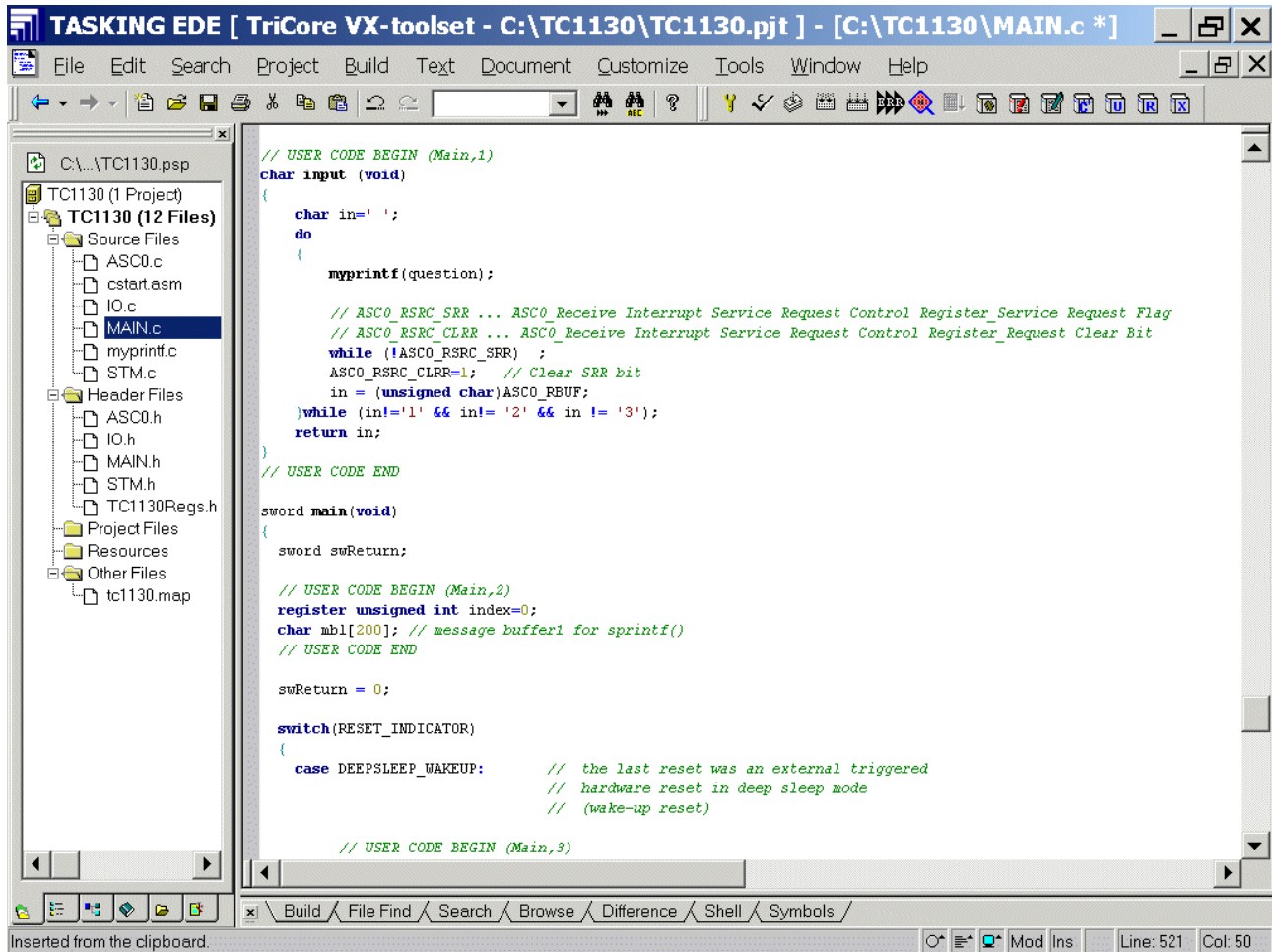
```
#pragma noclear
char cArray[SDRAM_MAX]; // 64 MBytes-On-Board-SDRAM ( max. 67.108.864 Bytes )
#pragma clear
```



```
TASKING EDE [ TriCore VX-toolset - C:\TC1130\TC1130.pjt ] - [C:\TC1130\MAIN.c *]
File Edit Search Project Build Text Document Customize Tools Window Help
C:\TC1130.psp
TC1130 (1 Project)
  TC1130 (12 Files)
    Source Files
      ASC0.c
      cstart.asm
      IO.c
      MAIN.c
      myprintf.c
      STM.c
    Header Files
      ASC0.h
      IO.h
      MAIN.h
      STM.h
      TC1130Regs.h
    Project Files
    Resources
    Other Files
      tc1130.map
//*****
// @Global Variables
//*****
// USER CODE BEGIN (MAIN_General,7)
const char menu[] =
"\r\n\r\n\r\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";
const char question[] =
"your choice: ";
const char message1[] =
"\n\r*** LED is ON ***\r\n";
const char message2[] =
"\n\r*** LED is OFF ***\r\n";
const char message3[] =
"\n\r*** LED is BLINKING ***\r\n";
volatile int RS232_wait=2;
volatile unsigned int blinking=0N;
char select=' ';
#pragma noclear
char cArray[SDRAM_MAX]; // 64 MBytes-On-Board-SDRAM ( max. 67.108.864 Bytes )
#pragma clear
// USER CODE END
Inserted from the clipboard. Mod Ins Line: 112 Col: 1
```

Double click: **MAIN.C** and **insert** the following code in the **main** function:

```
register unsigned int index=0;
char mb1[200]; // message buffer1 for sprintf()
```



```

// USER CODE BEGIN (Main,1)
char input (void)
{
    char in=' ';
    do
    {
        myprintf(question);

        // ASC0_RSRC_SRR ... ASC0_Receive Interrupt Service Request Control Register_Service Request Flag
        // ASC0_RSRC_CLR ... ASC0_Receive Interrupt Service Request Control Register_Request Clear Bit
        while (!ASC0_RSRC_SRR) ;
        ASC0_RSRC_CLR=1; // Clear SRR bit
        in = (unsigned char)ASC0_RBUF;
    }while (in!='1' && in!='2' && in != '3');
    return in;
}
// USER CODE END

sword main(void)
{
    sword swReturn;

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

    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)
    
```

Double click: **MAIN.C** and **insert** the following code in the **main** function:

```

//****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
    }
}

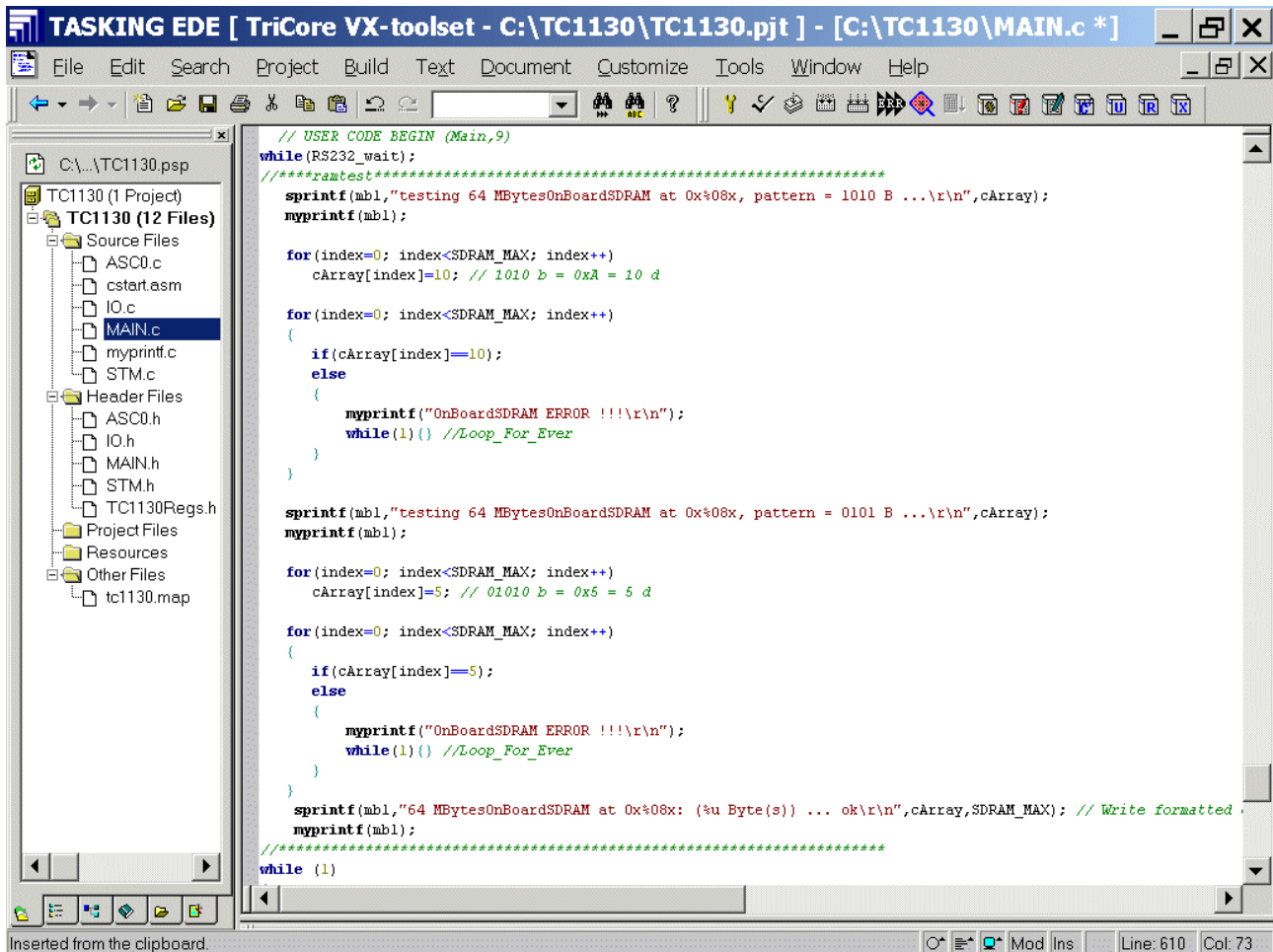
sprintf(mb1,"testing 64 MBytesOnBoardSDRAM at 0x%08x, pattern = 0101 B
...\r\n",cArray);
myprintf(mb1);

for(index=0; index<SDRAM_MAX; index++)
    cArray[index]=5; // 0101 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)) ...
ok\r\n",cArray,SDRAM_MAX); // Write formatted data to string mb
myprintf(mb1);
//*****

```



```

// USER CODE BEGIN (Main,9)
while(RS232_wait);
//*****ramtest*****
sprintf(mbl,"testing 64 MBytesOnBoardSDRAM at 0x%08x, pattern = 1010 B ...\\r\\n",cArray);
myprintf(mbl);

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

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(mbl,"64 MBytesOnBoardSDRAM at 0x%08x: (%u Byte(s)) ... ok\\r\\n",cArray,SDRAM_MAX); // Write formatted .
myprintf(mbl);
//*****
while (1)
    
```

Note:

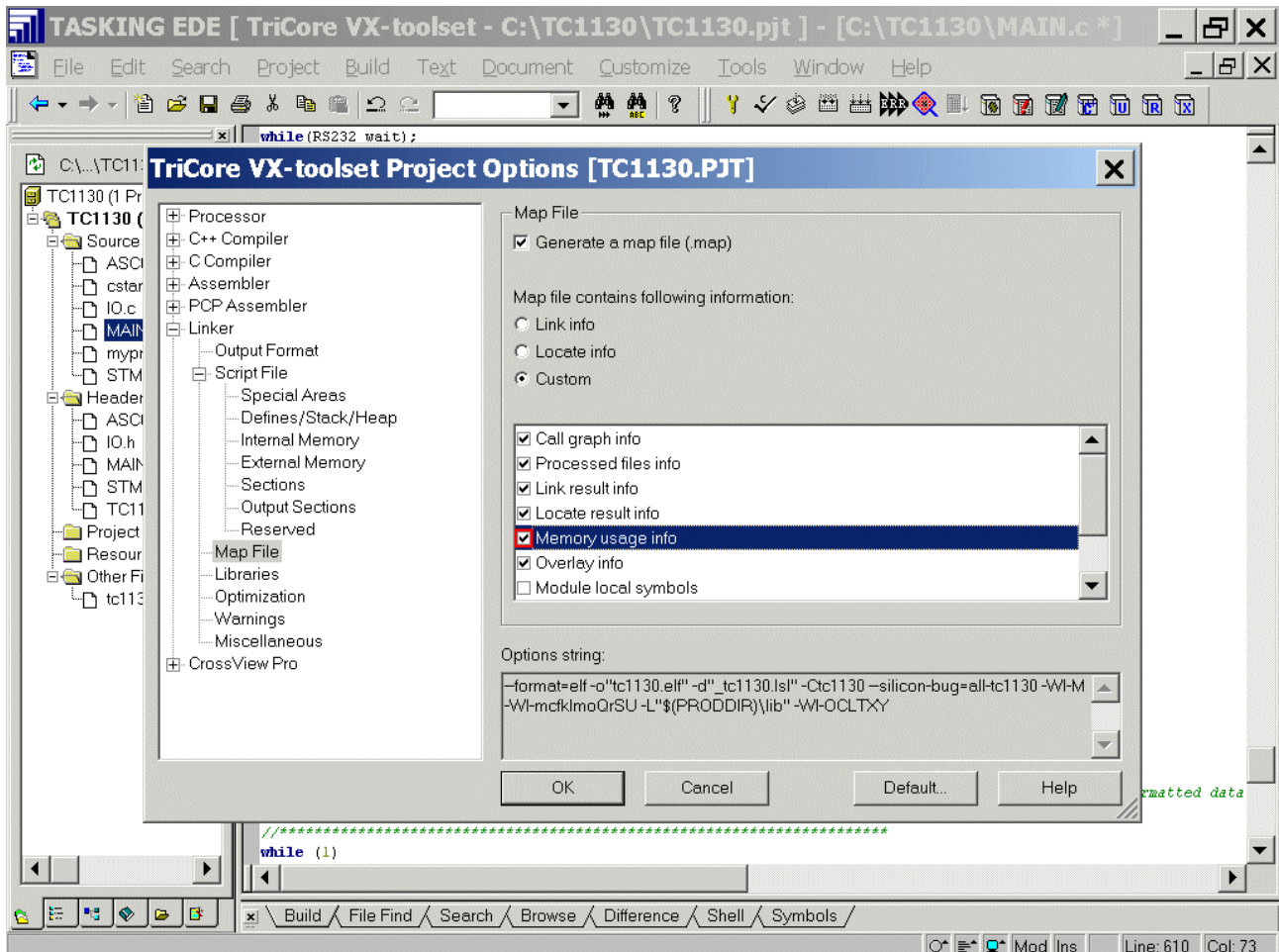
Testing 65.000.000 Bytes take about 11 minutes 13 seconds.
Therefore we suggest to make the array (cArray) smaller
(e.g. `#define SDRAM_MAX 100`).



Configure Linker – Control:

Project – Project Options

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

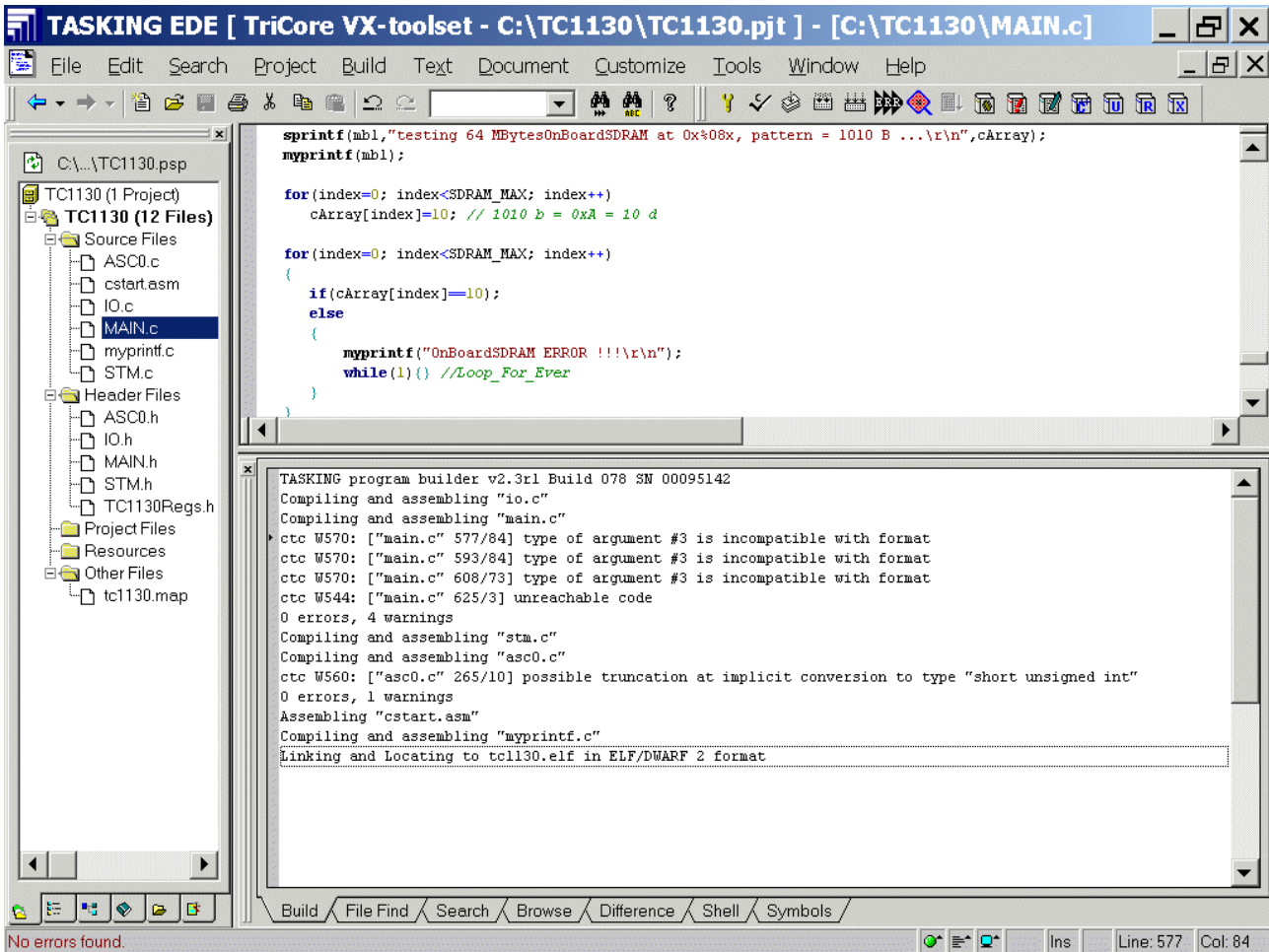


OK

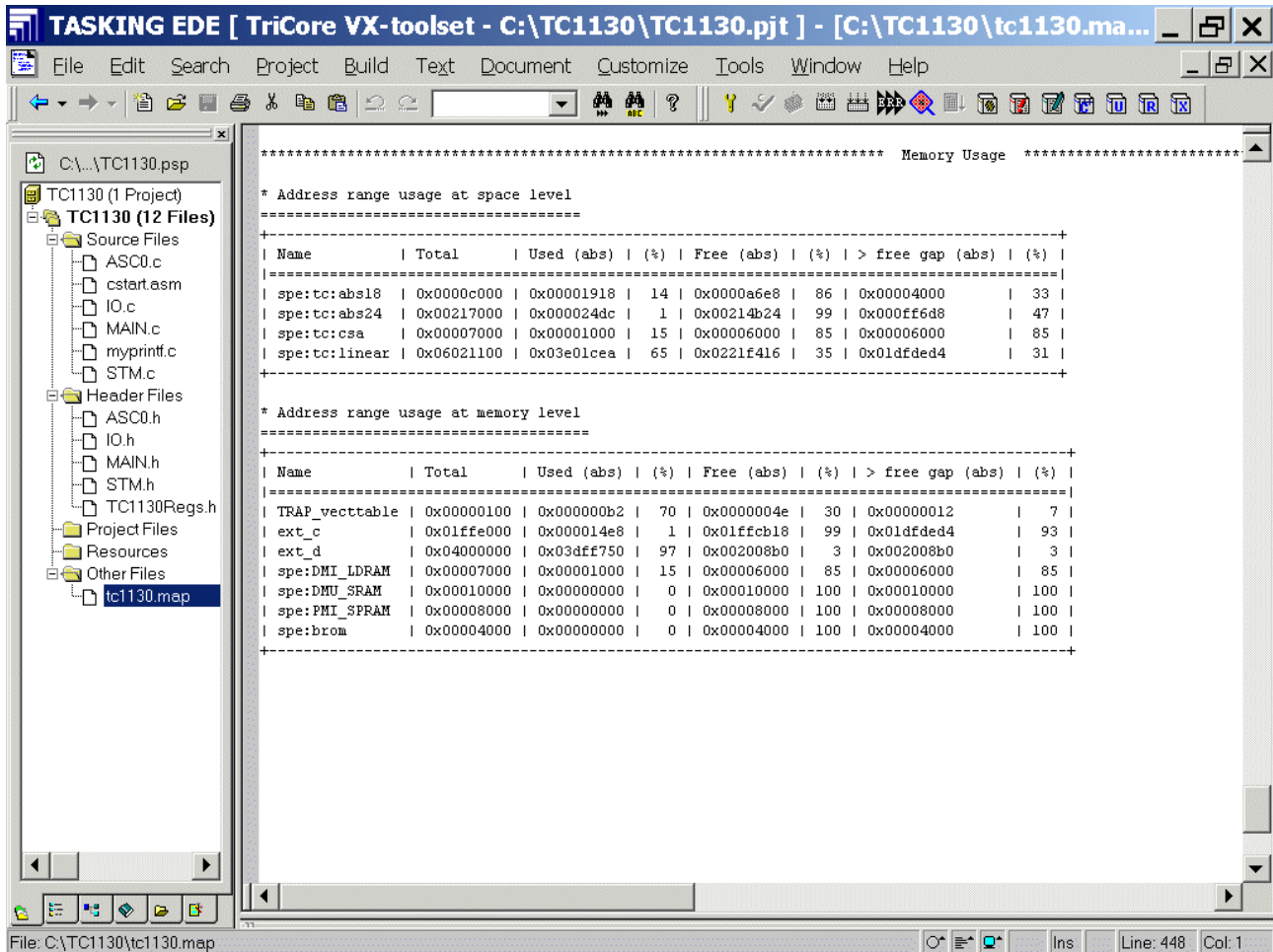
Generate your application program:

Build
Rebuild

or



See the Memory Usage:



```

***** Memory Usage *****
* Address range usage at space level
=====
| Name          | Total      | Used (abs) | (%) | Free (abs) | (%) | > free gap (abs) | (%) |
=====
| spe:tc:abs18  | 0x0000c000 | 0x00001918 | 14  | 0x0000a6e8 | 86  | 0x00004000       | 33  |
| spe:tc:abs24  | 0x00217000 | 0x000024dc | 1  | 0x00214b24 | 99  | 0x000ff6d8       | 47  |
| spe:tc:csa    | 0x00007000 | 0x00001000 | 15  | 0x00006000 | 85  | 0x00006000       | 85  |
| spe:tc:linear | 0x06021100 | 0x03e01cea | 65  | 0x0221f416 | 35  | 0x01dfded4       | 31  |
=====
* Address range usage at memory level
=====
| Name          | Total      | Used (abs) | (%) | Free (abs) | (%) | > free gap (abs) | (%) |
=====
| TRAP_vecttable | 0x00000100 | 0x000000b2 | 70  | 0x0000004e | 30  | 0x00000012       | 7  |
| ext_c         | 0x01ffe000 | 0x000014e8 | 1  | 0x01ffc18  | 99  | 0x01dfded4       | 93  |
| ext_d         | 0x04000000 | 0x03dff750 | 97  | 0x002008b0 | 3  | 0x002008b0       | 3  |
| spe:DMI_LDRAM | 0x00007000 | 0x00001000 | 15  | 0x00006000 | 85  | 0x00006000       | 85  |
| spe:DMU_SRAM  | 0x00010000 | 0x00000000 | 0  | 0x00010000 | 100 | 0x00010000       | 100 |
| spe:PMI_SPRAM | 0x00008000 | 0x00000000 | 0  | 0x00008000 | 100 | 0x00008000       | 100 |
| spe:brom      | 0x00004000 | 0x00000000 | 0  | 0x00004000 | 100 | 0x00004000       | 100 |
=====

```

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

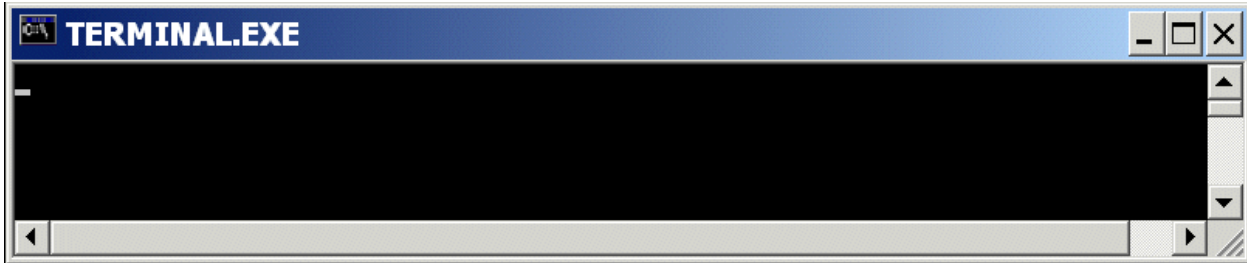
Exit

File – Close Workspace

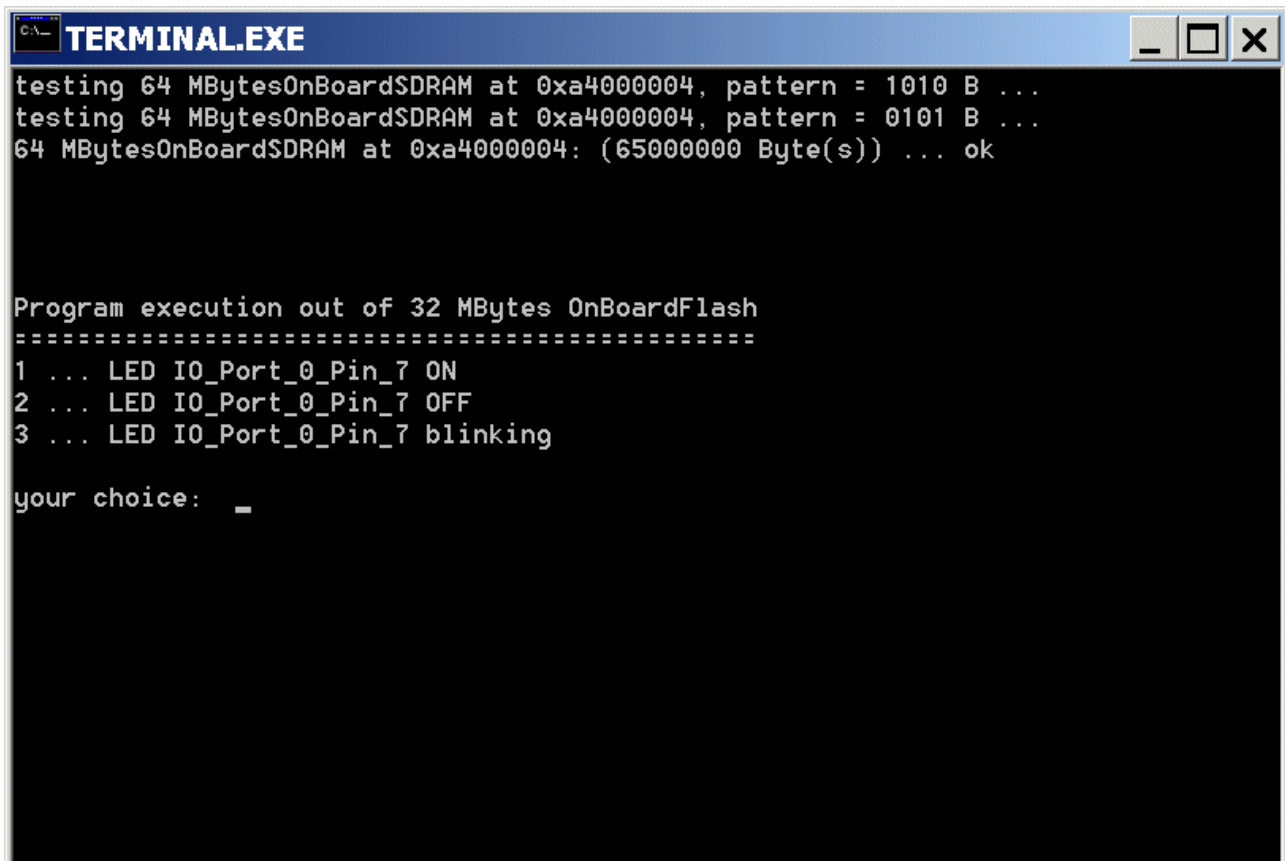
Yes

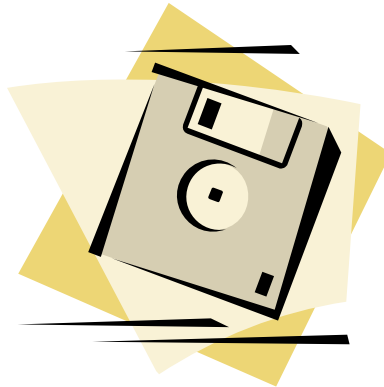
File – Exit

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

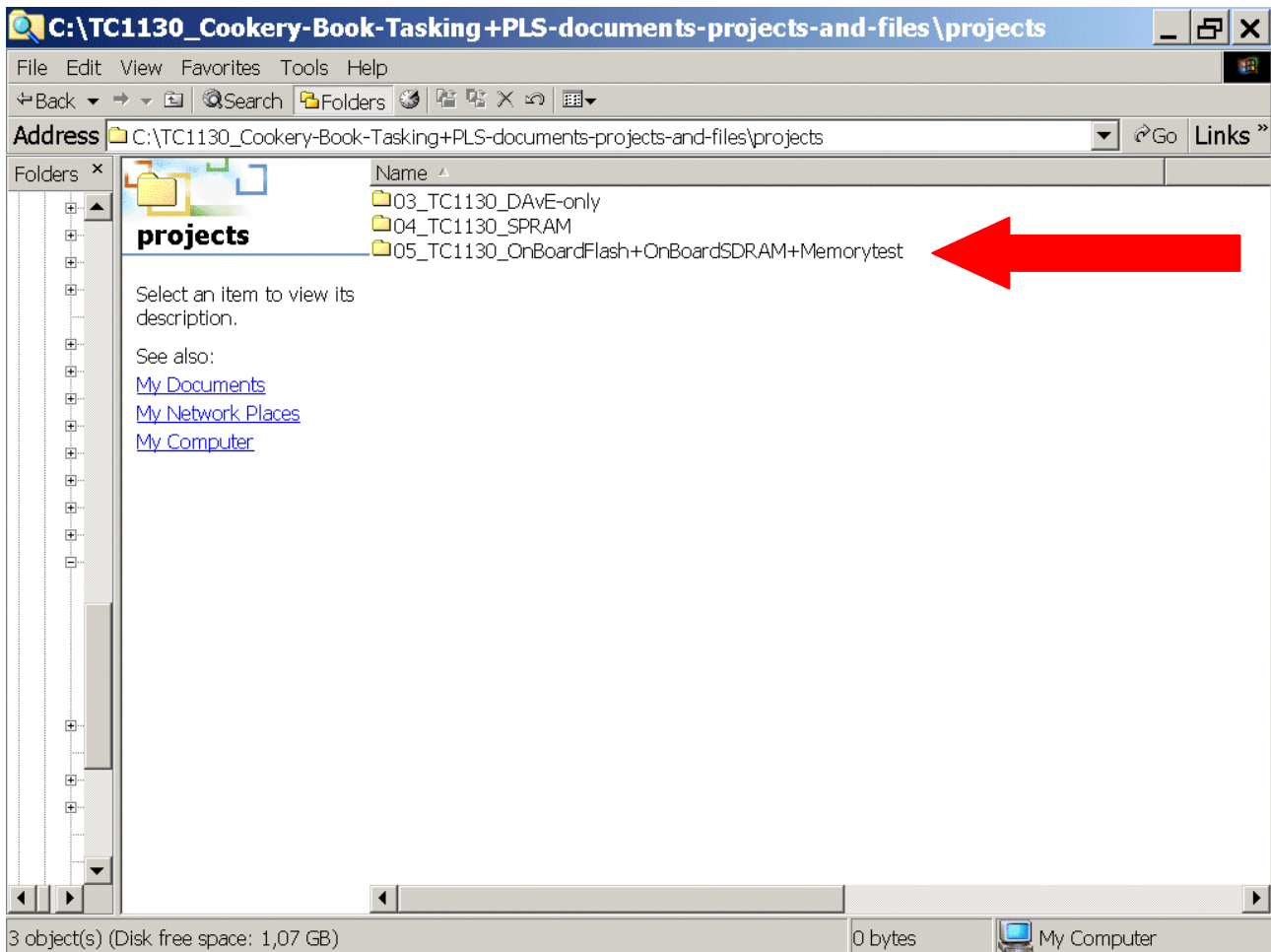


Power-On the Board and see the result:





We recommend now to **copy and store** your project-directory “C:\TC1130” to “05_TC1130_OnBoardFlash+OnBoardSDRAM+Memorytest”:

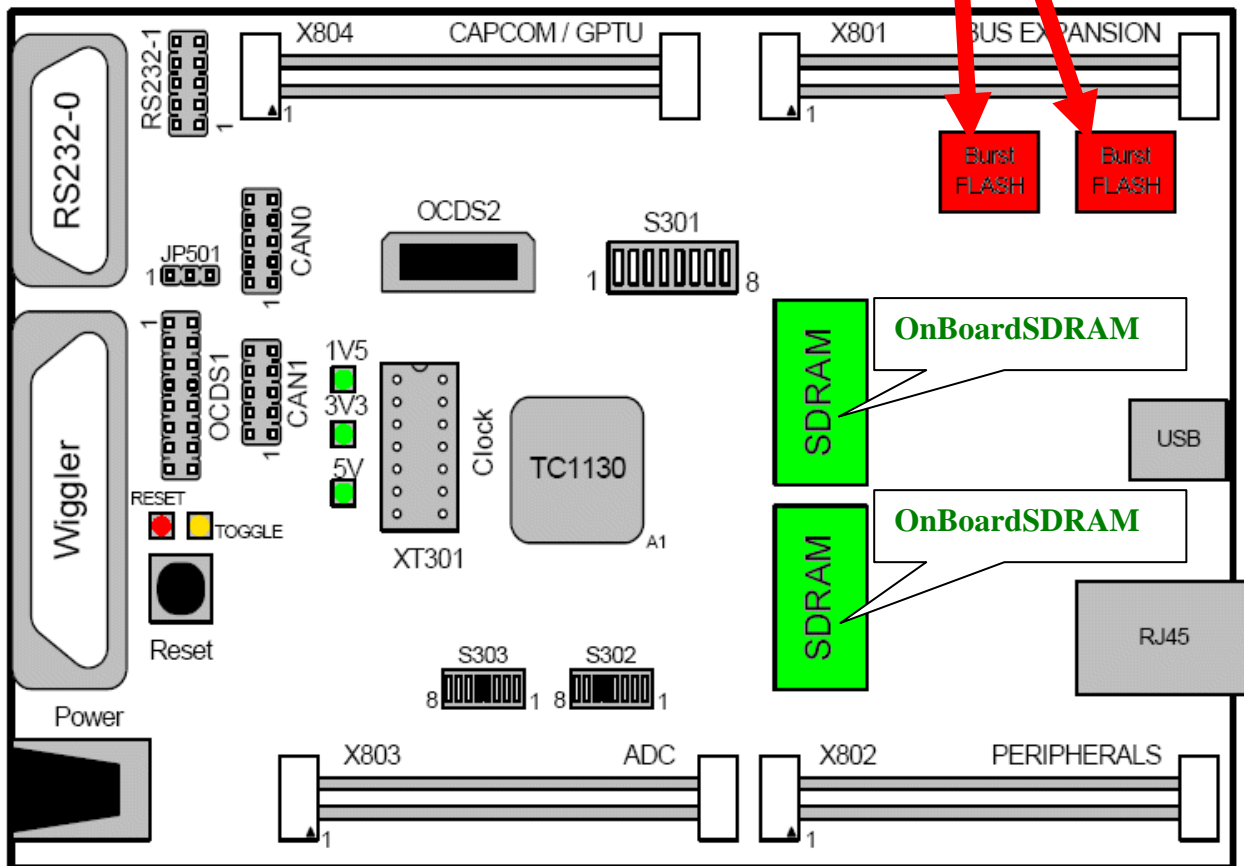
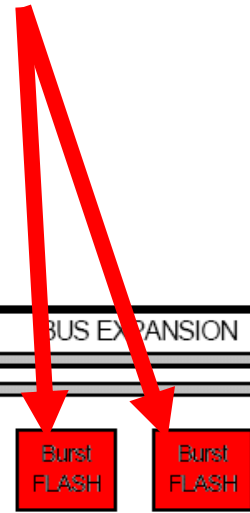


6.) Using of the TASKING - EDE Development Tools:



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

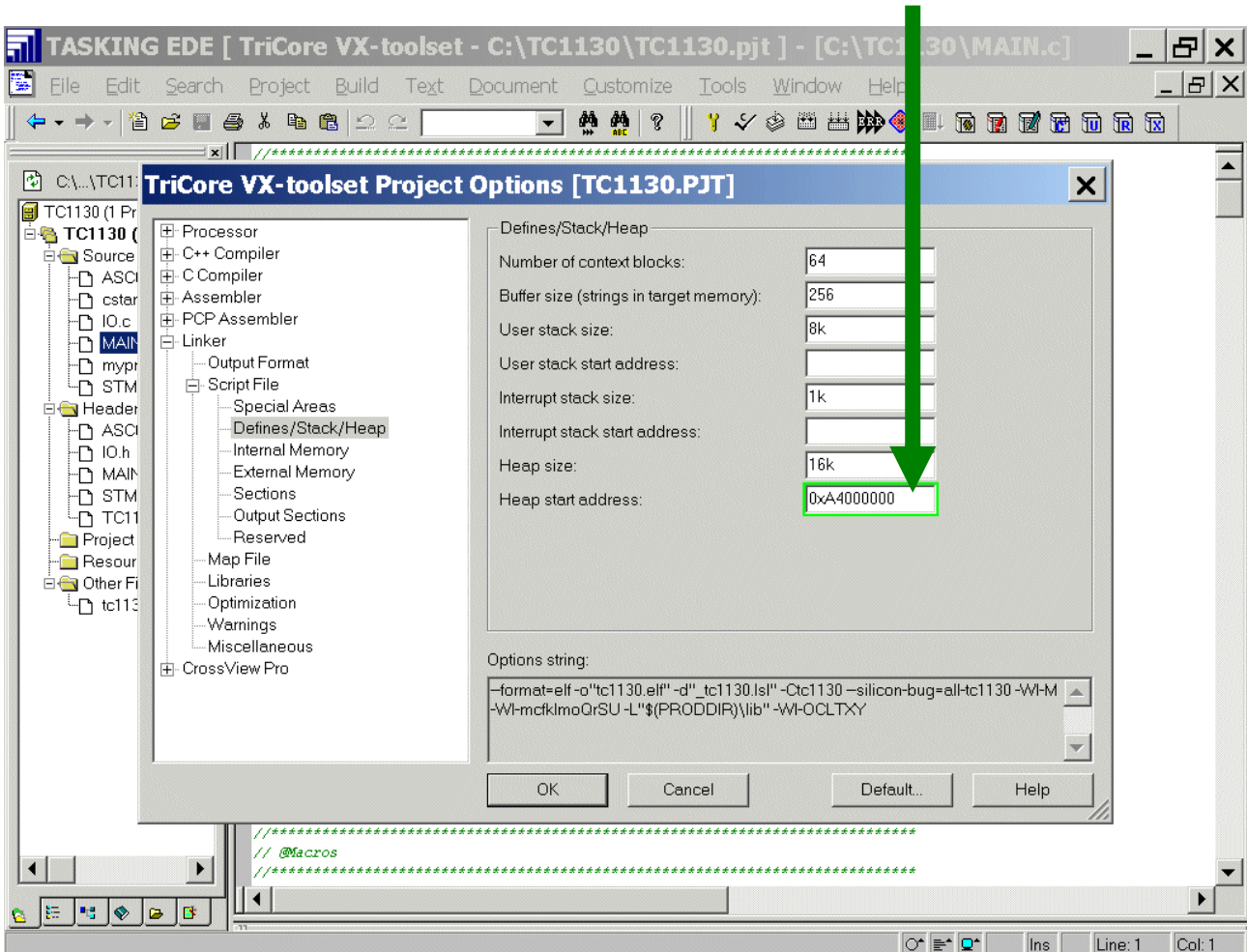
File name: select TC1130.psp

Open

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

Project – Project Options

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



OK

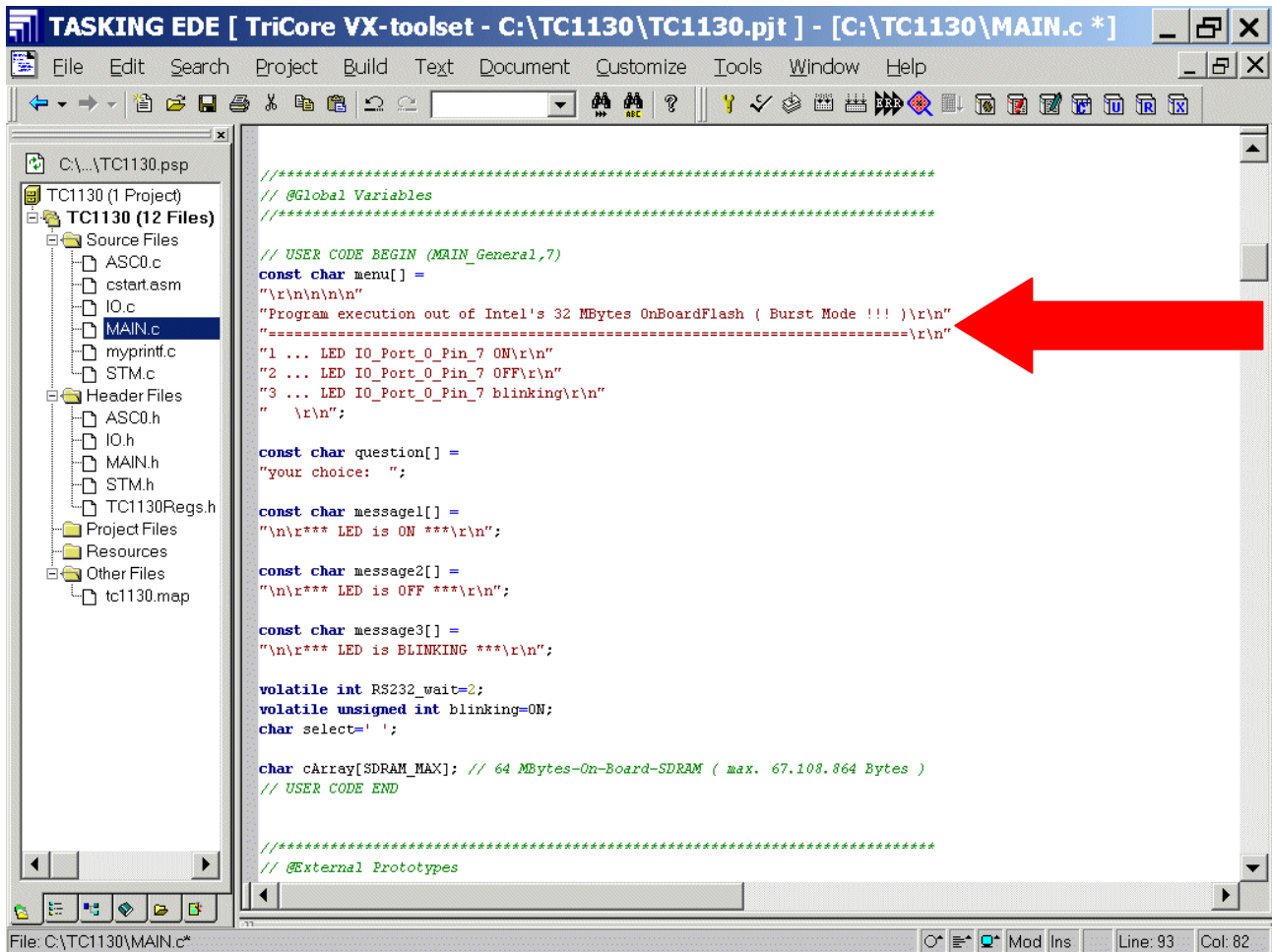
Insert your application specific program:

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

```
const char menu[] =
"\r\n\n\n\r\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

```
const char menu[] =
"\r\n\n\n\r\n"
"Program execution out of Intel's 32 MBytes OnBoardFlash ( Burst
Mode !!! )\r\n"
"=====\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";
```



The screenshot shows the TASKING EDE IDE interface. The title bar reads "TASKING EDE [TriCore VX-toolset - C:\TC1130\TC1130.pjt] - [C:\TC1130\MAIN.c *]". The menu bar includes File, Edit, Search, Project, Build, Text, Document, Customize, Tools, Window, and Help. The toolbar contains various icons for file operations and development tools. The left sidebar shows a project tree for "TC1130 (1 Project)" with 12 files, including Source Files (ASC0.c, cstart.asm, IO.c, MAIN.c, myprintf.c, STM.c), Header Files (ASC0.h, IO.h, MAIN.h, STM.h, TC1130Regs.h), Project Files, Resources, and Other Files (tc1130.map). The main editor window displays the source code for MAIN.c. A red arrow points to the line: "Program execution out of Intel's 32 MBytes OnBoardFlash (Burst Mode !!!)\r\n".

```

//*****
// @Global Variables
//*****

// USER CODE BEGIN (MAIN_General,7)
const char menu[] =
"\r\n\r\n\r\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";

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

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

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

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

volatile int RS232_wait=2;
volatile unsigned int blinking=0N;
char select=' ';

char cArray[SDRAM_MAX]; // 64 MBytes-On-Board-SDRAM ( max. 67.108.864 Bytes )
// USER CODE END

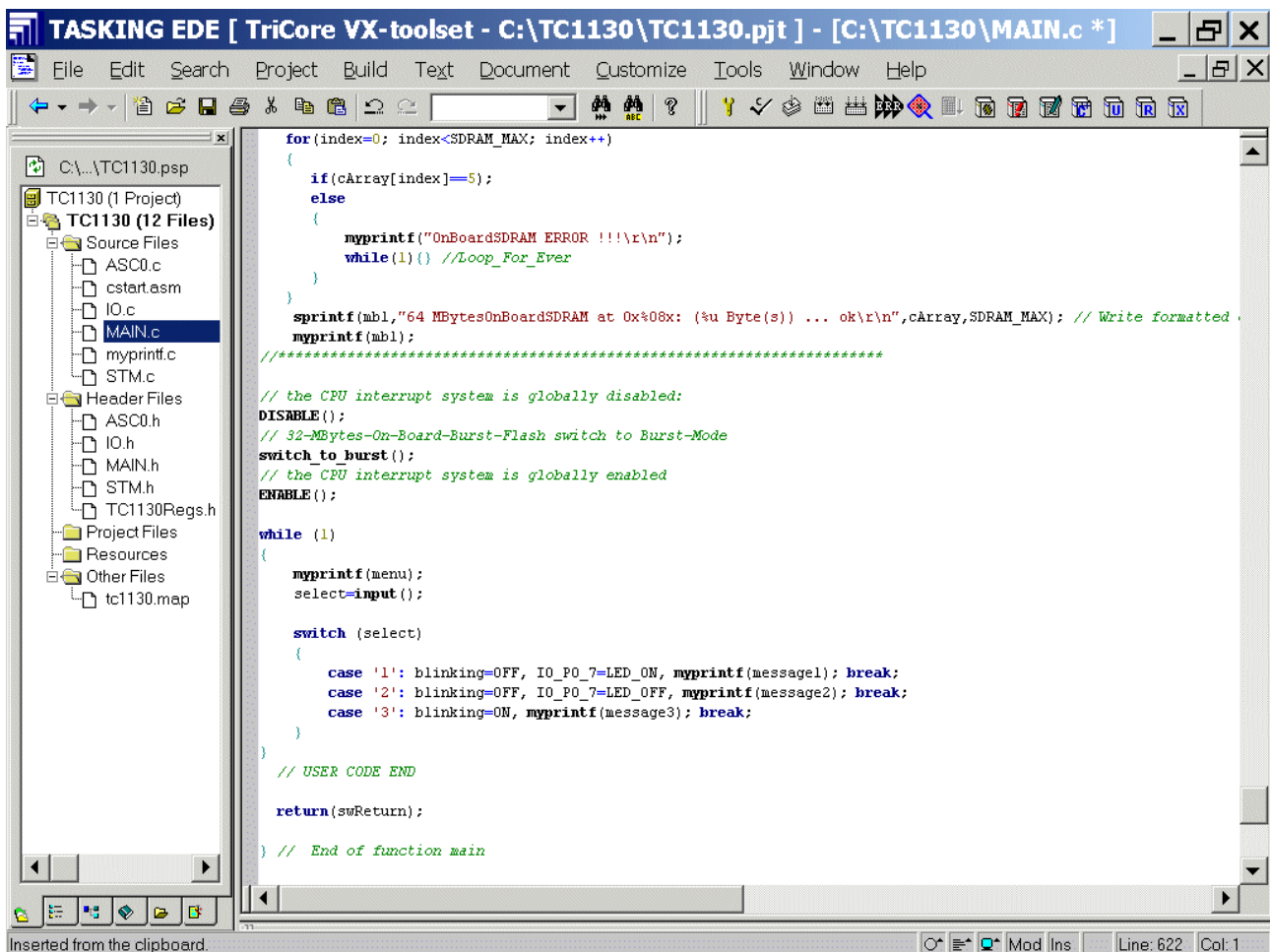
//*****
// @External Prototypes

```

File: C:\TC1130\MAIN.c* Mod Ins Line: 93 Col: 82

Double click: **Main.c**: insert application-specific-program:

```
// 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();
```



```
TASKING EDE [ TriCore VX-toolset - C:\TC1130\TC1130.pjt ] - [C:\TC1130\MAIN.c *]
File Edit Search Project Build Text Document Customize Tools Window Help
C:\...TC1130.psp
TC1130 (1 Project)
  TC1130 (12 Files)
    Source Files
      ASC0.c
      cstart.asm
      IO.c
      MAIN.c
      myprintf.c
      STM.c
    Header Files
      ASC0.h
      IO.h
      MAIN.h
      STM.h
      TC1130Regs.h
    Project Files
    Resources
    Other Files
      tc1130.map

for(index=0; index<SDRAM_MAX; index++)
{
    if(cArray[index]==5);
    else
    {
        myprintf("OnBoardSDRAM ERROR !!!\r\n");
        while(1) {} //Loop_For_Ever
    }
}
sprintf(mbl,"64 MBytesOnBoardSDRAM at 0x%08x: (%u Byte(s)) ... ok\r\n",cArray,SDRAM_MAX); // Write formatted
myprintf(mbl);
//*****
// 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();

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;
    }
}
// USER CODE END

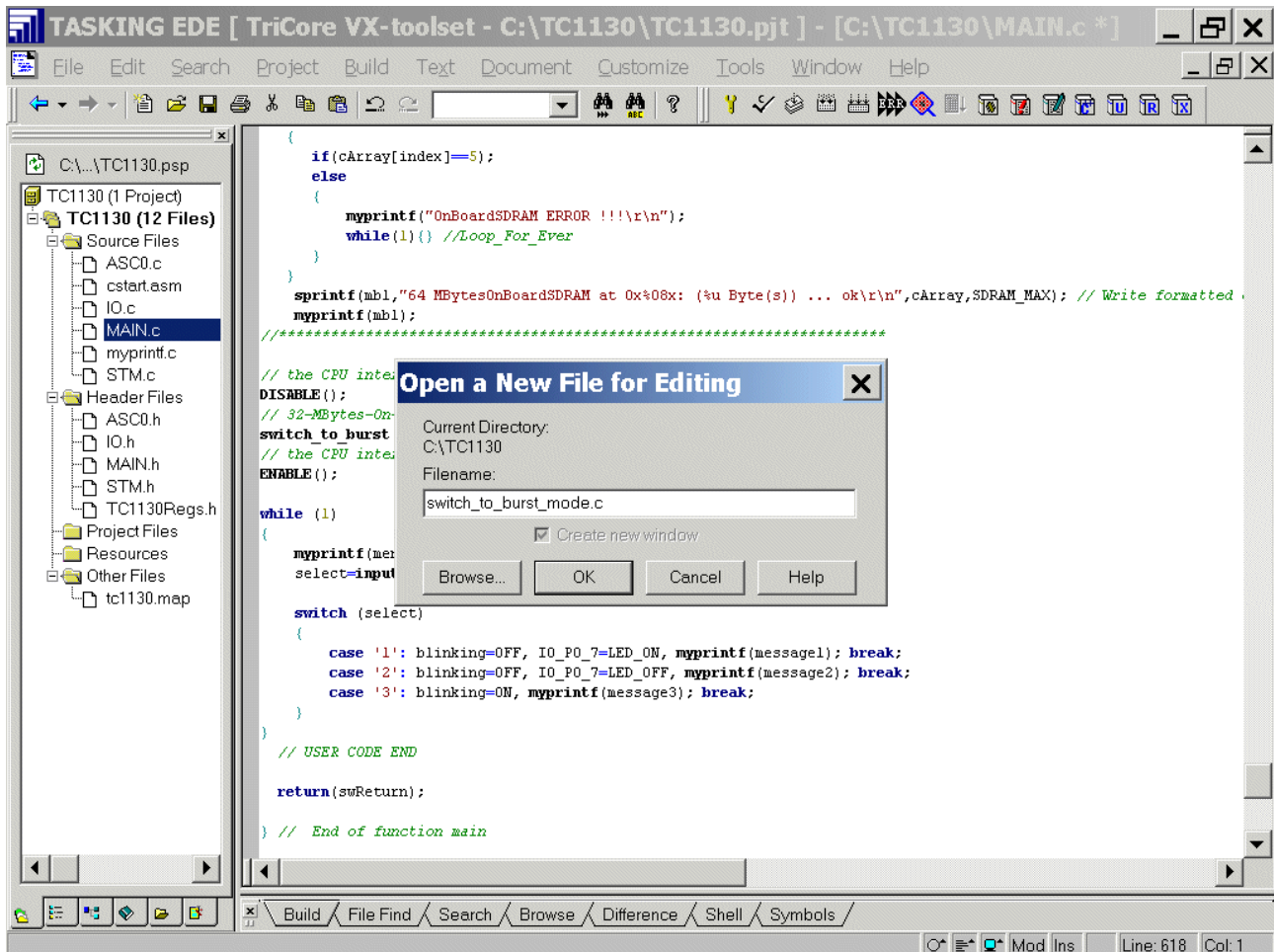
return(swReturn);

} // End of function main

Inserted from the clipboard. Mad Ins Line: 622 Col: 1
```

File – New

Insert switch_to_burst_mode.c



OK

Insert code:

```

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

#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 absolutly 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");

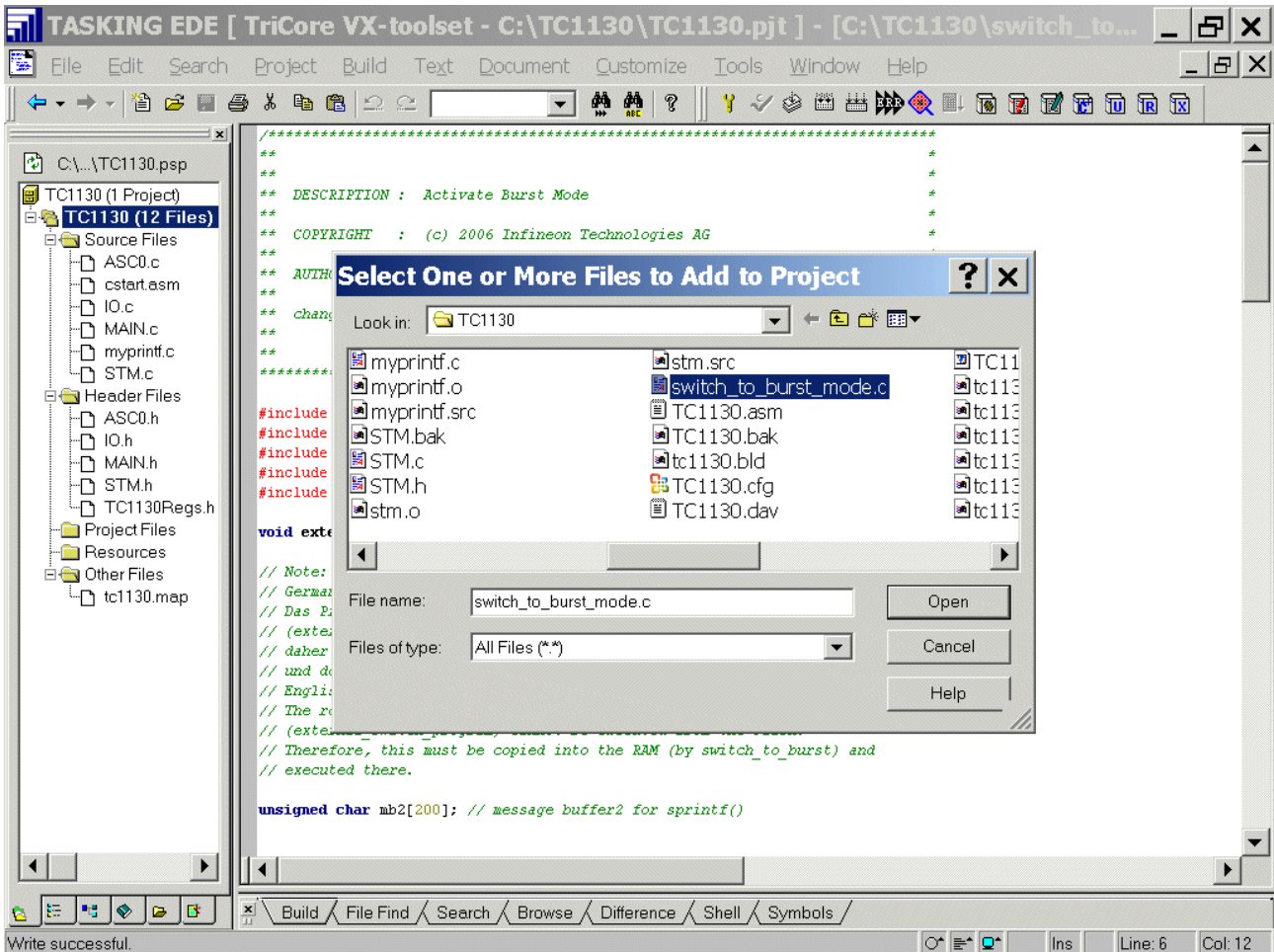
        ((void(*) (void))puiDestination)(); // jump to external ram (heap)

        free(puiDataHeap);
        myprintf("BACK from heap, Burst-Mode activated\r\n");
    }
}

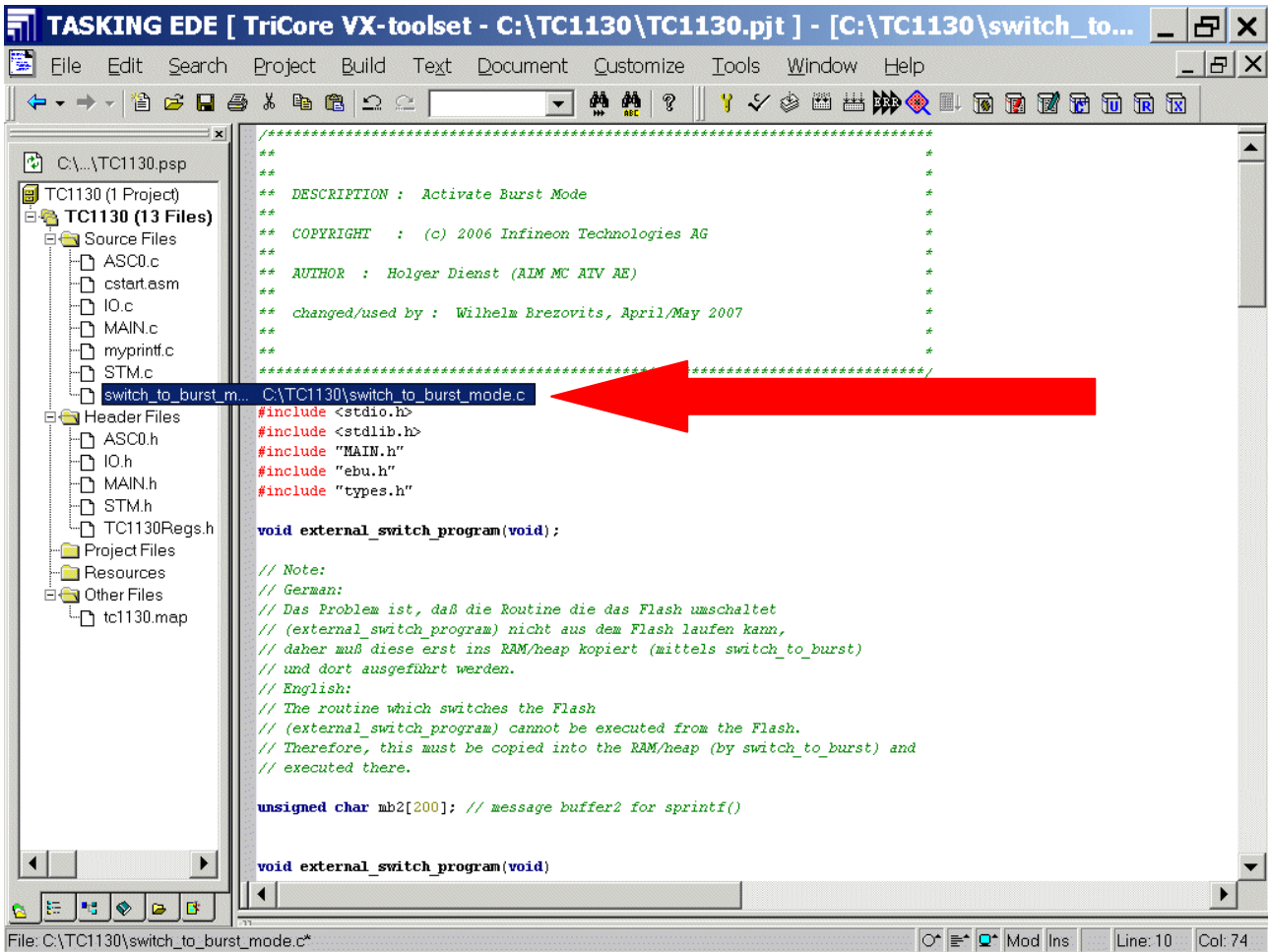
```

File – Save

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



Select switch_to_burst_mode.c
Open
OK



TASKING EDE [TriCore VX-toolset - C:\TC1130\TC1130.pjt] - [C:\TC1130\switch_to...

File Edit Search Project Build Text Document Customize Tools Window Help

C:\...TC1130.psp

- TC1130 (1 Project)
 - TC1130 (13 Files)
 - Source Files
 - ASC0.c
 - cstart.asm
 - IO.c
 - MAIN.c
 - myprintf.c
 - STM.c
 - switch_to_burst_m... C:\TC1130\switch_to_burst_mode.c
 - Header Files
 - ASC0.h
 - IO.h
 - MAIN.h
 - STM.h
 - TC1130Regs.h
 - Project Files
 - Resources
 - Other Files
 - tc1130.map

```

*****
**
** DESCRIPTION : Activate Burst Mode
**
** COPYRIGHT : (c) 2006 Infineon Technologies AG
**
** AUTHOR : Holger Dienst (AIM MC RIV AE)
**
** changed/used by : Wilhelm Brezovits, April/May 2007
**
**
**
*****
#include <stdio.h>
#include <stdlib.h>
#include "MAIN.h"
#include "ebu.h"
#include "types.h"

void external_switch_program(void);

// Note:
// German:
// Das Problem ist, daß die Routine die das Flash umschaltet
// (external_switch_program) nicht aus dem Flash laufen kann,
// daher muß diese erst ins RAM/heap kopiert (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.

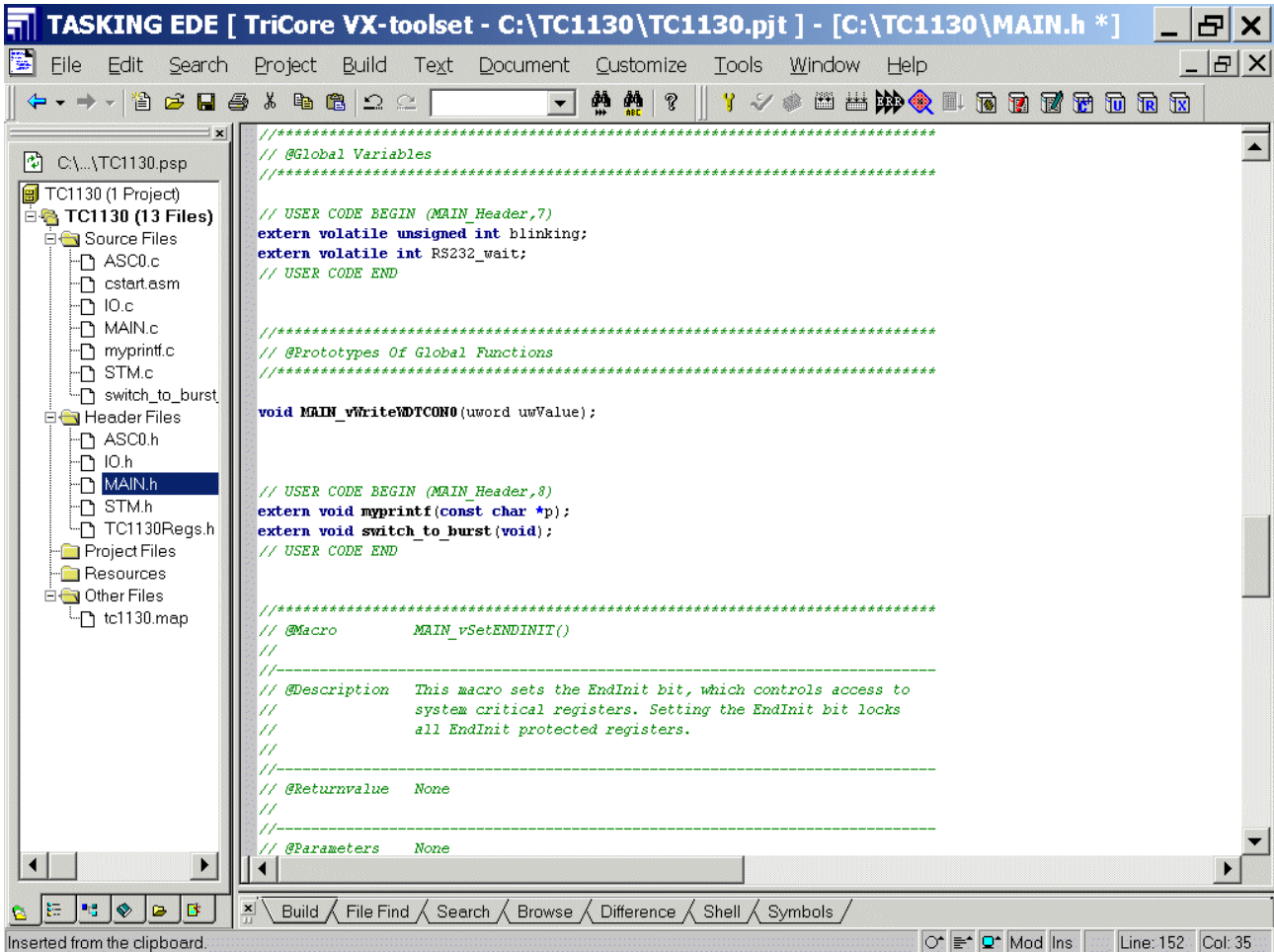
unsigned char mb2[200]; // message buffer2 for sprintf()

void external_switch_program(void)
  
```

File: C:\TC1130\switch_to_burst_mode.c* Mod Ins Line: 10 Col: 74

Double click: **Main.h** and insert Prototype of Global Function:

```
extern void switch_to_burst(void);
```



File – New

Insert types.h

OK

Insert code:

```
/******  
*  
* 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:

```

/*****
*
* 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 */
}

```

```

volatile unsigned int BUSAP0; /* Bus access parameter Register 0 (32) */
volatile unsigned int RESERVED20[1]; /* Reserved */
volatile unsigned int BUSAP1; /* Bus access parameter Register 1 (32) */
volatile unsigned int RESERVED21[1]; /* Reserved */
volatile unsigned int BUSAP2; /* Bus access parameter Register 2 (32) */
volatile unsigned int RESERVED22[1]; /* Reserved */
volatile unsigned int BUSAP3; /* Bus access parameter Register 3 (32) */
volatile unsigned int RESERVED23[17]; /* Reserved */
volatile unsigned int EMUAS; /* Emulator Memory Address Select Register (32) */
volatile unsigned int RESERVED24[1]; /* Reserved */
volatile unsigned int EMUBC; /* Emulator Memory Bus Configuration Register (32) */
volatile unsigned int RESERVED25[1]; /* Reserved */
volatile unsigned int EMUBAP; /* Emulator Memory access parameter (32) */
volatile unsigned int RESERVED26[1]; /* Reserved */
volatile unsigned int EMUOVL; /* Overlay memory chip-select generation (32) */
volatile unsigned int RESERVED27[5]; /* Reserved */
volatile unsigned int USERCON; /* Test/Control Configuration Register (32) */

} EBU;

/* Global Control Register */
#define EBUEXTLOCK 0x00000010 /* EBU External Bus Lock Control */
#define EBUARBSYNC 0x00000020 /* EBU Arbitration Inputs Evaluation Control */
#define EBUARBMODE_MSK 0x000000C0 /* EBU Arbitration Strategy */
#define EBUARBMODE_EBUDIS 0x00000000 /* EBU is Disabled */
#define EBUARBMODE_EXTM 0x00000040 /* EBU is External Master */
#define EBUARBMODE_EXTS 0x00000080 /* EBU is External Slave */
#define EBUARBMODE_NOARB 0x000000C0 /* EBU Arbitration is Disabled */
#define EBUTIMOUTC_MSK 0x0000FF00 /* EBU Time Out Control */
#define EBUGLOBAL_CS_MSK 0x000FF0000 /* EBU chip select global mask */
#define EBUGLOBAL_CS0 0x000100000 /* EBU chip select global 0 */
#define EBUGLOBAL_CS1 0x000200000 /* EBU chip select global 1 */
#define EBUGLOBAL_CS2 0x000400000 /* EBU chip select global 2 */
#define EBUGLOBAL_CS3 0x000800000 /* EBU chip select global 3 */
#define EBUBUSCLK_MSK 0x03000000 /* EBU bus clock */
#define EBUBUSCLK_1 0x000000000 /* EBU bus clock = LMB clock */
#define EBUBUSCLK_2 0x010000000 /* EBU bus clock = LMB clock/2 */
#define EBUBUSCLK_4 0x020000000 /* EBU bus clock = LMB clock/4 */
#define EBUSDCMSEL 0x040000000 /* EBU SDRAM clock gated */
#define EBUCSOFAM 0x080000000 /* EBU CS0 Fills Address Map */
#define EBUEMUFAM 0x100000000 /* EBU CSemu Fills Address Map */
#define EBUBFSSS 0x200000000 /* EBU Burst Flash Single Stage Synchronisation */

/* Burst Flash Control Register */
#define EBUFETBLEN0_MSK 0x0000000F /* Fetch burst length Type 0 */
#define EBUFETBLEN0_1 0x00000000 /* Fetch burst length 1 data access Type 0 */
#define EBUFETBLEN0_2 0x00000001 /* Fetch burst length 2 data access Type 0 */
#define EBUFETBLEN0_4 0x00000002 /* Fetch burst length 4 data access Type 0 */
#define EBUFETBLEN0_8 0x00000003 /* Fetch burst length 8 data access Type 0 */
#define EBUFBBMSEL0_ 0x00000010 /* Flash burst buffer mode select Type 0 */
#define EBUWAITFUNC0 0x00000020 /* Operation of /WAIT input Type 0 */
#define EBUEXTCLK_MSK 0x000000C0 /* BFCLK external clock */
#define EBUEXTCLK_1 0x00000000 /* BFCLK = LMBCLK */
#define EBUEXTCLK_2 0x00000040 /* BFCLK = LMBCLK/2 */
#define EBUEXTCLK_3 0x00000080 /* BFCLK = LMBCLK/3 */
#define EBUEXTCLK_4 0x000000C0 /* BFCLK = LMBCLK/4 */
#define EBUBFCMSEL 0x00000100 /* BFCLK only present during burst access */
#define EBUEBSE0 0x00000200 /* ADV and BAA not 1/2LMBCLK delayed Type 0 */
#define EBUDBA0 0x00000400 /* Disable Burst Address Wrapping Type 0 */
#define EBUFDBKEN 0x00000800 /* Burst Clock Feedback Enable */
#define EBUdTALNCY 0x0000F000 /* Latency Cycle Control */
#define EBUFETBLEN1_MSK 0x0000F000 /* Fetch burst length Type 1 */
#define EBUFETBLEN1_1 0x00000000 /* Fetch burst length 1 data access Type 1 */
#define EBUFETBLEN1_2 0x00010000 /* Fetch burst length 2 data access Type 1 */
#define EBUFETBLEN1_4 0x00020000 /* Fetch burst length 4 data access Type 1 */
#define EBUFETBLEN1_8 0x00030000 /* Fetch burst length 8 data access Type 1 */
#define EBUFBBMSEL1_ 0x00100000 /* Flash burst buffer mode select Type 1 */
#define EBUWAITFUNC1 0x00200000 /* Operation of /WAIT input Type 1 */
#define EBUEBSE1 0x00400000 /* ADV and BAA not 1/2LMBCLK delayed Type 1 */
#define EBUDBA1 0x00800000 /* Disable Burst Address Wrapping Type 1 */

/* SDRAM Refresh Register */
#define EBUREFRESHC_MSK 0x0000003F /* Refresh Counter Period Value */
#define EBUREFRESHR_MSK 0x000001C0 /* Number of Refresh Commands */
#define EBUSELFRXST 0x00000200 /* Self refresh exit status */
#define EBUSELFRFX 0x00000400 /* Self refresh exit */
#define EBUSELFRXNST 0x00000800 /* Self refresh entry status */
#define EBUSELFRFXN 0x00001000 /* Self refresh entry */
#define EBUAUTOSELFR 0x00002000 /* Automatic Self Refresh (only SDRMREF0) */

/* SDRAM Configuration */
#define EBUCRAS_MSK 0x0000000F /* Row Activate to Precharge Cycle Counter */
#define EBUCRFSH_MSK 0x000000F0 /* Refresh Commands Counter Value */
#define EBUCRSC_MSK 0x00000300 /* Mode Register Setup Time Counter Value */
#define EBUCRP_MSK 0x00000C00 /* Row Precharge Time Counter Value */

```



```

#define EBUAWIDTH_MSK      0x00003000      /* Starting Index of Address Lines in SDRAM Row Address
*/
#define EBUCRCD_MSK        0x0000C000      /* Row-to-Column Delay Counter Value */
#define EBUCRC_MSK         0x00070000      /* Row Cycle Time Counter Value */
#define EBUPAGEM_MSK       0x00380000      /* Mask for page tag */
#define EBUBANKM_MSK       0x01C00000      /* Mask for bank tag */
#define EBU DTALTNCY_MSK   0xF0000000      /* Mask Latency Cylce Control */

/* SDRAM Mode */
#define EBUBURSTL_MSK      0x00000007      /* SDRAM burst length mask */
#define EBUBURSTL_1        0x00000000      /* SDRAM burst length 1 */
#define EBUBURSTL_8        0x00000003      /* SDRAM burst length 8 */
#define EBUCASLAT_MSK      0x00000070      /* CAS-Latency Mask */
#define EBUCASLAT_2        0x00000020      /* CAS-Latency 2 */
#define EBUCASLAT_3        0x00000030      /* CAS-Latency 3 */

/* SDRAM Status Register */
#define EBUREFERR          0x00000001      /* SDRAM Refresh Error Flag */
#define EBUSDRMBUSY        0x00000002      /* SDRAM Busy Flag */

/* Address Select Register */
#define EBUREGENAB         0x00000001      /* Memory Region Enable Control */
#define EBUALTENAB         0x00000002      /* Altseg is always compared with LMB address */
#define EBUMASK_MSK        0x000000F0      /* Memory Region Address Mask */
#define EBUALTSEG_MSK      0x000000F0      /* Memory Region Alternate Segment */
#define EBUBASE_MSK        0xFFFFF000      /* Memory Region Base Address */

/* Bus Configuration Register */
#define EBUCMULTMAP        0x0000007F      /* Multiplier Map */
#define EBUCMULTMAP_ADDR   0x00000001      /* Multiplier Map addr */
#define EBUCMULTMAP_AHOLDC 0x00000002      /* Multiplier Map aholdc */
#define EBUCMULTMAP_CMDDELAY 0x00000004      /* Multiplier Map cmddelay */
#define EBUCMULTMAP_BURSTC 0x00000008      /* Multiplier Map burstc */
#define EBUCMULTMAP_DATAC  0x00000010      /* Multiplier Map datac */
#define EBUCMULTMAP_RDRECOVC 0x00000020      /* Multiplier Map rdrecovc */
#define EBUCMULTMAP_WRRECOVC 0x00000040      /* Multiplier Map wrrecovc */
#define EBUWEAK_PREFETCH  0x00000100      /* Code prefetch can be aborted by data access */
#define EBUAALIGN          0x00000200      /* Address Alignment */
#define EBUCTYPE_MSK       0x00000C00      /* Cycle Typ */
#define EBUCTYPE_NOMUX     0x00000000      /* Cycle Typ is non-multiplexed */
#define EBUCMULT_MSK       0x0000E000      /* Cycle Multiplier Control */
#define EBUCMULT_CM1       0x00000000      /* Cycle Multiplier 1 */
#define EBUCMULT_CM4       0x00002000      /* Cycle Multiplier 4 */
#define EBUCMULT_CM8       0x00004000      /* Cycle Multiplier 8 */
#define EBUCMULT_CM16      0x00006000      /* Cycle Multiplier 16 */
#define EBUCMULT_CM32      0x00008000      /* Cycle Multiplier 32 */
#define EBUDLOAD           0x00020000      /* Data access always fed from external bus */
#define EBUPREFETCH        0x00040000      /* Code access always uses prefetch buffer */
#define EBUWAITINV         0x00080000      /* #WAIT Input is active High */
#define EBUBCGEN_MSK       0x00300000      /* Byte Control Signal Timing Mode */
#define EBUBCGEN_CSM       0x00000000      /* Chipselect Mode */
#define EBUBCGEN_CM        0x00100000      /* Control Mode */
#define EBUBCGEN_WEM       0x00200000      /* Write Enable Mode */
#define EBUBCGEN_DQMM      0x00300000      /* DQM Mode */
#define EBUPORTW_MSK       0x00C00000      /* External Device Data Width Control */
#define EBUPORTW_16        0x00400000      /* 16 Bit */
#define EBUPORTW_32        0x00800000      /* 32 Bit */
#define EBUWAIT_MSK        0x03000000      /* External Waitstate Control */
#define EBUWAIT_DIS        0x00000000      /* External Waitstate Control OFF */
#define EBUWAIT_ASYN       0x01000000      /* Asynchronous Input at #WAIT */
#define EBUWAIT_SYN        0x02000000      /* (Early for Burst Flash Devices) Synchronous Input at
#WAIT */
#define EBUWAIT_NAND        0x03000000      /* NAND Flash Mode (not valid for Burst Flash Devices) */
#define EBUAGEN_MSK        0x70000000      /* Address Generation Control */
#define EBUAGEN_DEMUX      0x00000000      /* Demultiplex address */
#define EBUAGEN_MUX        0x10000000      /* Multiplex address */
#define EBUAGEN_BURST0     0x20000000      /* Burst Flash Type 0 access */
#define EBUAGEN_SDRAM0     0x30000000      /* SDRAM Type 0 access */
#define EBUAGEN_SDRAM1     0x40000000      /* SDRAM Type 1 access */
#define EBUAGEN_BURST1     0x50000000      /* Burst Flash Type 1 access */
#define EBUWRITE           0x80000000      /* Write Protection */

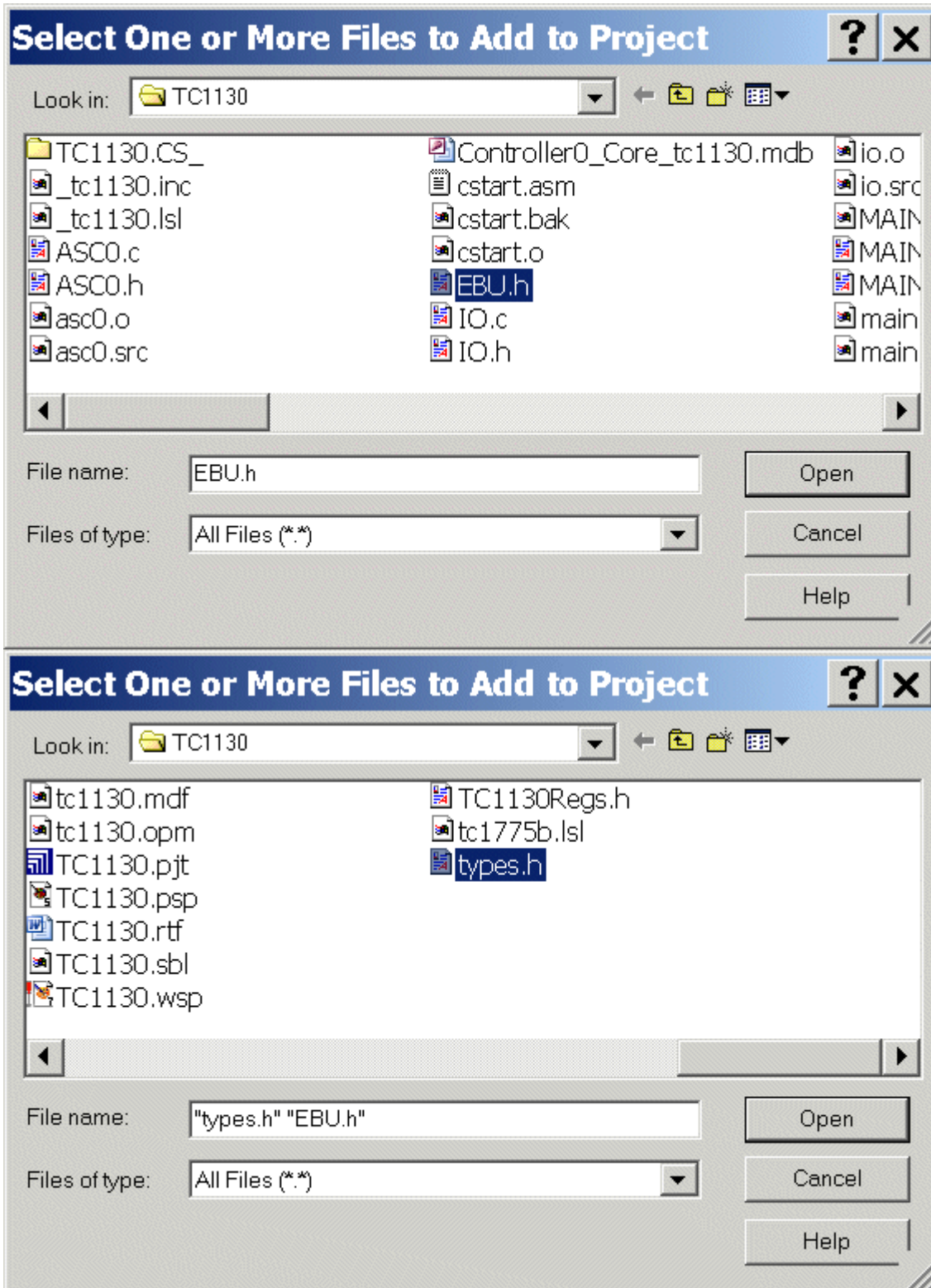
/* Bus Access Parameter Register */
#define EBU DTACS_MSK       0x0000000F      /* Recovery cycles between different regions */
#define EBU DTARDWR_MSK     0x000000F0      /* Recovery cycles between read and write accesses */
#define EBUWRRECOVC_MSK    0x00000070      /* Recovery Cycles after write accesses */
#define EBU RDRECOVC_MSK    0x00000380      /* Recovery Cycles after read accesses */
#define EBU DATAC_MSK       0x0000C000      /* Data hold Cycles for write accesses */
#define EBU BURSTC_MSK      0x00007000      /* Data Cycles during burst accesses */
#define EBU WAITWRC_MSK     0x00380000      /* Programmed Waitstates for Write Access */
#define EBU WAITRDC_MSK     0x01C00000      /* Programmed Waitstates for Read Access */
#define EBU CMDDELAY_MSK    0x0E000000      /* Programmed command delay cycles */
#define EBU HOLDC_MSK       0x30000000      /* Address Hold Cycles for multiplexes accesses */
#define EBU ADDR_MSK        0xC0000000      /* Address Cycles */

```

```
#endif /* EBU H */
```

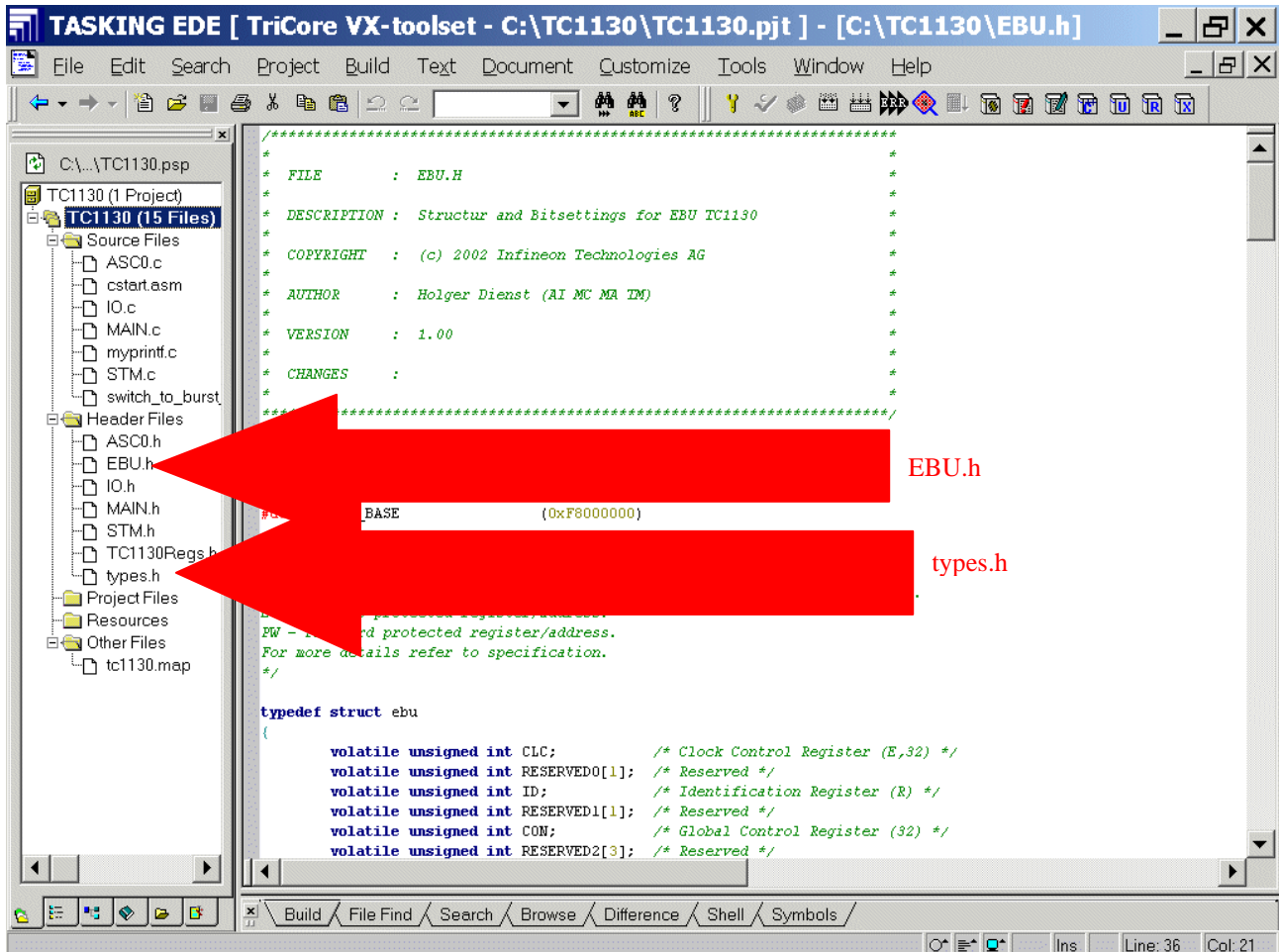
File – Save

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



Select EBU.h
Select types.h

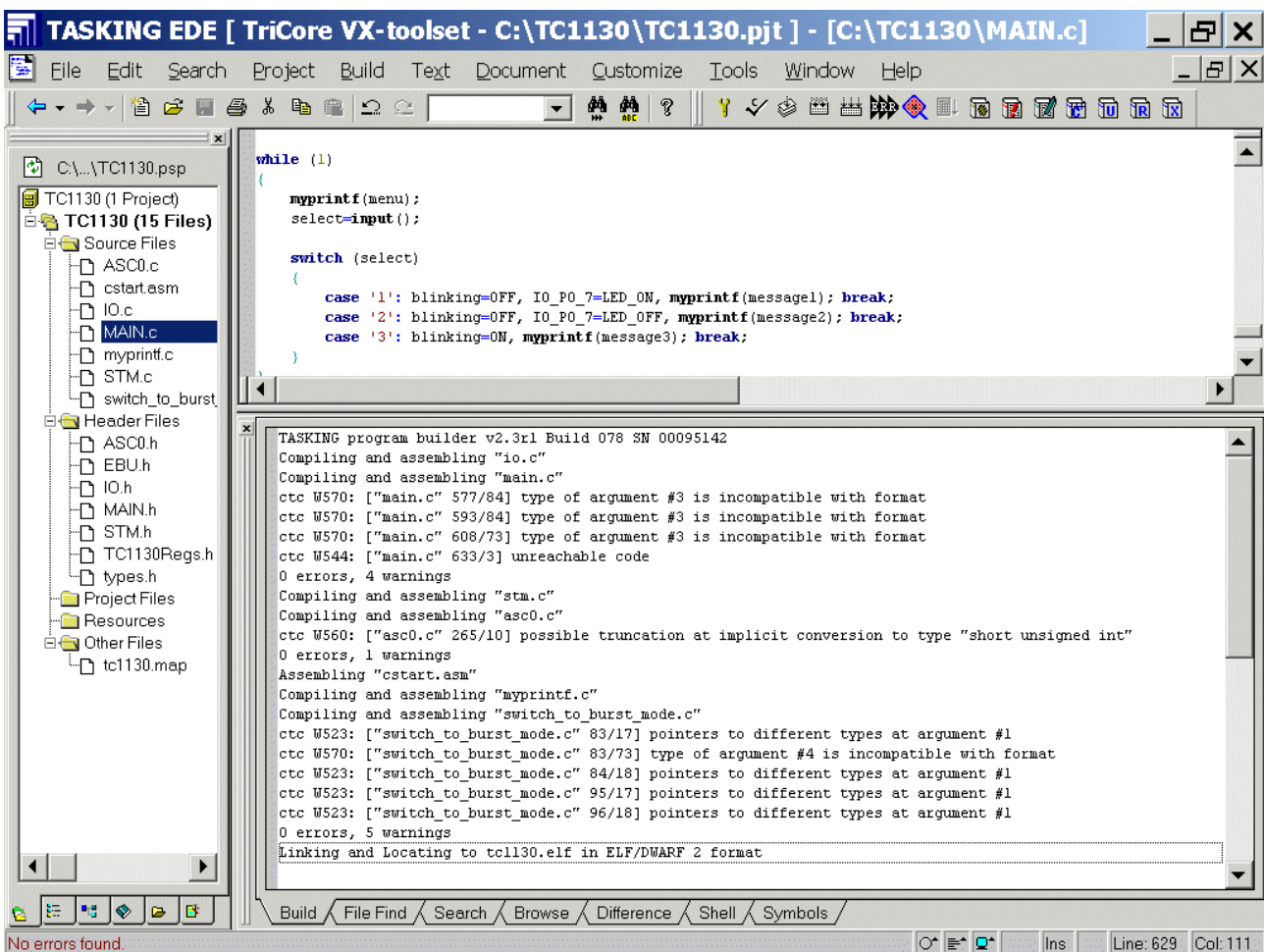
Open
OK



Generate your application program:

Build
Rebuild

OR



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

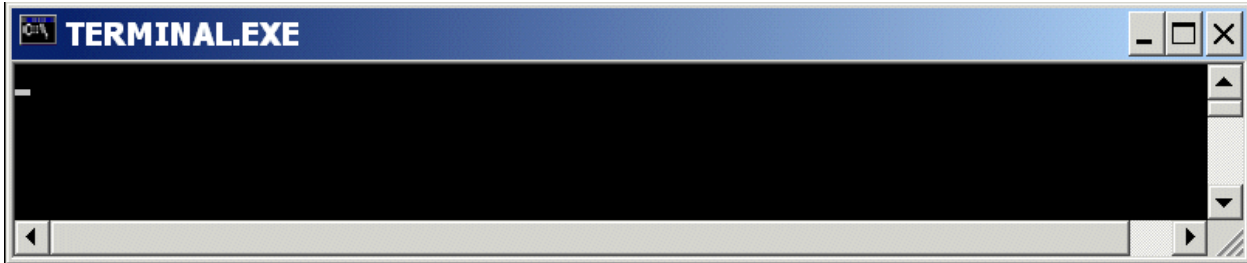
Exit

File – Close Workspace

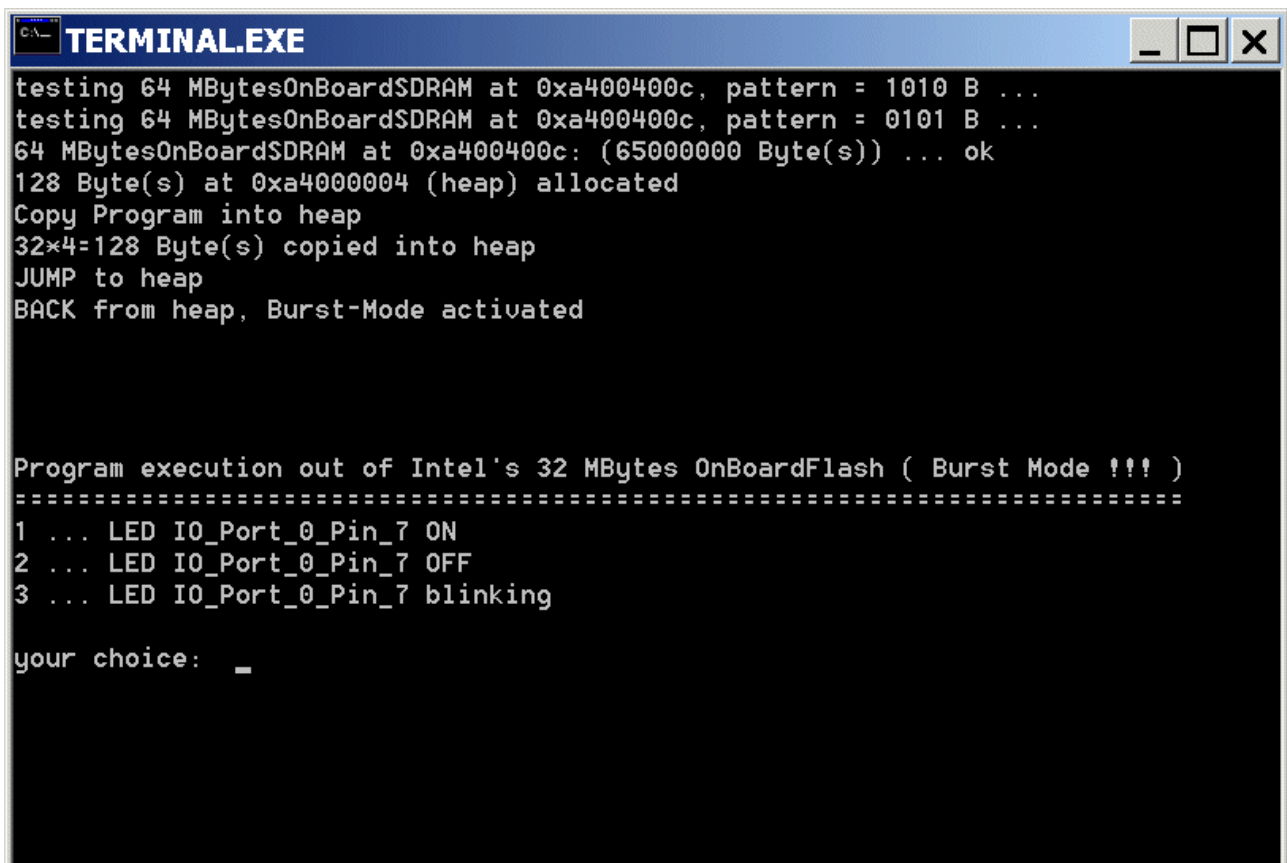
Yes

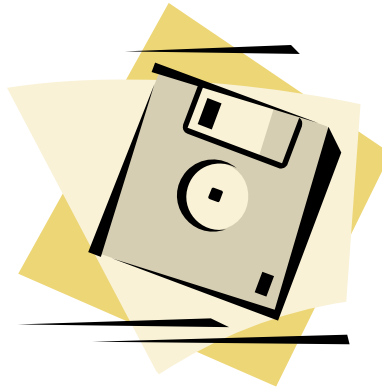
File – Exit

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

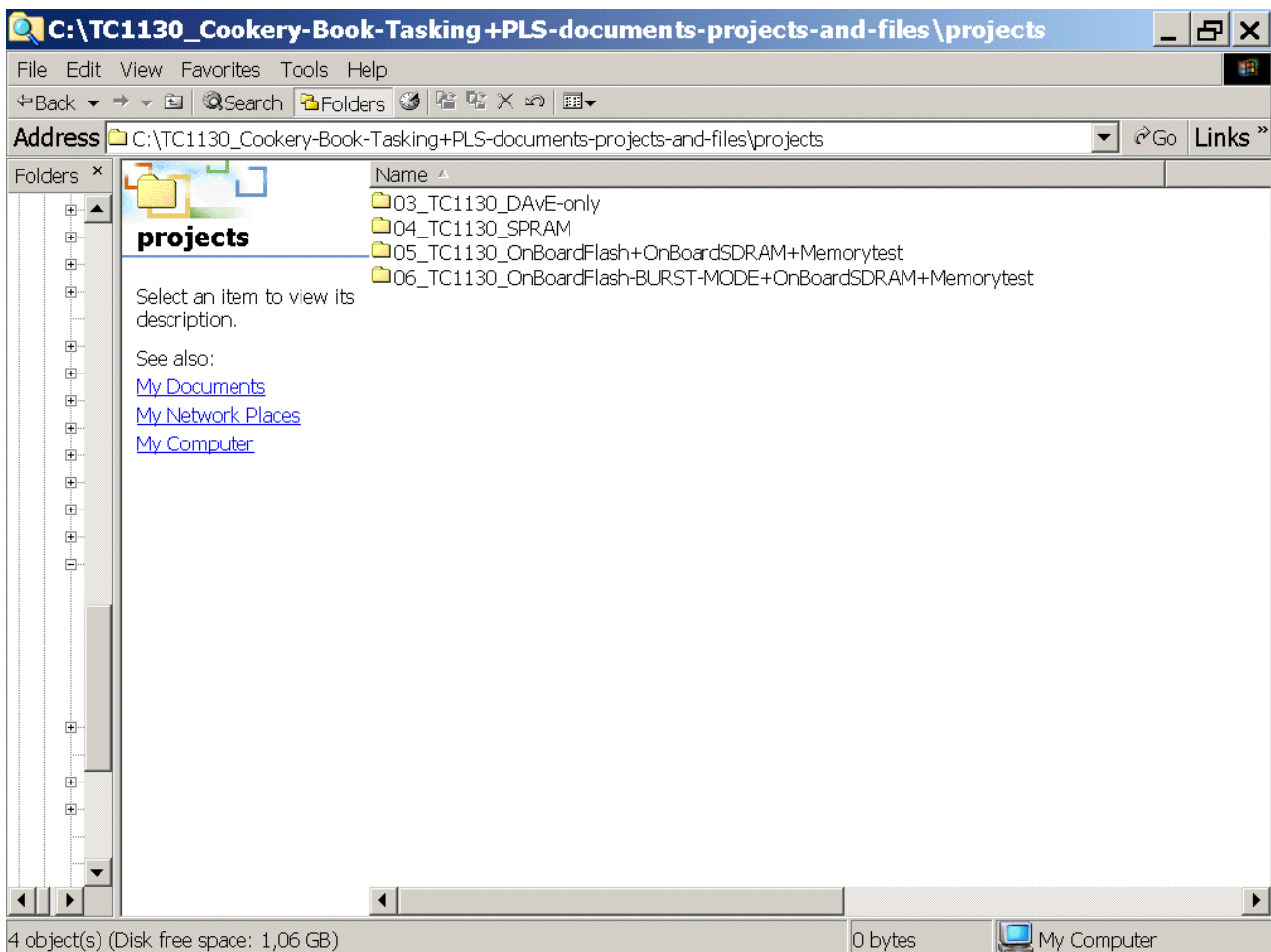


Power-On the Board and see the result:

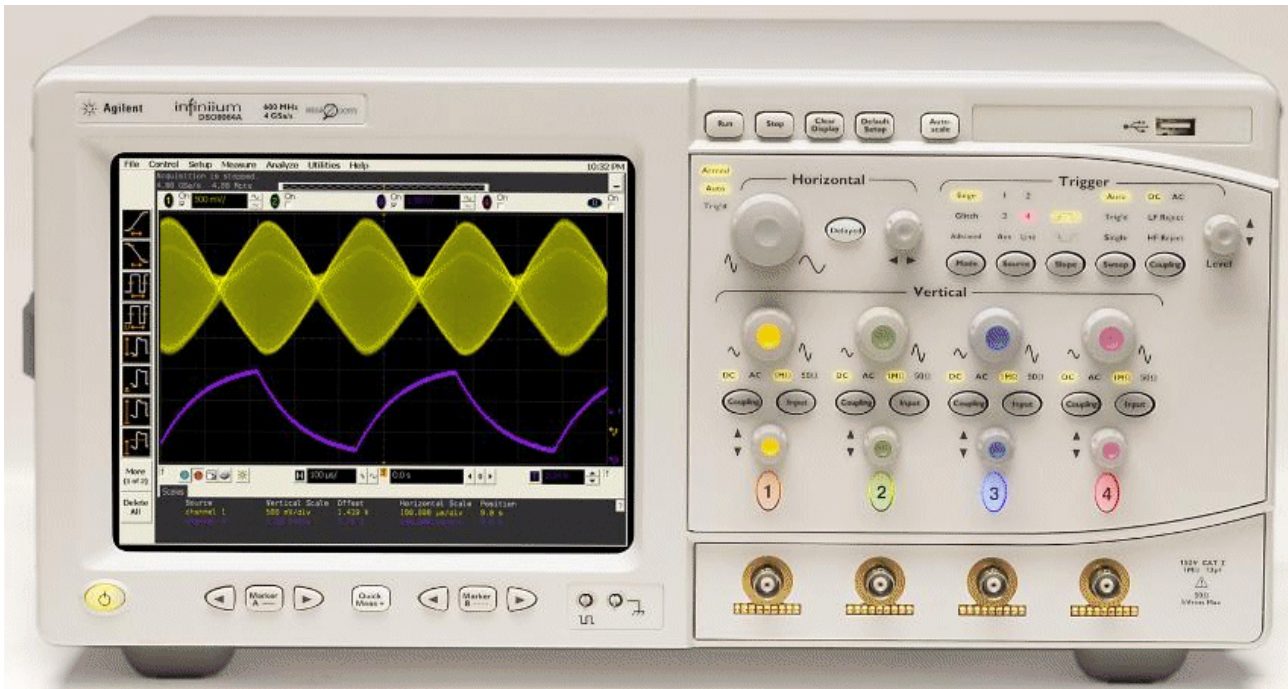




We recommend now to **copy and store** your project-directory “C:\TC1130” to “06_TC1130_OnBoardFlash-BURST-MODE+OnBoardSDRAM+Memorytest”:



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

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

to

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

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

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

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

```
// - 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(); __nop(); __nop(); __nop(); __nop(); __nop(); __nop();
    }
}
```

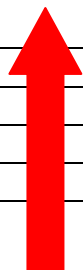

Result:



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:

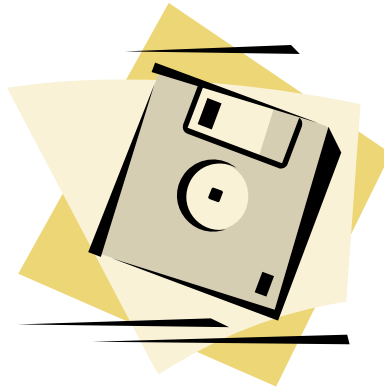
Program execution out of	Time [s] ^{*1}	Frequency [Hz] ^{*2}
PMI_SPRAM	0,104	9,60
OnBoardFlash (Burst Mode)	1,086	0,92
OnBoardFlash	2,265	0,44



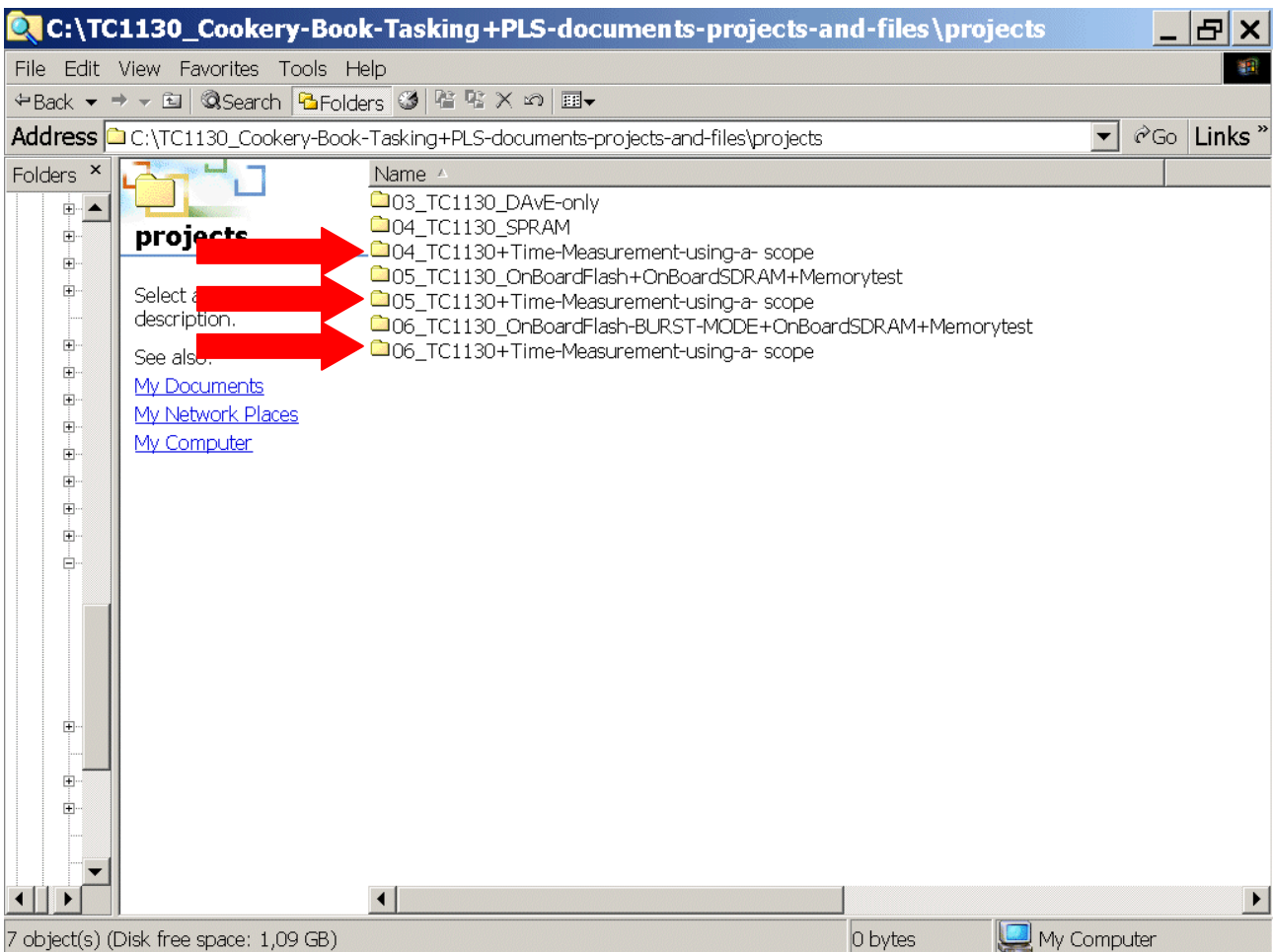
Speed

*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”:



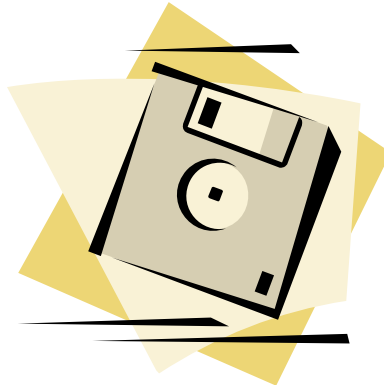
8.) Time – Measurement [Using the SystemTimer (STM)]

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

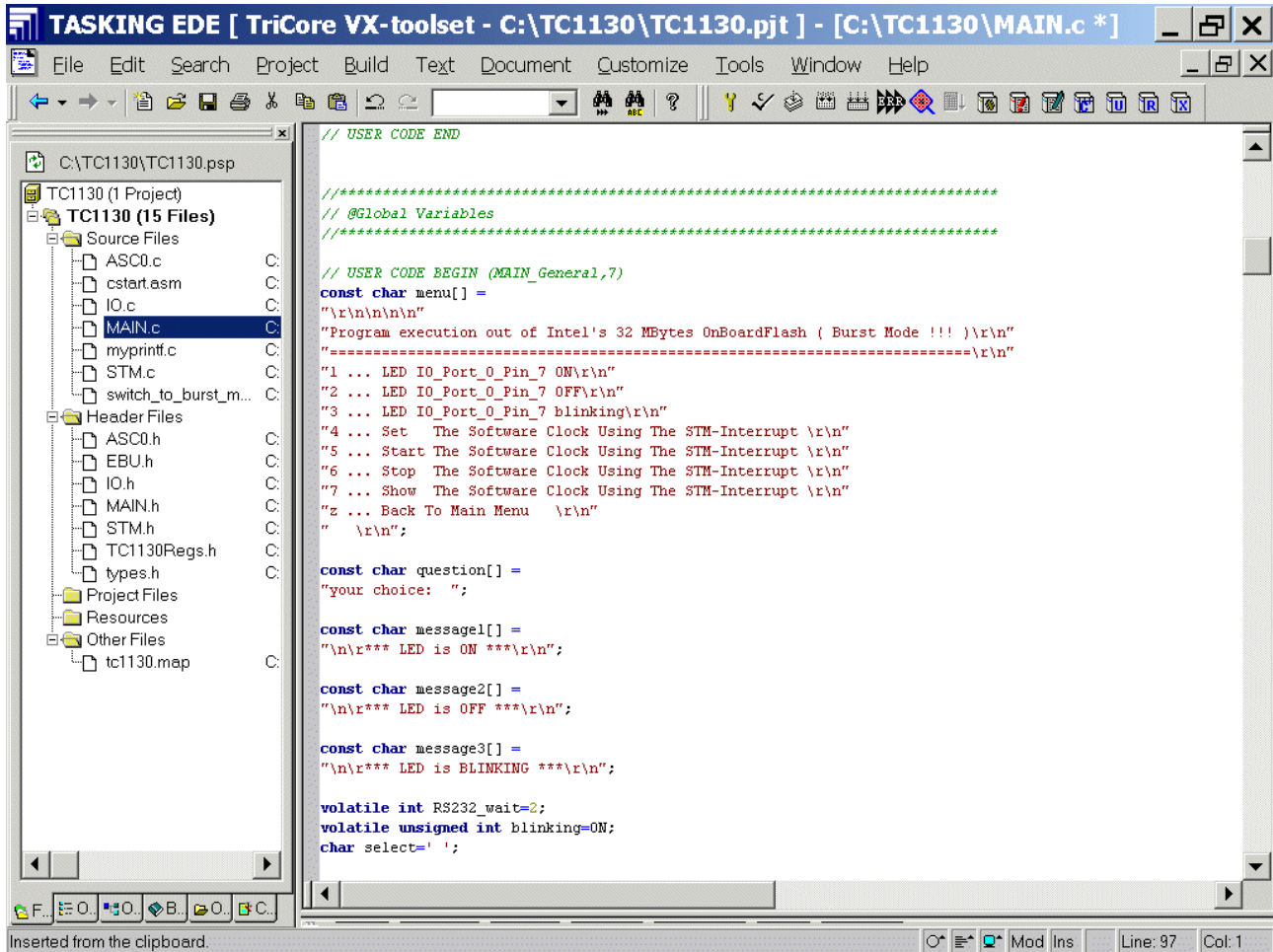
Insert your application specific program:

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

```
const char menu[] =
"\r\n\n\n\r\n"
"Program execution out of Intel's 32 MBytes OnBoardFlash ( Burst
Mode !!! )\r\n"
"=====\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\r\n"
"Program execution out of Intel's 32 MBytes OnBoardFlash ( Burst
Mode !!! )\r\n"
"=====\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"
"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";
```

```

TASKING EDE [ TriCore VX-toolset - C:\TC1130\TC1130.pjt ] - [ C:\TC1130\MAIN.c *]
File Edit Search Project Build Text Document Customize Tools Window Help
C:\TC1130\TC1130.psp
TC1130 (1 Project)
  TC1130 (15 Files)
    Source Files
      ASC0.c
      cstart.asm
      IO.c
      MAIN.c
      myprintf.c
      STM.c
      switch_to_burst_m...
    Header Files
      ASC0.h
      EBU.h
      IO.h
      MAIN.h
      STM.h
      TC1130Regs.h
      types.h
    Project Files
    Resources
    Other Files
      tc1130.map

// USER CODE END
//*****
// @Global Variables
//*****
// USER CODE BEGIN (MAIN_General,7)
const char menu[] =
"\r\n\r\n\r\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"
"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";

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

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

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

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

volatile int RS232_wait=2;
volatile unsigned int blinking=0;
char select=' ';

```

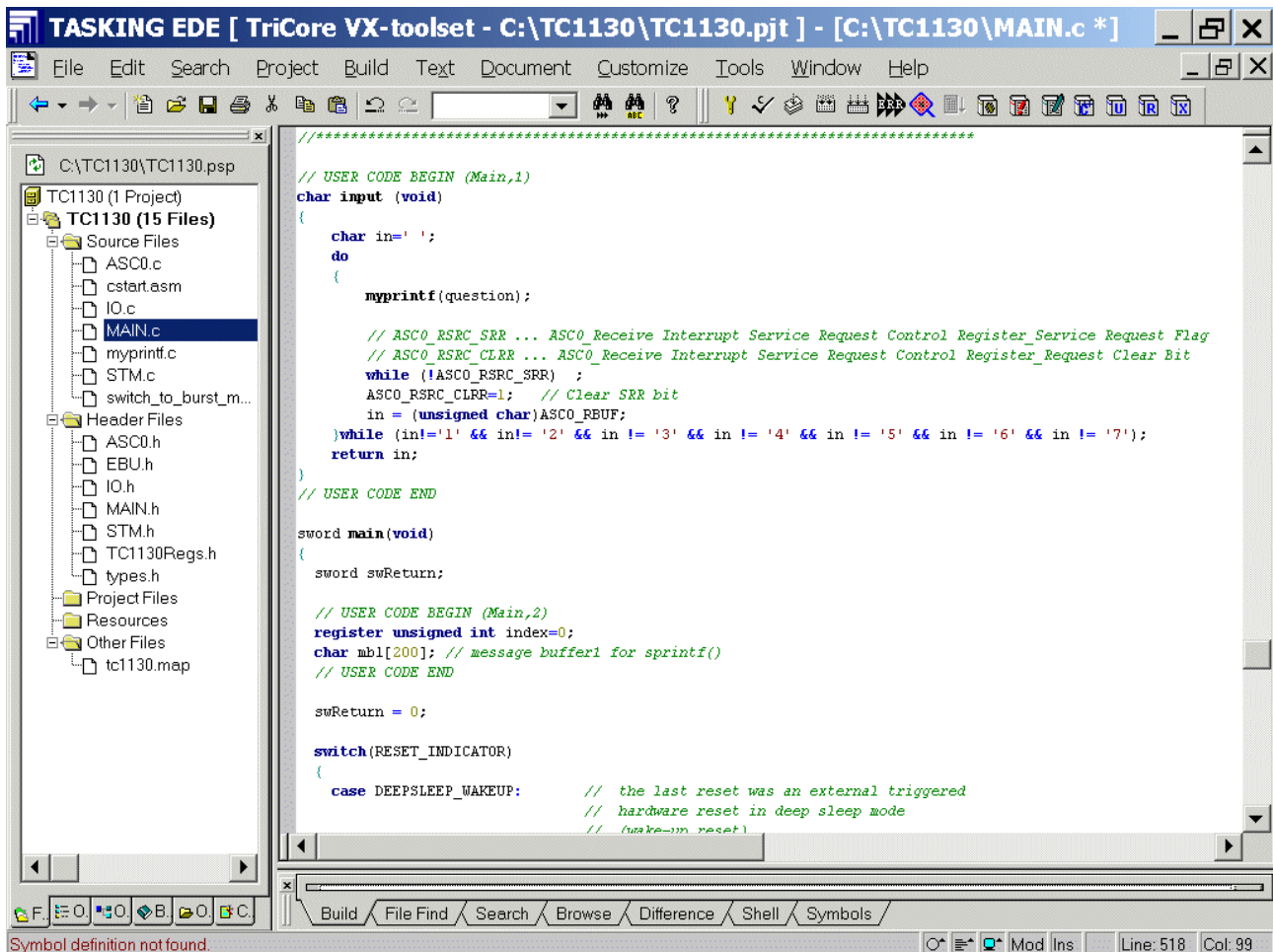
Inserted from the clipboard. Mod Ins Line: 97 Col: 1

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

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

To

```
}while (in!='1' && in!= '2' && in != '3' && in != '4' && in != '5'  
&& in != '6' && in != '7');
```



The screenshot shows the TASKING EDE IDE interface. The left pane displays the project structure for TC1130, with the file **MAIN.c** selected under Source Files. The main editor window shows the C code for **char input (void)**. The original code (shown in red in the previous blocks) is being replaced with a new code block (shown in green in the screenshot) that checks for characters '1' through '7'.

```

// USER CODE BEGIN (Main,1)
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' && in != '4' && in != '5' && in != '6' && in != '7');
    return in;
}
// USER CODE END

sword main(void)
{
    sword swReturn;

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

    swReturn = 0;

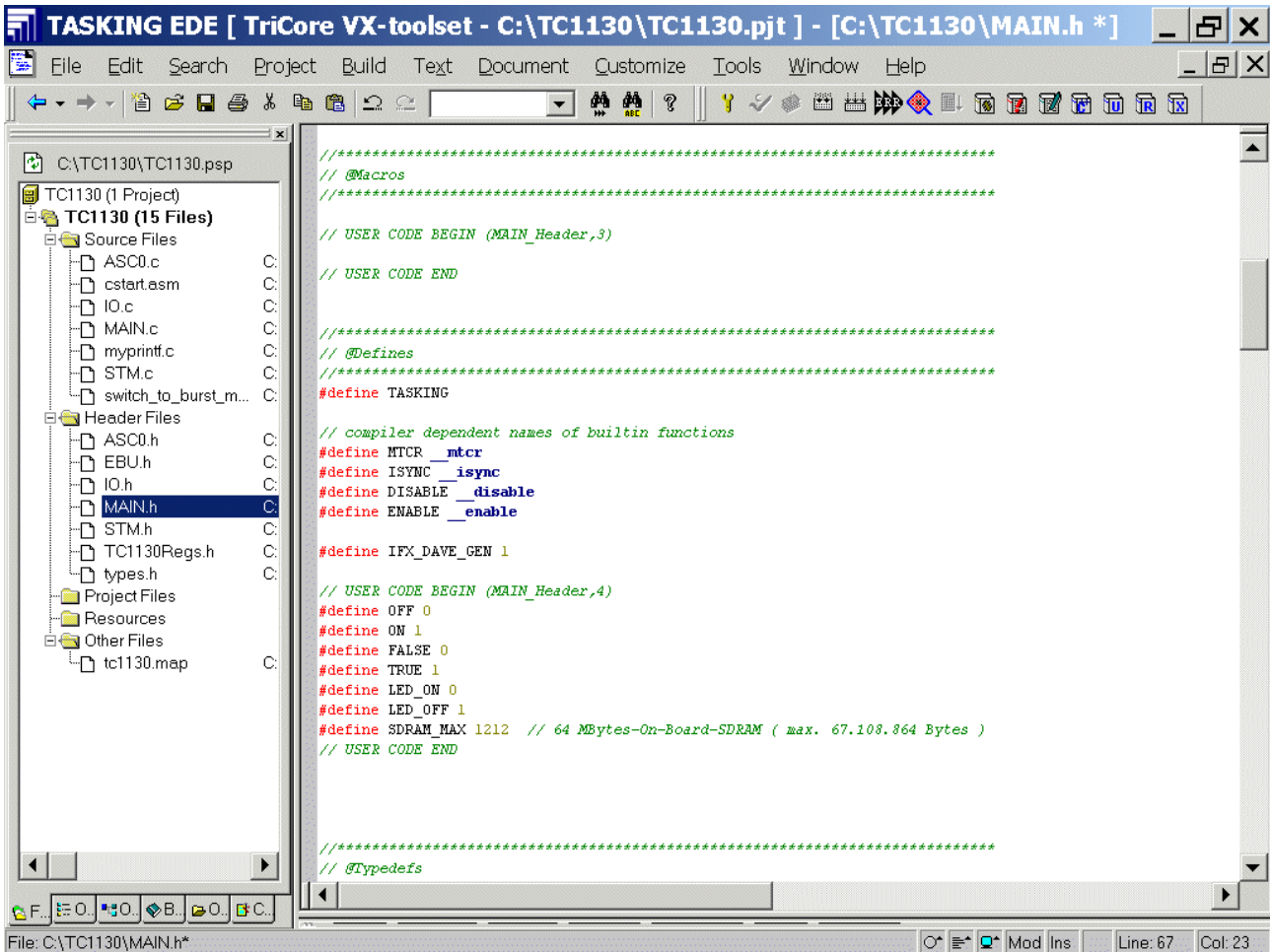
    switch(RESET_INDICATOR)
    {
        case DEEPSLEEP_WAKEUP: // the last reset was an external triggered
                               // hardware reset in deep sleep mode
                               // (wake-up reset)
    }
}

```

The status bar at the bottom indicates "Symbol definition not found." and shows the current position as Line: 518, Col: 99.

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

```
#define FALSE 0
#define TRUE 1
```



The screenshot shows the TASKING EDE IDE interface. The left pane displays a project tree for 'TC1130 (1 Project)' with 'MAIN.h' selected under 'Header Files'. The main editor window shows the content of 'MAIN.h', which includes various macros and defines. The defines section is highlighted in red text.

```

//*****
// @Macros
//*****

// USER CODE BEGIN (MAIN_Header,3)

// USER CODE END

//*****
// @Defines
//*****
#define TASKING

// compiler dependent names of builtin functions
#define MTCR __mcr
#define ISYNC __isync
#define DISABLE __disable
#define ENABLE __enable

#define IFX_DAVE_GEN 1

// USER CODE BEGIN (MAIN_Header,4)
#define OFF 0
#define ON 1
#define FALSE 0
#define TRUE 1
#define LED_ON 0
#define LED_OFF 1
#define SDRAM_MAX 1212 // 64 Mbytes-On-Board-SDRAM ( max. 67.108.864 Bytes )
// USER CODE END

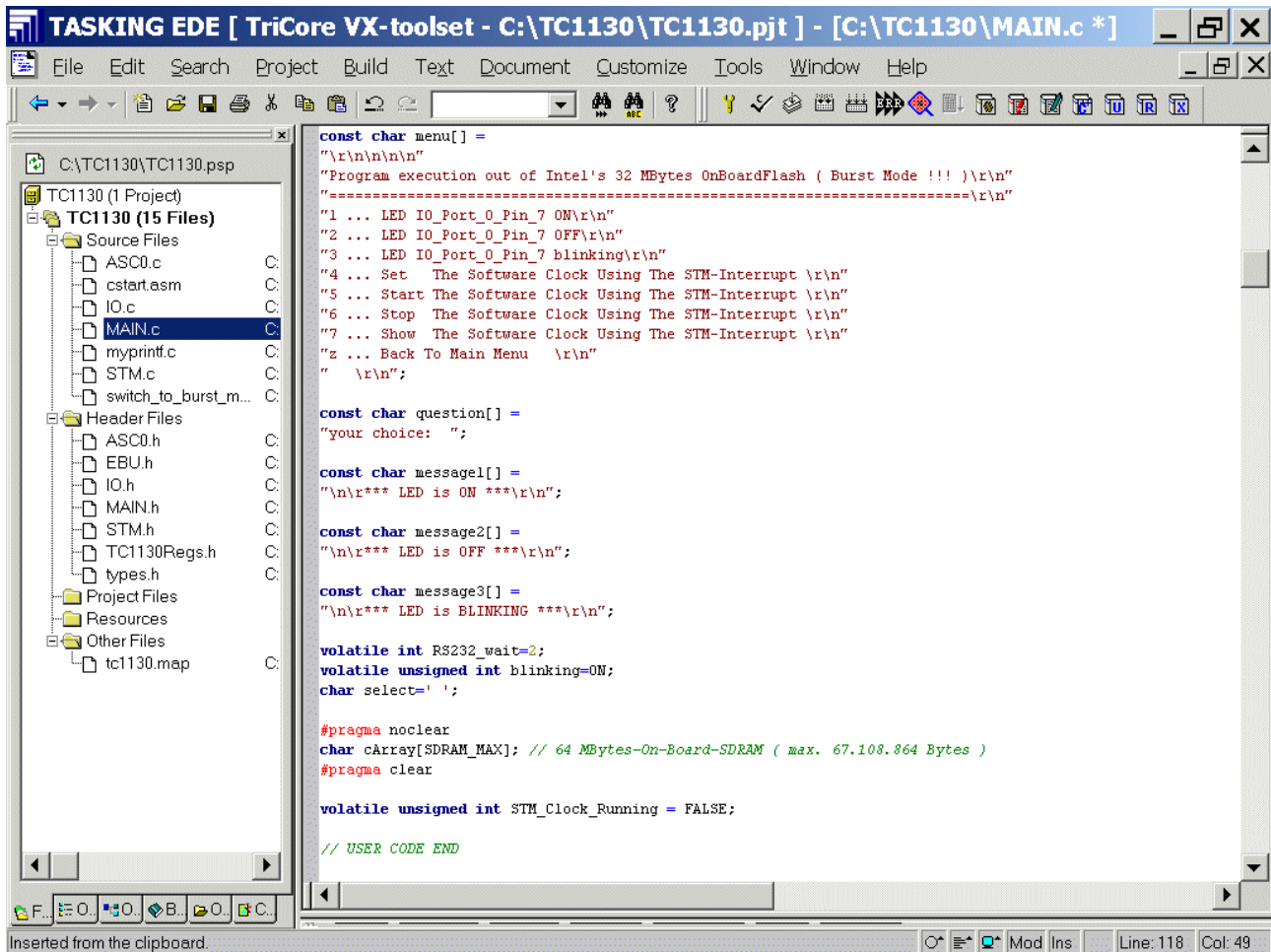
//*****
// @Typedefs

```

File: C:\TC1130\MAIN.h* | Line: 67 | Col: 23

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

```
volatile unsigned int STM_Clock_Running = FALSE;
```



```

const char menu[] =
"\r\n\r\n\r\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"
"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";

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

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

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

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

volatile int RS232_wait=2;
volatile unsigned int blinking=0W;
char select=' ';

#pragma noclear
char cArray[SDRAM_MAX]; // 64 MBytes-On-Board-SDRAM ( max. 67.108.864 Bytes )
#pragma clear

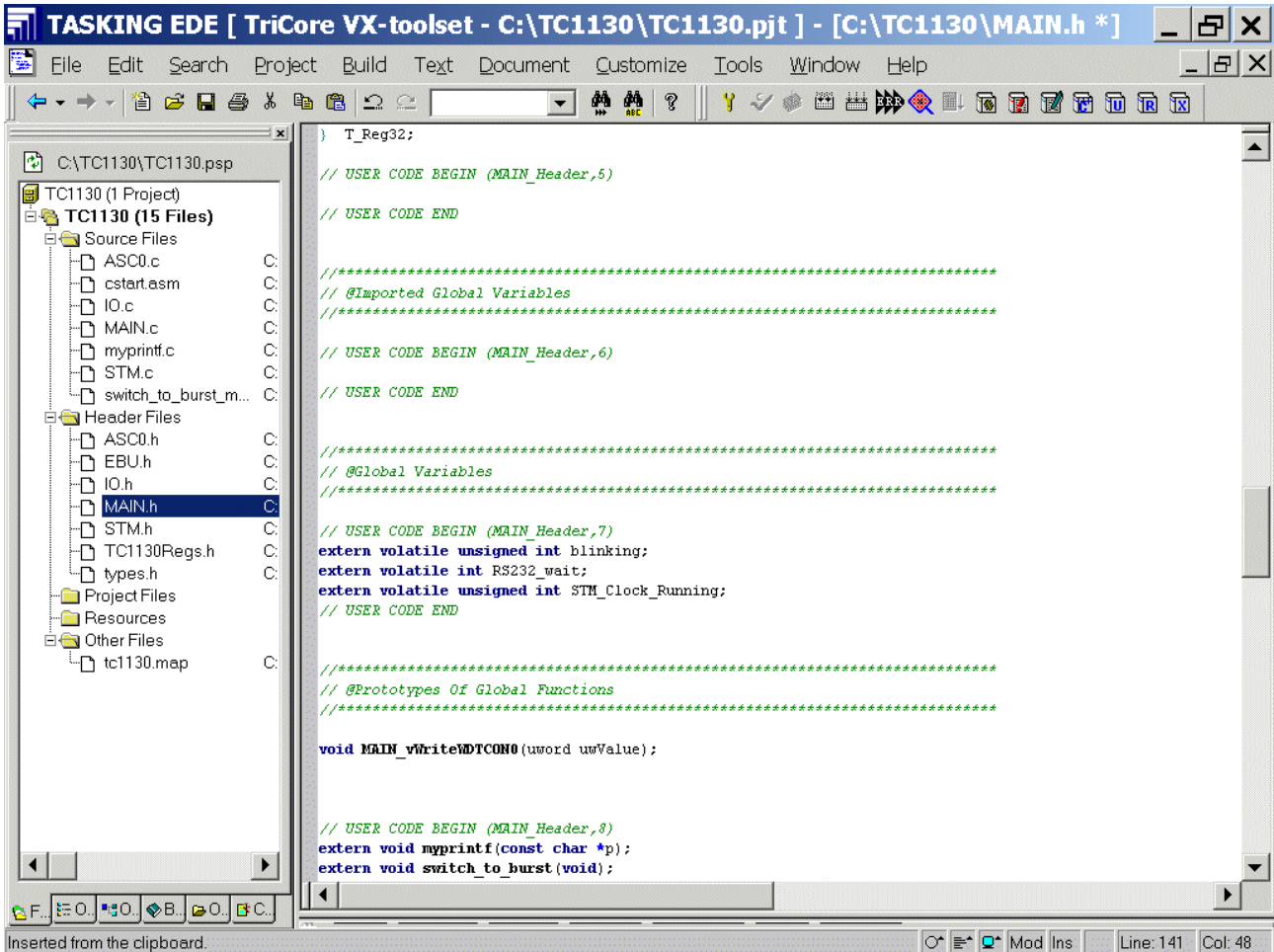
volatile unsigned int STM_Clock_Running = FALSE;

// USER CODE END

```

Double click: **Main.h** and insert Global Variables (Extern Declarations):

```
extern volatile unsigned int STM_Clock_Running;
```



The screenshot shows the TASKING EDE IDE interface. On the left, the project tree displays the file structure for TC1130, with **MAIN.h** selected under the Header Files folder. The main editor window shows the content of **MAIN.h**, which includes several sections: **USER CODE BEGIN (MAIN_Header,5)**, **@Imported Global Variables**, **USER CODE BEGIN (MAIN_Header,6)**, **USER CODE END**, **@Global Variables**, **USER CODE BEGIN (MAIN_Header,7)**, and **USER CODE END**. The **@Global Variables** section contains the following extern declarations:

```
extern volatile unsigned int blinking;
extern volatile int RS232_wait;
extern volatile unsigned int STM_Clock_Running;
```

The **USER CODE BEGIN (MAIN_Header,7)** section contains the function prototype:

```
void MAIN_writeDTCOH0(uword uwValue);
```

The **USER CODE BEGIN (MAIN_Header,8)** section contains the following extern declarations:

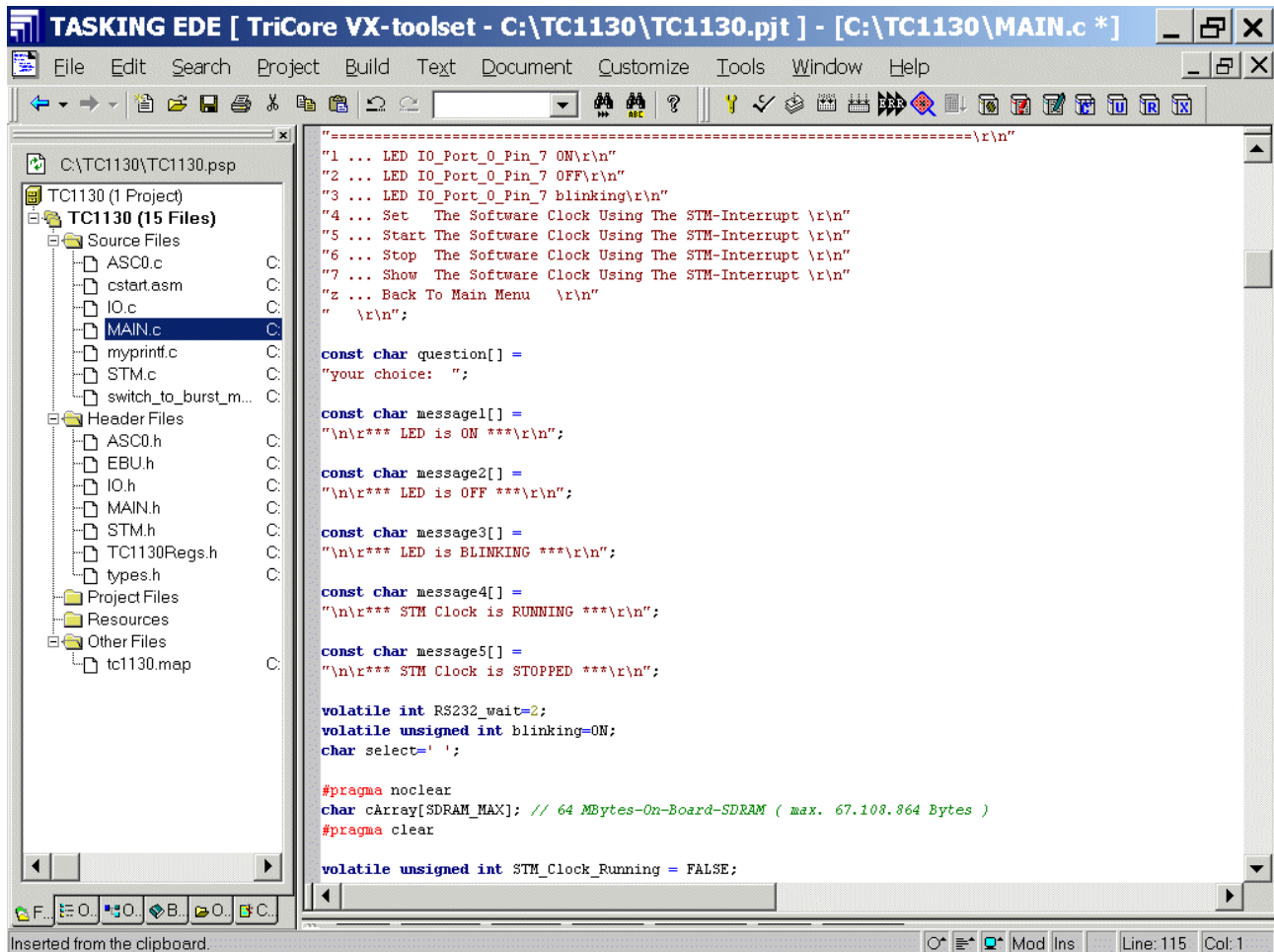
```
extern void myprintf(const char *p);
extern void switch_to_burst(void);
```

The status bar at the bottom indicates "Inserted from the clipboard." and shows the current cursor position as "Line: 141 Col: 48".

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

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

const char message5[] =
"\n\r*** STM Clock is STOPPED ***\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";

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

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

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

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

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

const char message5[] =
"\n\r*** STM Clock is STOPPED ***\r\n";

volatile int RS232_wait=2;
volatile unsigned int blinking=0N;
char select=' ';

#pragma noclear
char cArray[SDRAM_MAX]; // 64 MBytes-On-Board-SDRAM ( max. 67.108.864 Bytes )
#pragma clear

volatile unsigned int STM_Clock_Running = FALSE;

```

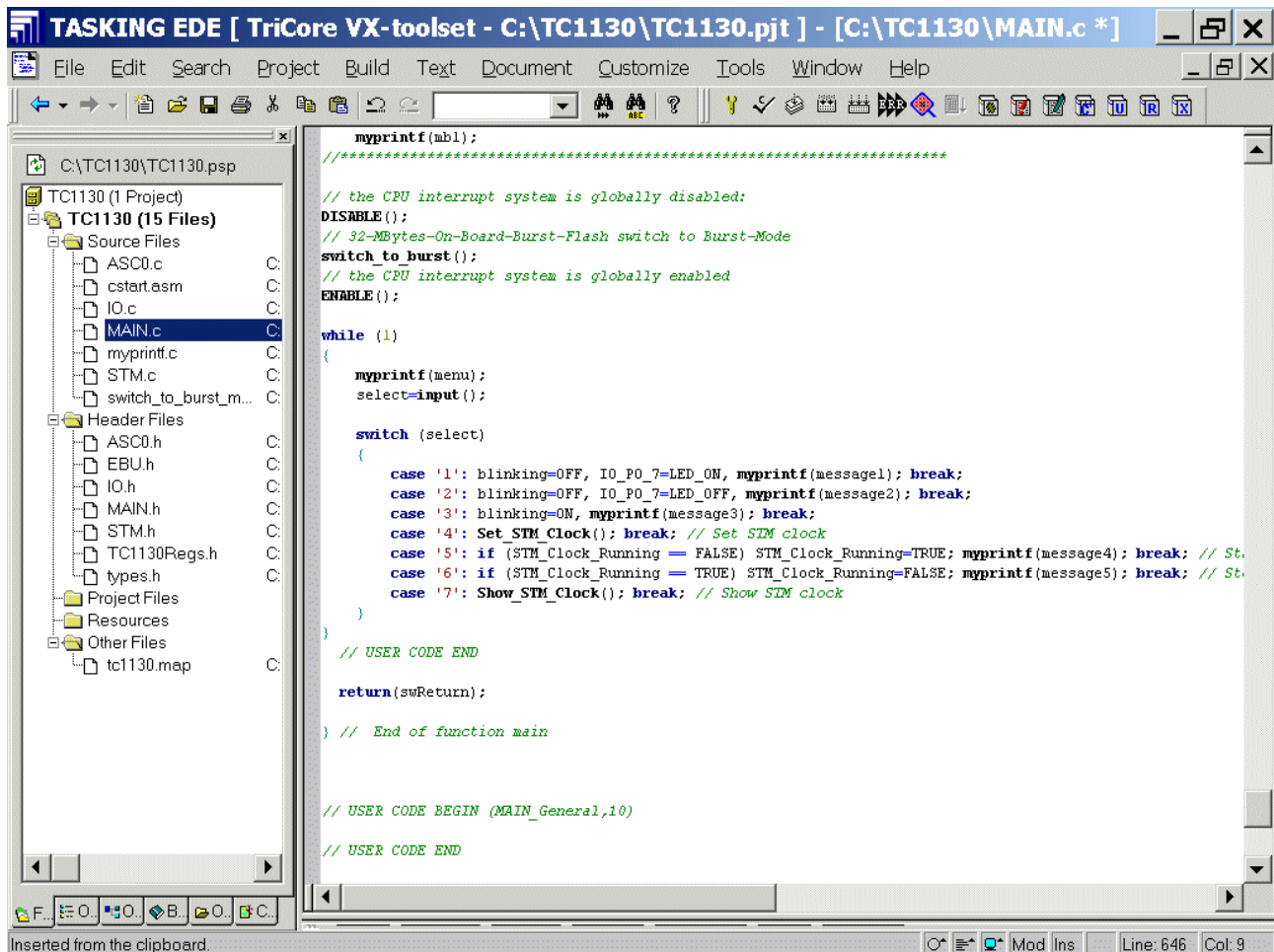
Inserted from the clipboard. Line: 115 Col: 1

Double click: **Main.c** insert User Code [void main (void) - Function]:

```

        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

```



```

TASKING EDE [ TriCore VX-toolset - C:\TC1130\TC1130.pjt ] - [C:\TC1130\MAIN.c *]
File Edit Search Project Build Text Document Customize Tools Window Help
C:\TC1130\TC1130.psp
TC1130 (1 Project)
  TC1130 (15 Files)
    Source Files
      ASC0.c
      cstart.asm
      IO.c
      MAIN.c
      myprintf.c
      STM.c
      switch_to_burst_m...
    Header Files
      ASC0.h
      EBU.h
      IO.h
      MAIN.h
      STM.h
      TC1130Regs.h
      types.h
    Project Files
    Resources
    Other Files
      tc1130.map
myprintf(mbl);
//*****
// 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();

while (1)
{
  myprintf(menu);
  select=input();

  switch (select)
  {
    case '1': blinking=OFF, IO_PO_7=LED_ON, myprintf(message1); break;
    case '2': blinking=OFF, IO_PO_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; // St
    case '6': if (STM_Clock_Running == TRUE) STM_Clock_Running=FALSE; myprintf(message5); break; // St
    case '7': Show_STM_Clock(); break; // Show STM clock
  }
}
// USER CODE END

return(swReturn);
} // End of function main

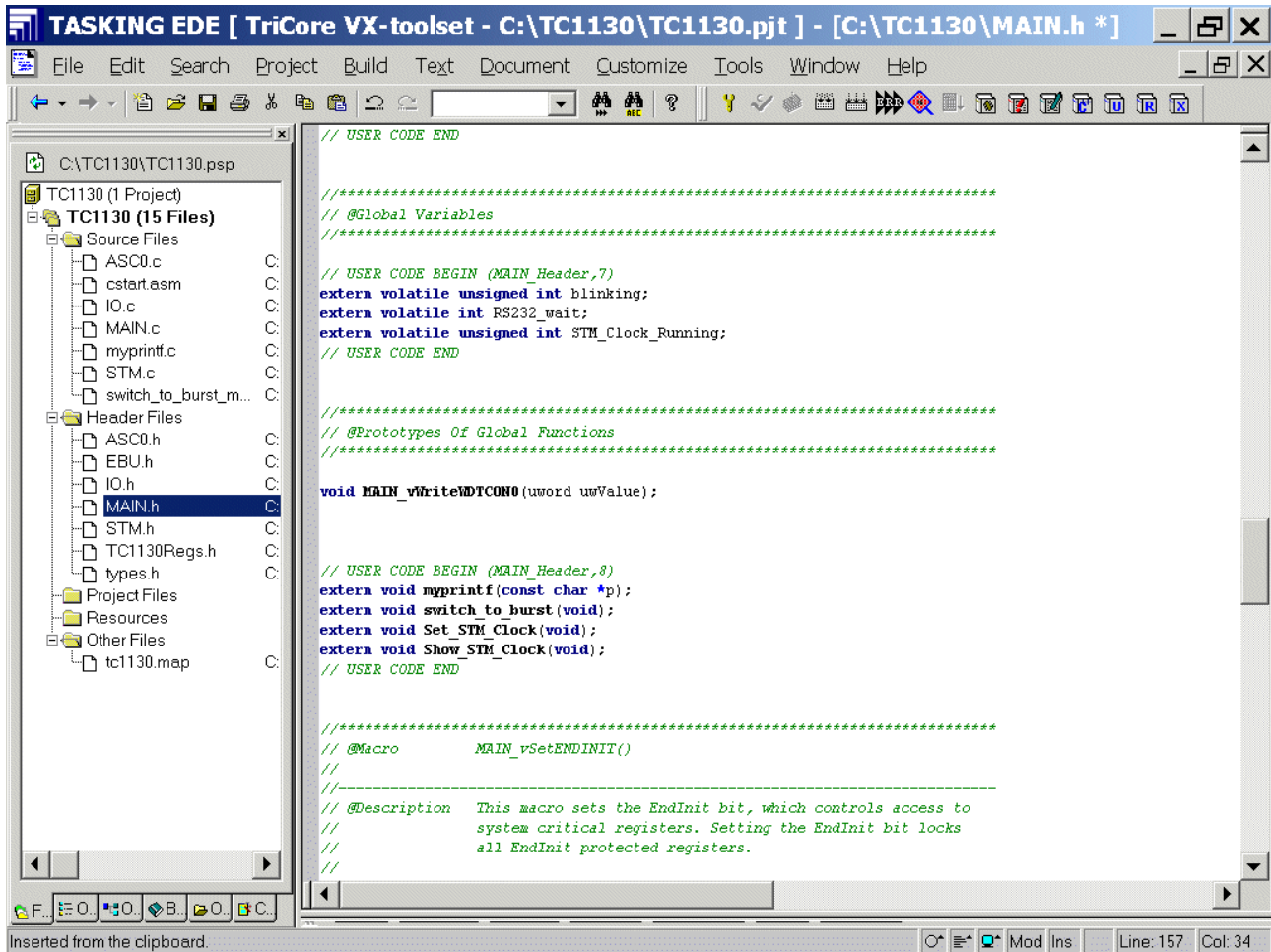
// USER CODE BEGIN (MAIN_General,10)
// USER CODE END

```

Inserted from the clipboard. Mod Ins Line: 646 Col: 9

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

```
extern void Set_STM_Clock(void);
extern void Show_STM_Clock(void);
```



The screenshot shows the TASKING EDE IDE interface. The left pane displays a project tree for 'TC1130 (1 Project)' with 15 files. The 'MAIN.h' file is selected. The main editor window shows the following code:

```
// USER CODE END

//*****
// @Global Variables
//*****

// USER CODE BEGIN (MAIN Header,7)
extern volatile unsigned int blinking;
extern volatile int RS232_wait;
extern volatile unsigned int STM_Clock_Running;
// USER CODE END

//*****
// @Prototypes Of Global Functions
//*****

void MAIN_vWriteWDTCON0(uword uwValue);

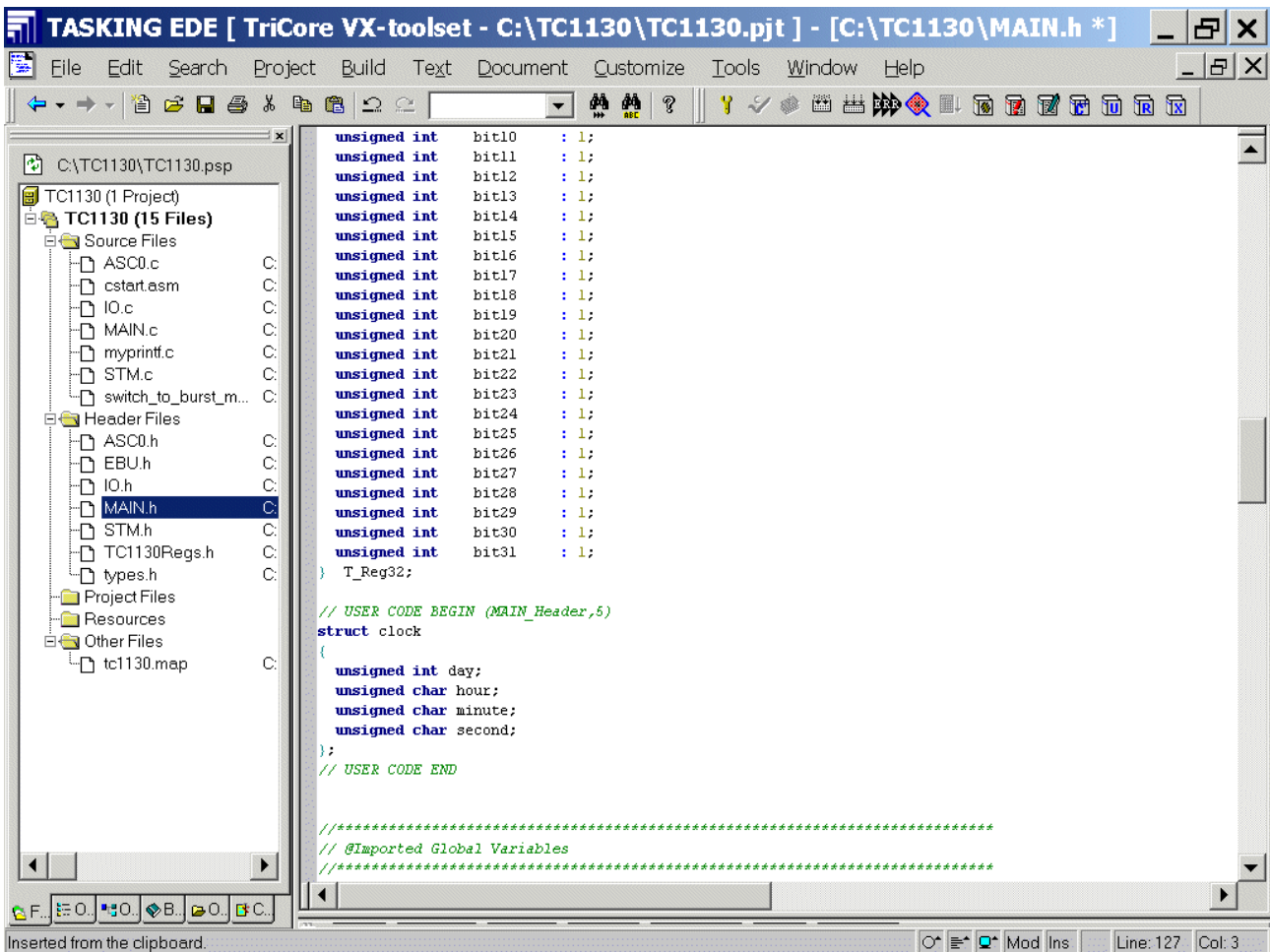
// USER CODE BEGIN (MAIN Header,8)
extern void myprintf(const char *p);
extern void switch_to_burst(void);
extern void Set_STM_Clock(void);
extern void Show_STM_Clock(void);
// USER CODE END

//*****
// @Macro    MAIN_vSetENDINIT()
//
//-----
// @Description  This macro sets the EndInit bit, which controls access to
//                system critical registers. Setting the EndInit bit locks
//                all EndInit protected registers.
//
```

At the bottom of the IDE, a status bar indicates 'Line: 157 Col: 34' and a message 'Inserted from the clipboard.'

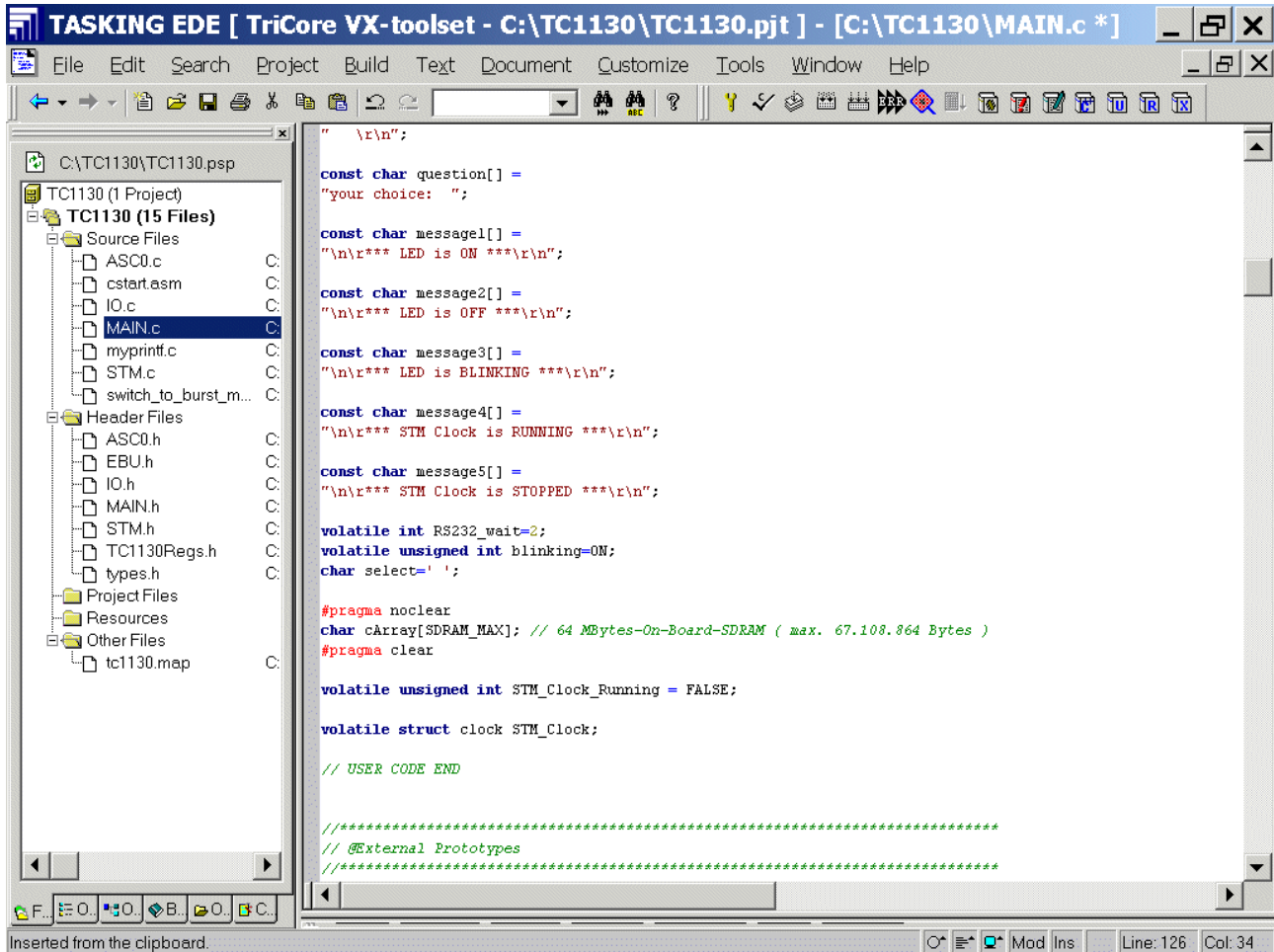
Double click: [Main.h](#) and insert Typedefs (struct Definition):

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



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

```
volatile struct clock STM_Clock;
```



```

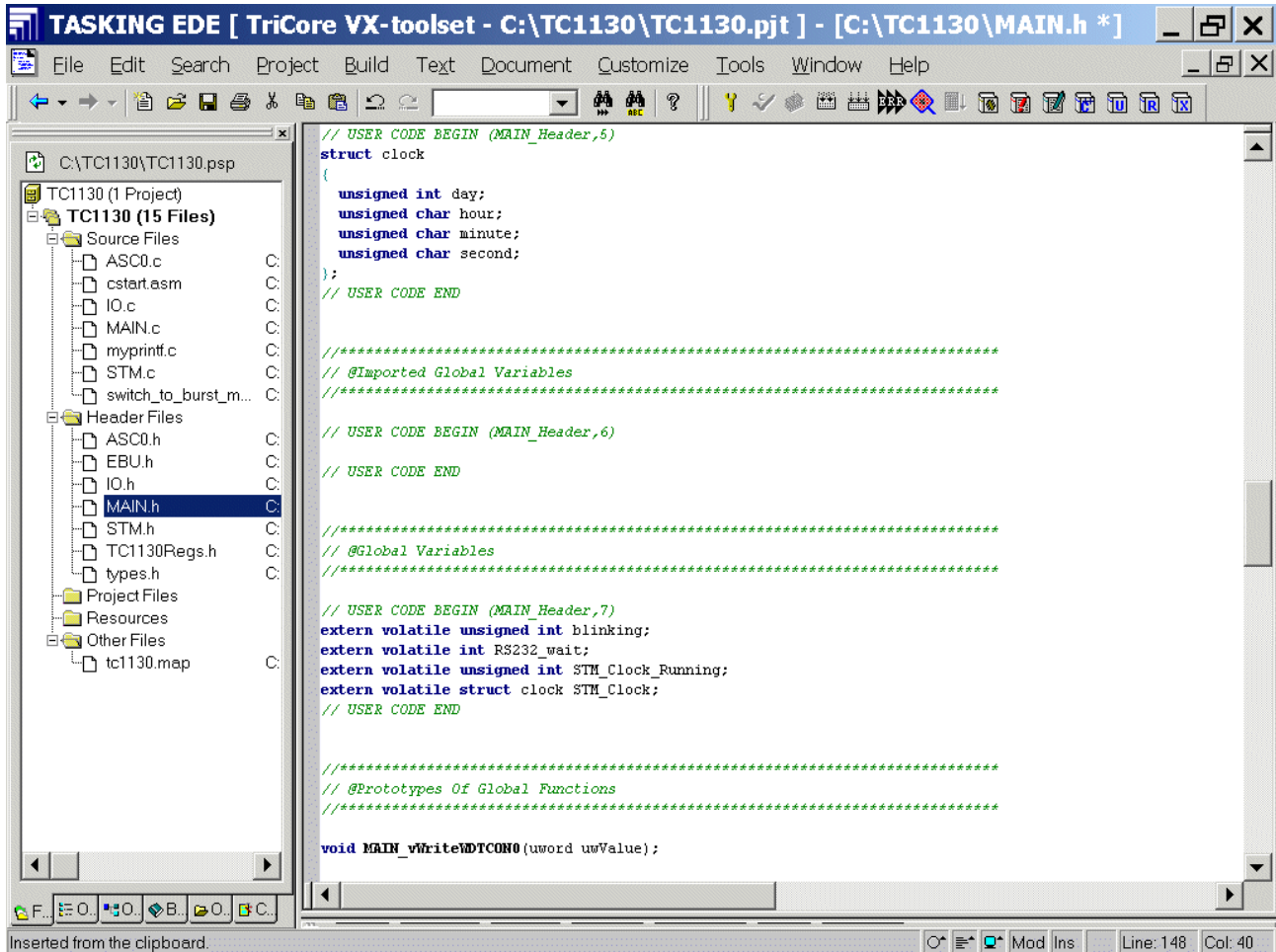
TASKING EDE [ TriCore VX-toolset - C:\TC1130\TC1130.pjt ] - [C:\TC1130\MAIN.c *]
File Edit Search Project Build Text Document Customize Tools Window Help
C:\TC1130\TC1130.psp
TC1130 (1 Project)
  TC1130 (15 Files)
    Source Files
      ASC0.c
      cstart.asm
      IO.c
      MAIN.c
      myprintf.c
      STM.c
      switch_to_burst_m...
    Header Files
      ASC0.h
      EBU.h
      IO.h
      MAIN.h
      STM.h
      TC1130Regs.h
      types.h
    Project Files
    Resources
    Other Files
      tc1130.map
" \r\n";
const char question[] =
"your choice: ";
const char message1[] =
"\n\r*** LED is ON ***\r\n";
const char message2[] =
"\n\r*** LED is OFF ***\r\n";
const char message3[] =
"\n\r*** LED is BLINKING ***\r\n";
const char message4[] =
"\n\r*** STH Clock is RUNNING ***\r\n";
const char message5[] =
"\n\r*** STH Clock is STOPPED ***\r\n";
volatile int RS232_wait=2;
volatile unsigned int blinking=0N;
char select=' ';
#pragma noclear
char cArray[SDRAM_MAX]; // 64 MBytes-On-Board-SDRAM ( max. 67.108.864 Bytes )
#pragma clear
volatile unsigned int STM_Clock_Running = FALSE;
volatile struct clock STM_Clock;
// USER CODE END
//*****
// @External Prototypes
//*****

```

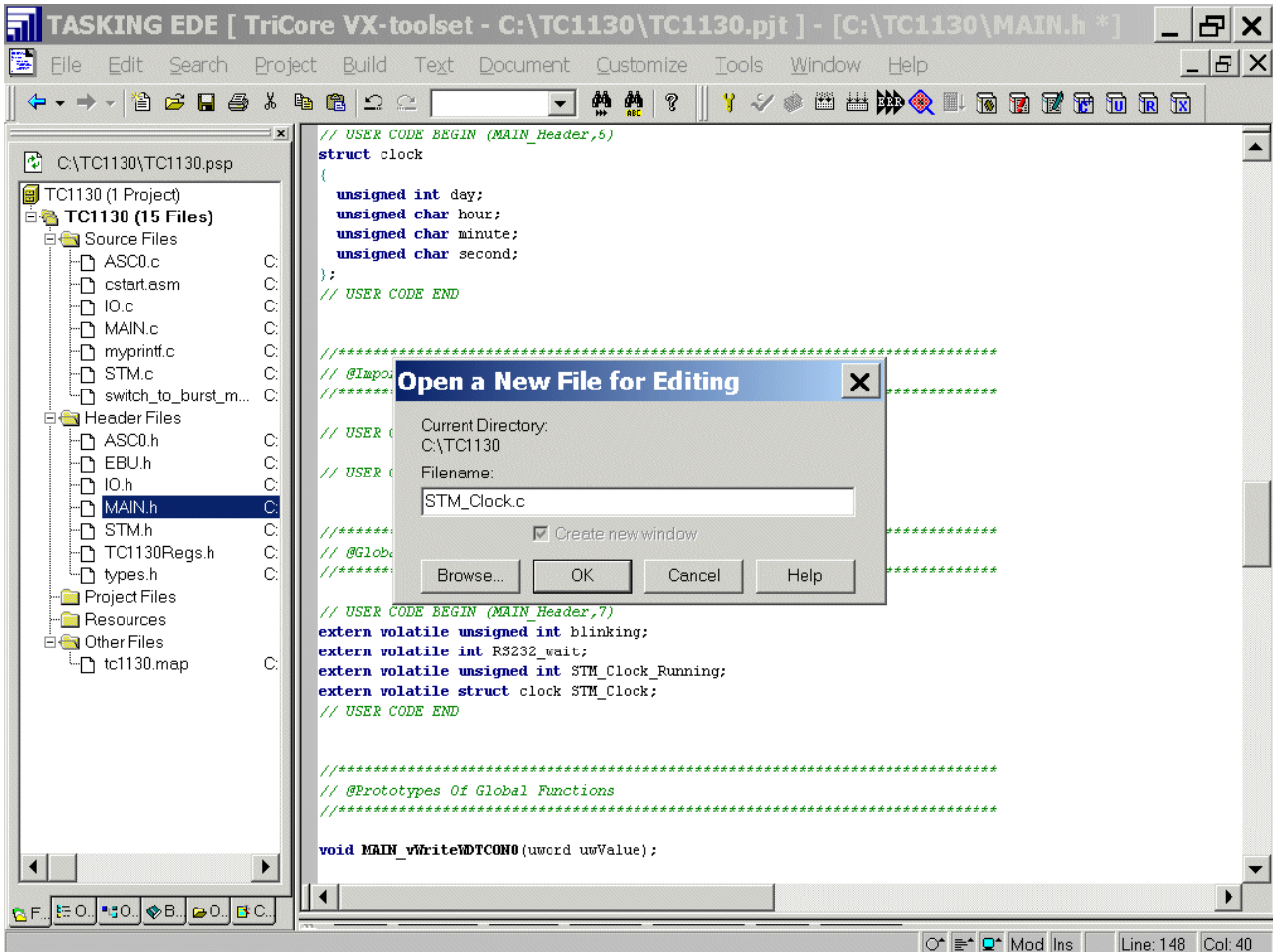
Inserted from the clipboard. Mod Ins Line: 126 Col: 34

Double click: **Main.h** and **insert** Extern Declaration:

extern volatile struct clock STM_Clock;



File – New
Insert STM_Clock.c



OK

Insert User Code:

```

#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()

    help_hour[2]=help_minute[2]=help_second[2]='\0';

    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\n***** TIME *****\r\n");
    do

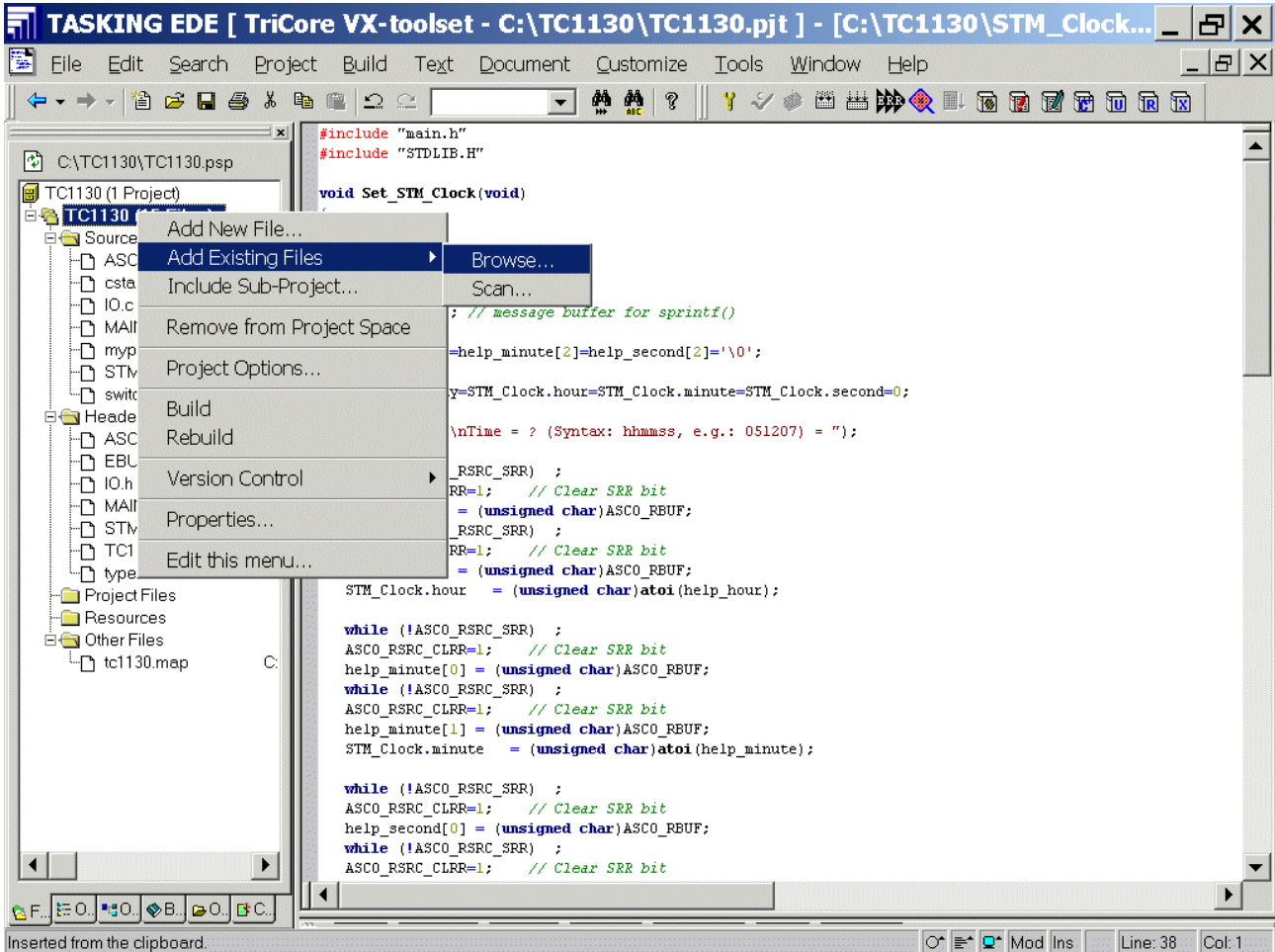
```

```
{
    sprintf(mb, "*** %02u:%02u:%02u
***\r", STM_Clock.hour, STM_Clock.minute, STM_Clock.second);
    myprintf(mb);

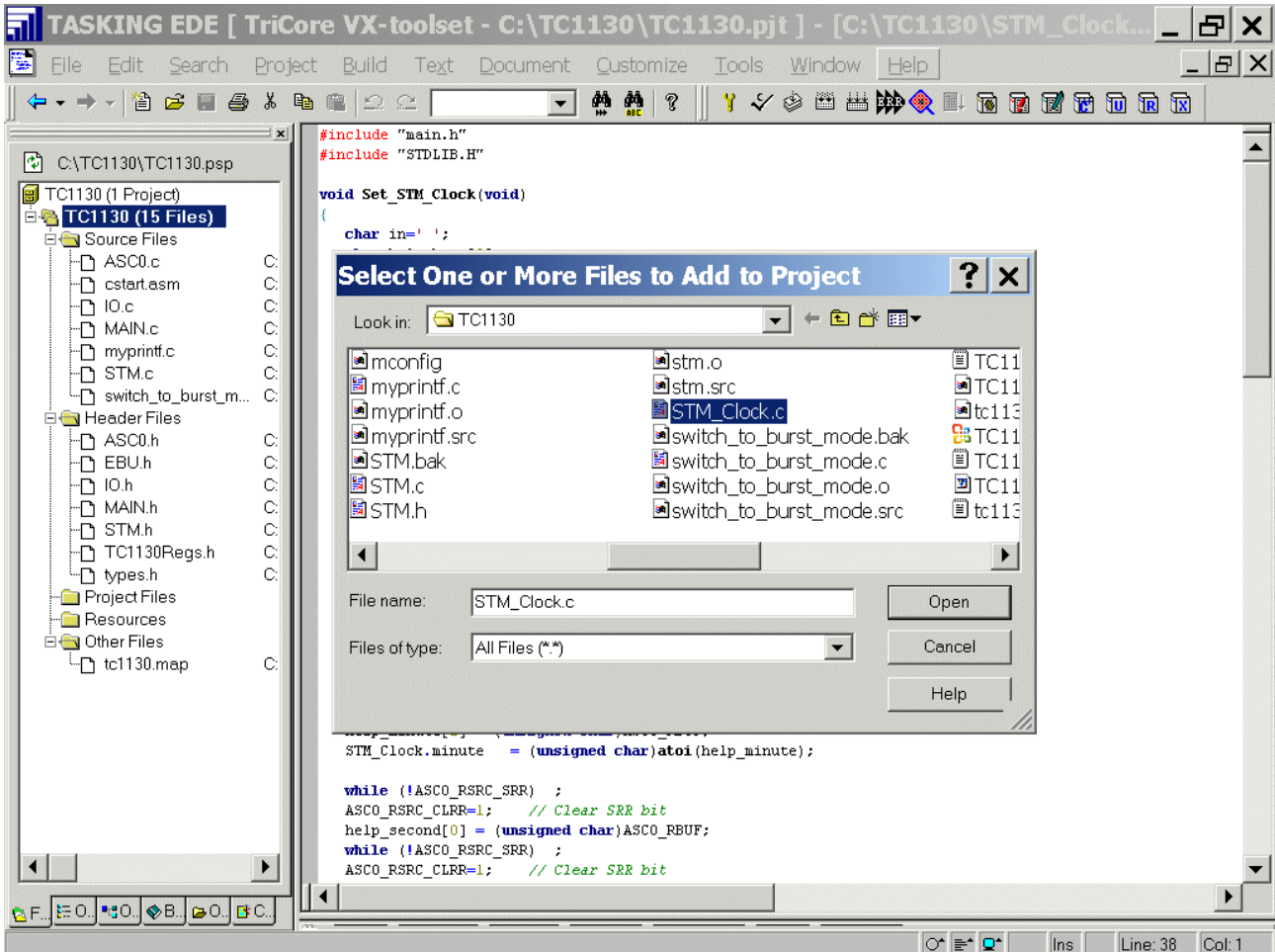
    in = (unsigned char)ASC0_RBUF;
}while (in!='z' && in!= 'Z');
```

File
Save all

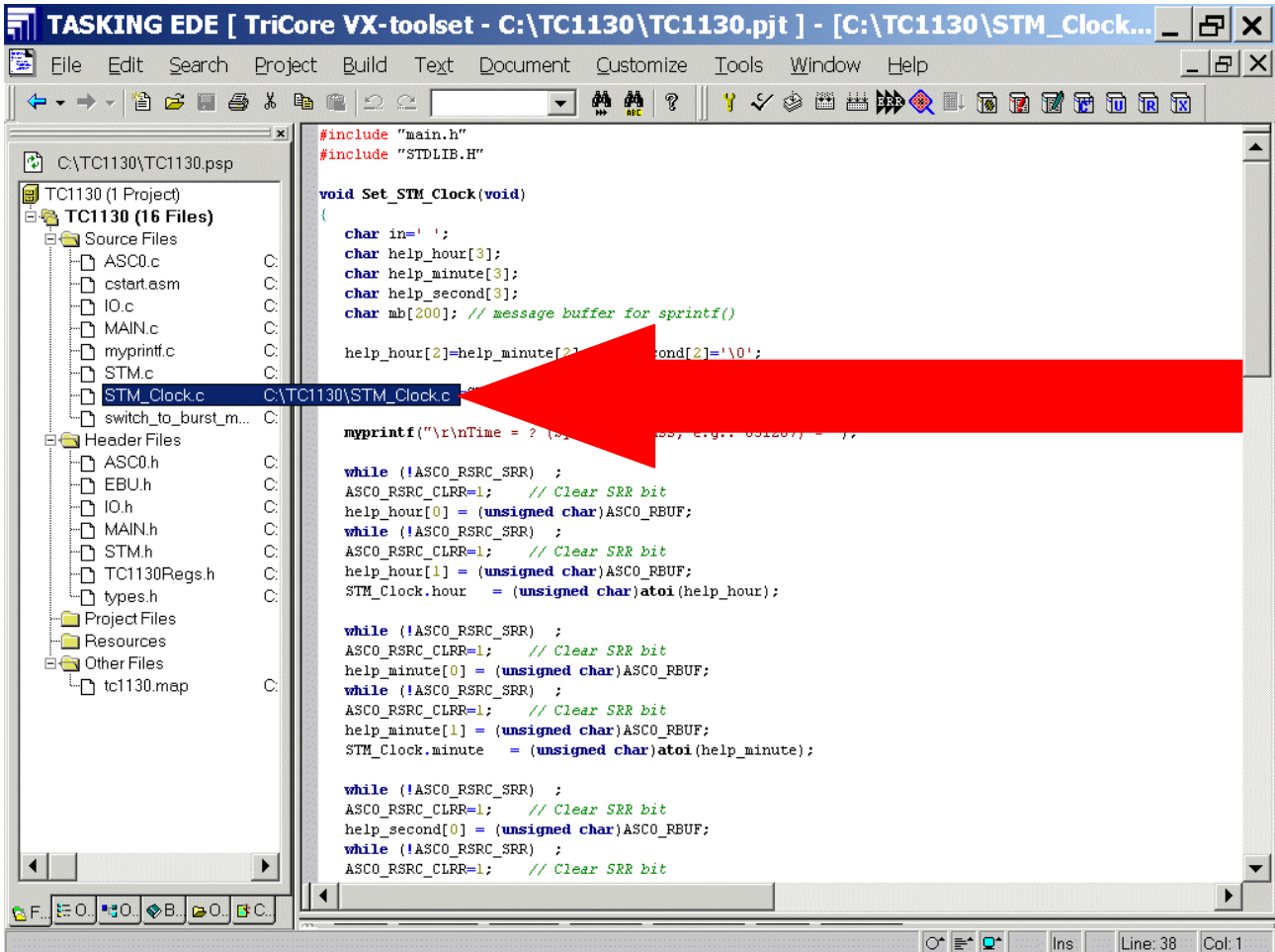
(Project Window **File View**) – TC1130 (Files) – **right mouse button click** – Add Existing Files – Browse



Select STM_Clock.c



Open - OK

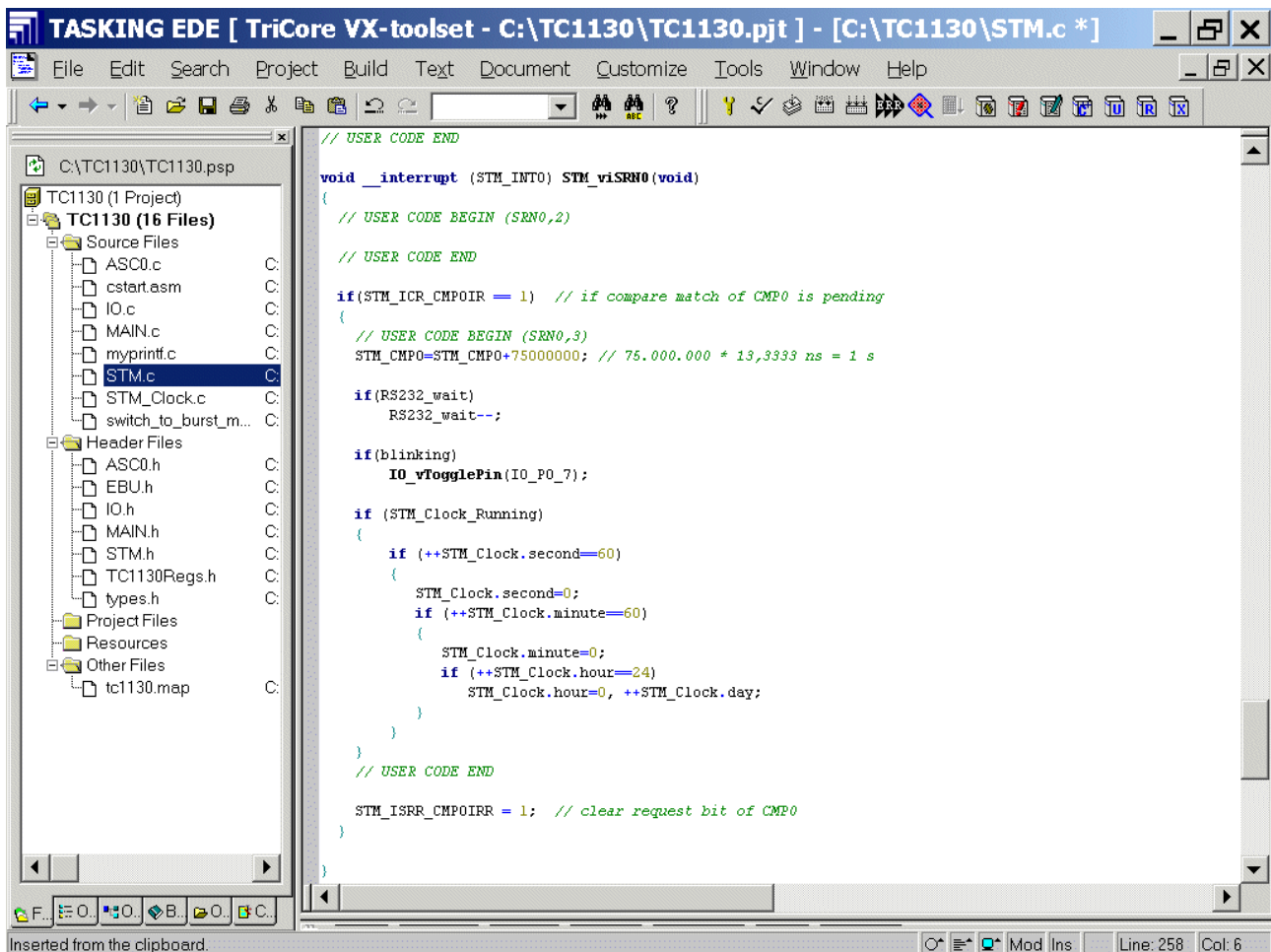


Double click: **STM.c** insert User Code for system-timer's interrupt-service-routine:

```

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;
        }
    }
}

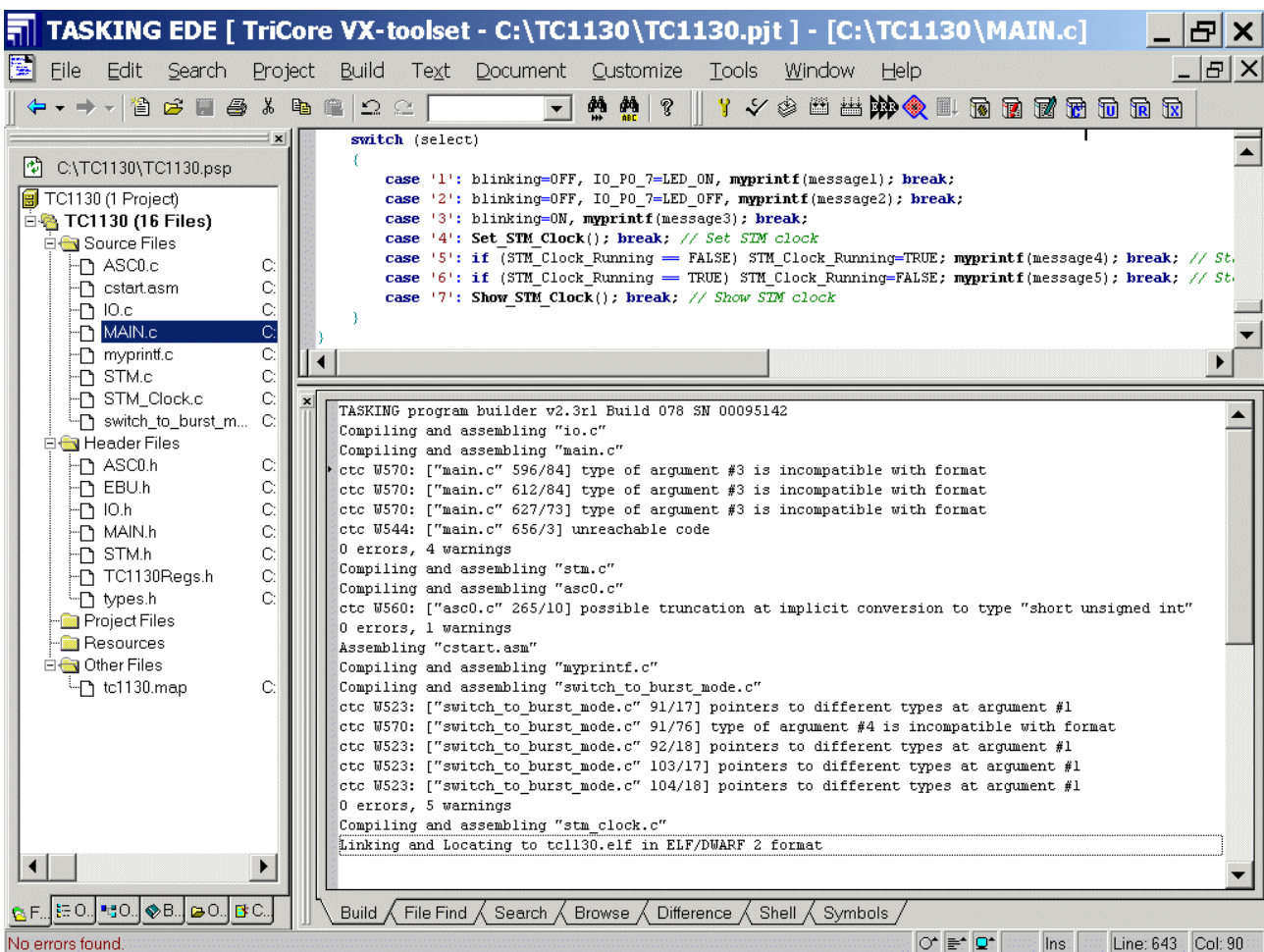
```



Generate your application program:

Build
Rebuild

or



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

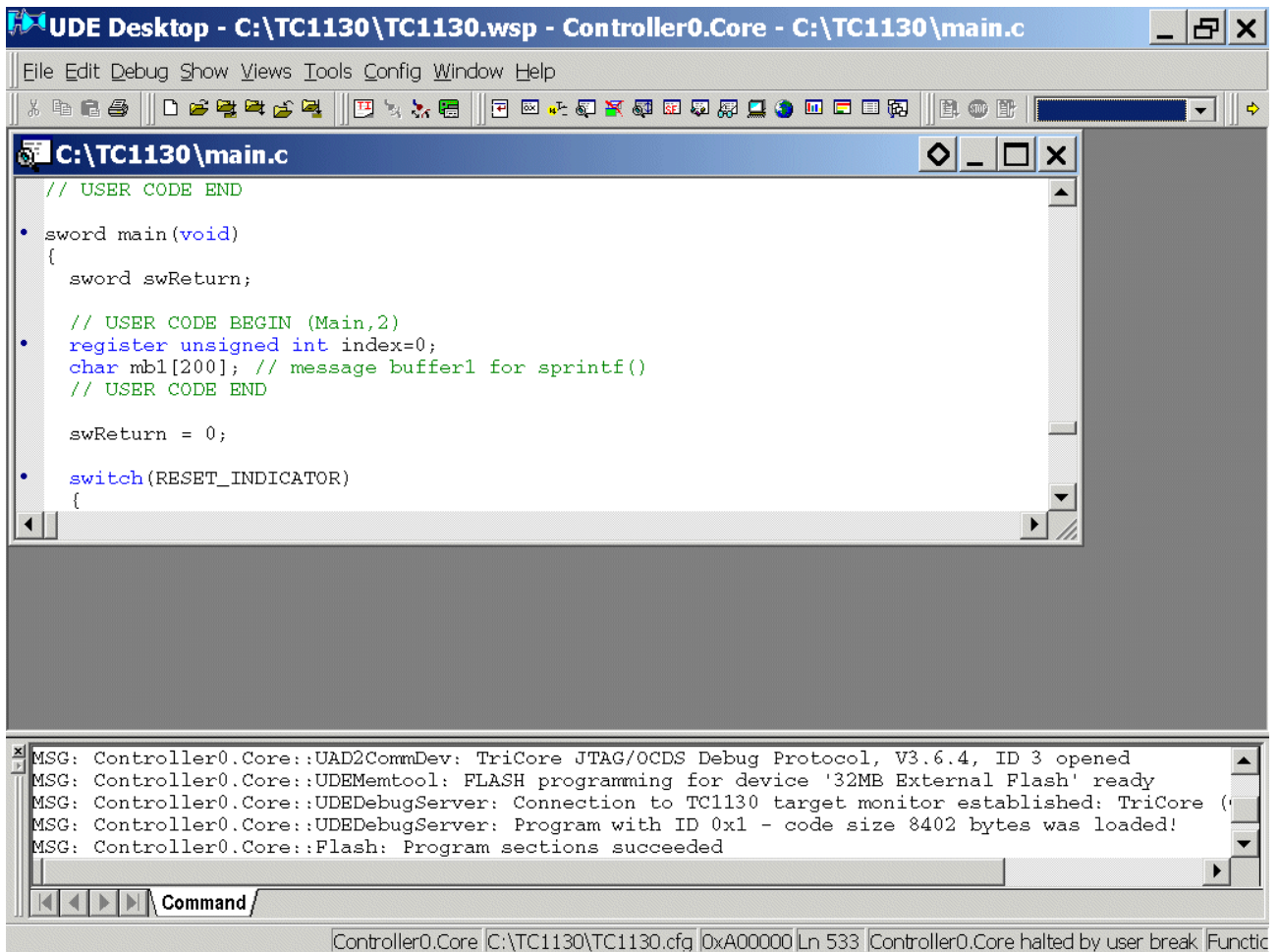
Open

Load

Click Program All

Exit

Exit



The screenshot shows the UDE Desktop IDE interface. The title bar reads "UDE Desktop - C:\TC1130\TC1130.wsp - Controller0.Core - C:\TC1130\main.c". The menu bar includes "File", "Edit", "Debug", "Show", "Views", "Tools", "Config", "Window", and "Help". The toolbar contains various icons for file operations and debugging. The main editor window displays the following C code:

```

// USER CODE END
sword main(void)
{
    sword swReturn;

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

    swReturn = 0;

    switch(RESET_INDICATOR)
    {

```

Below the editor is a console window showing the following messages:

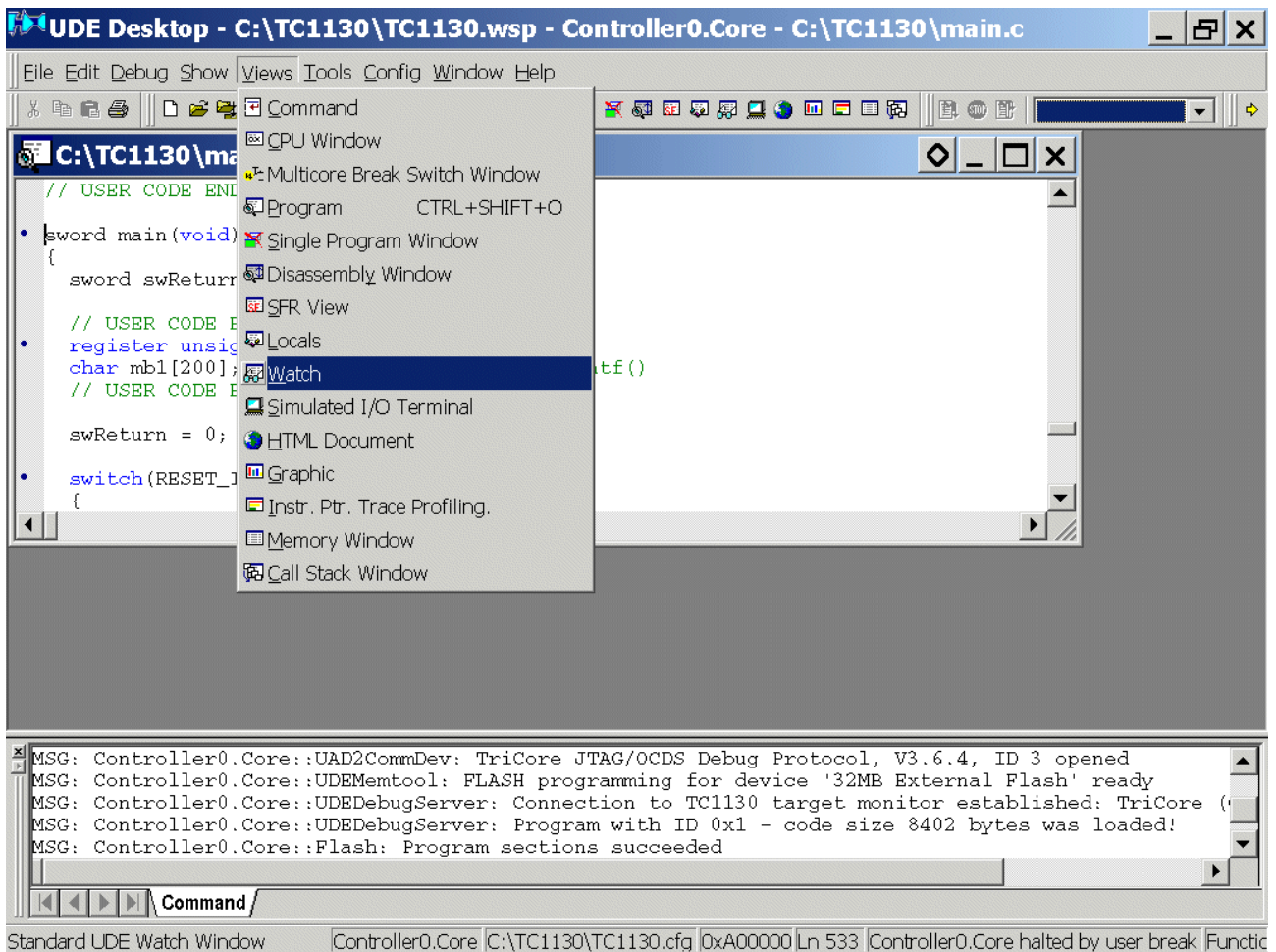
```

MSG: Controller0.Core::UAD2CommDev: TriCore JTAG/OCDS Debug Protocol, V3.6.4, ID 3 opened
MSG: Controller0.Core::UDEMentool: 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

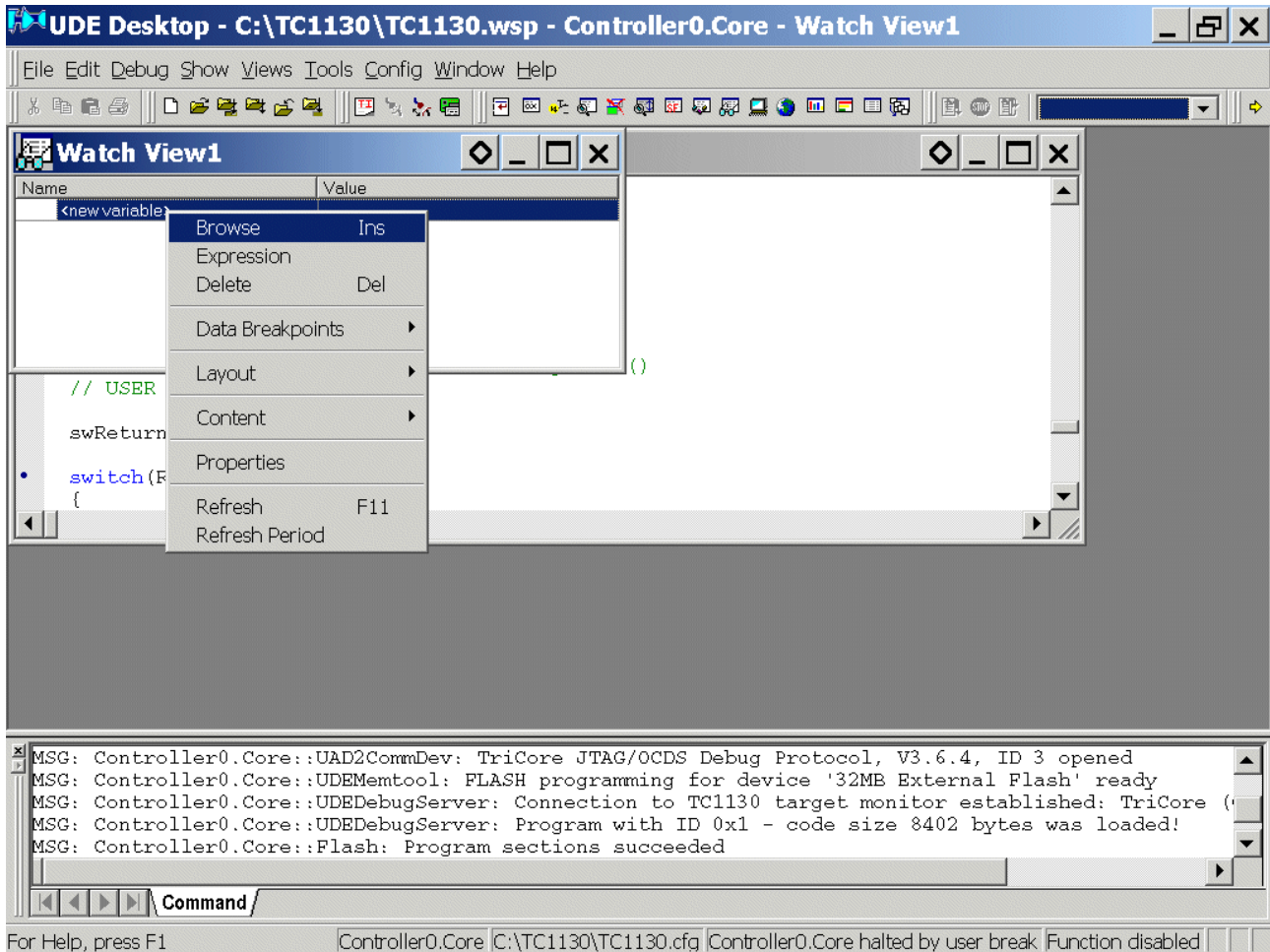
```

The status bar at the bottom indicates: "Controller0.Core C:\TC1130\TC1130.cfg |0xA00000 Ln 533 Controller0.Core halted by user break Functi".

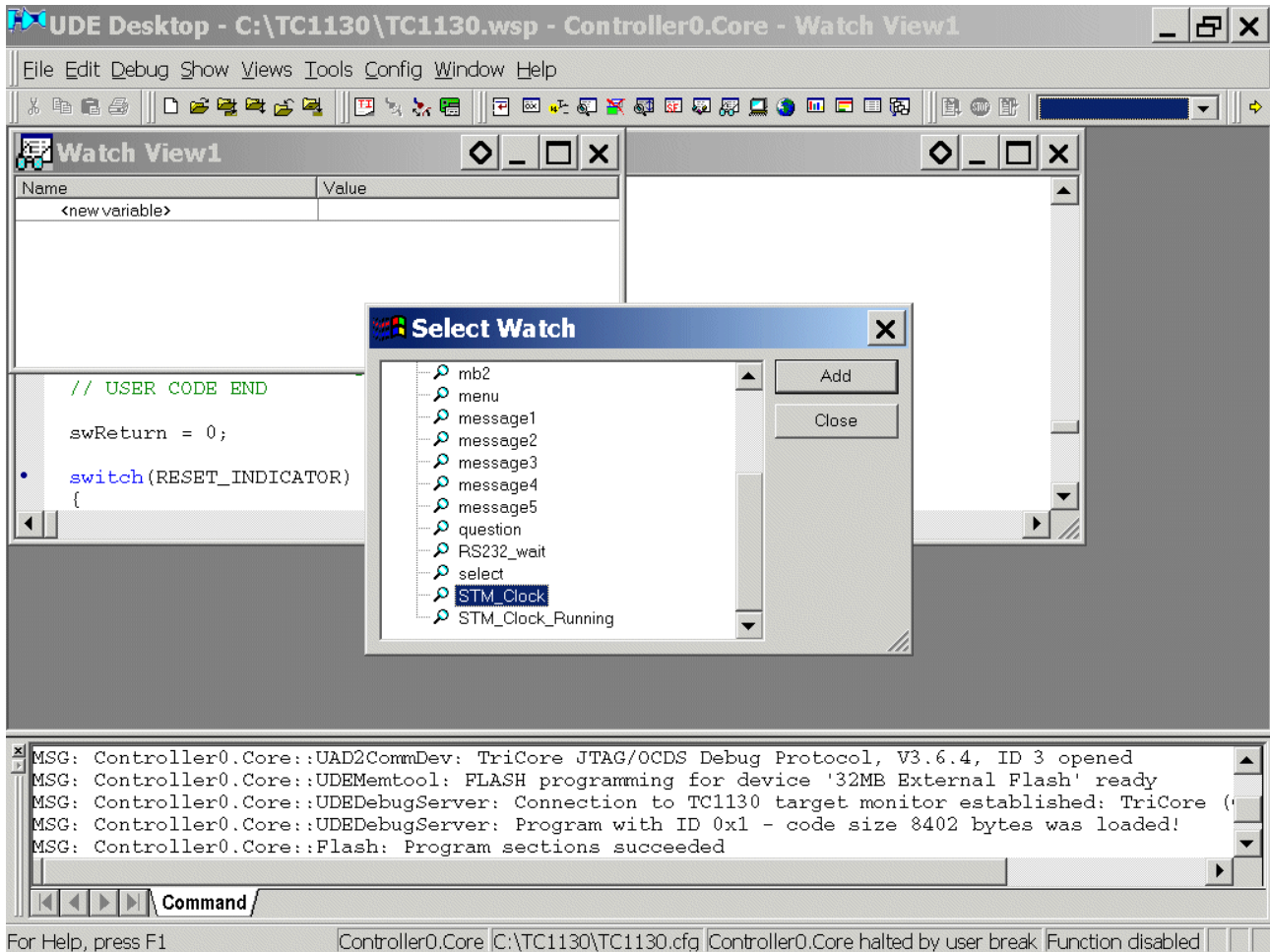
Views – Watch



Right mouse button click at “new variable” select Browse



Select "STM_Clock"



The screenshot shows the UDE Desktop interface for the TC1130 controller. The main window is titled "UDE Desktop - C:\TC1130\TC1130.wsp - Controller0.Core - Watch View1". It features a menu bar (File, Edit, Debug, Show, Views, Tools, Config, Window, Help) and a toolbar. The "Watch View1" window is open, displaying a table with columns "Name" and "Value". Below the table is a code editor showing the following code:

```
// USER CODE END
swReturn = 0;
switch (RESET_INDICATOR)
{
```

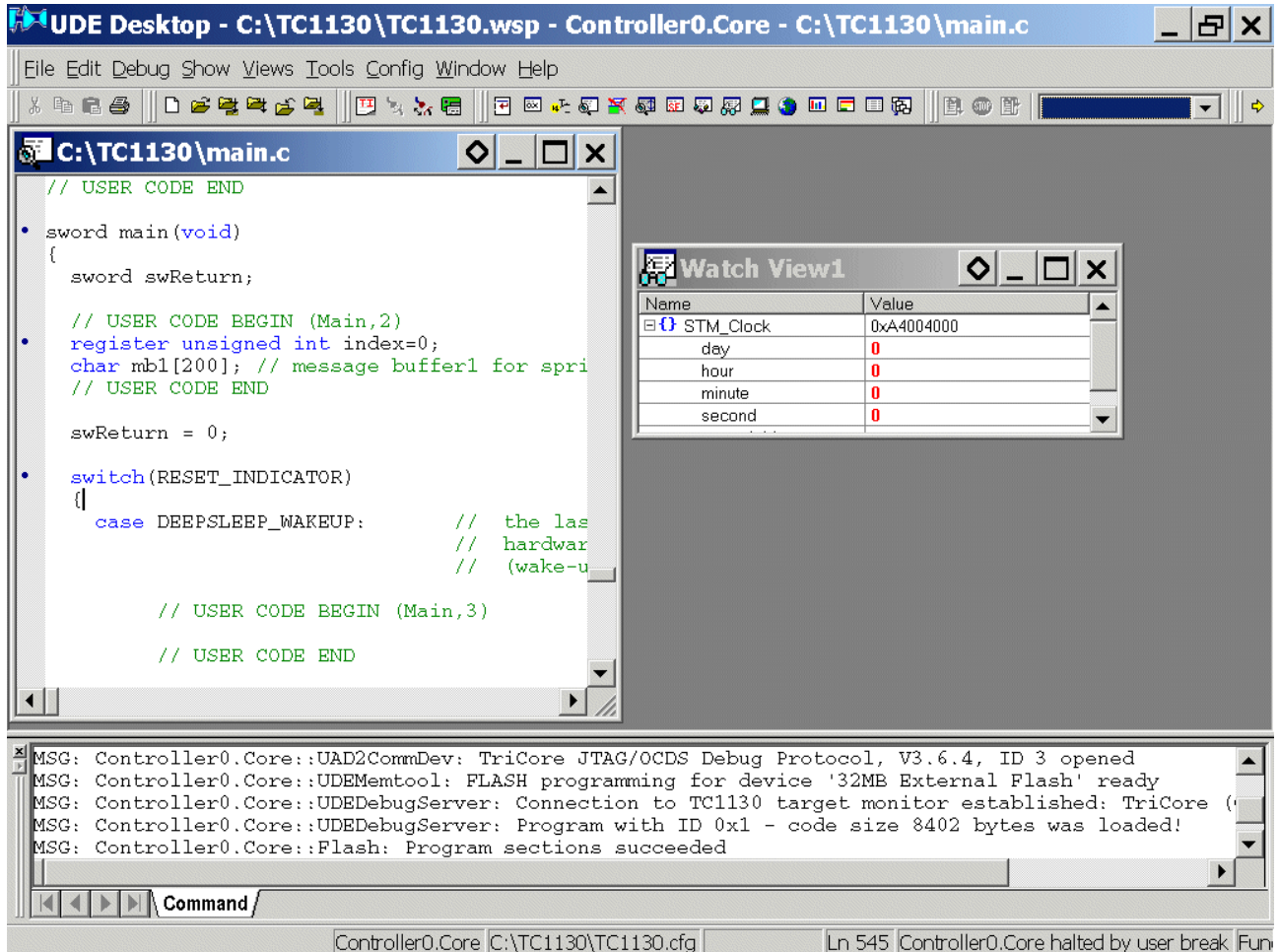
A "Select Watch" dialog box is overlaid on the code editor. It contains a list of variables with checkboxes: mb2, menu, message1, message2, message3, message4, message5, question, RS232_wait, select, **STM_Clock**, and STM_Clock_Running. The "STM_Clock" variable is selected. The dialog has "Add" and "Close" buttons.

At the bottom of the interface is a console window with the following messages:

```
MSG: Controller0.Core::UAD2CommDev: TriCore JTAG/OCDS Debug Protocol, V3.6.4, ID 3 opened
MSG: Controller0.Core::UDEMentool: 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
```

The status bar at the bottom indicates: "For Help, press F1 | Controller0.Core | C:\TC1130\TC1130.cfg | Controller0.Core halted by user break | Function disabled |"

Click Add
Click Close



The screenshot shows the UDE Desktop IDE interface. The main window displays a C program named `main.c` with the following code:

```
// USER CODE END
sword main(void)
{
    sword swReturn;

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

    swReturn = 0;

    switch(RESET_INDICATOR)
    {
        case DEEPSLEEP_WAKEUP: // the las
                               // hardwar
                               // (wake-u

        // USER CODE BEGIN (Main,3)

        // USER CODE END
    }
}
```

A Watch View window titled "Watch View1" is open, displaying the following data:

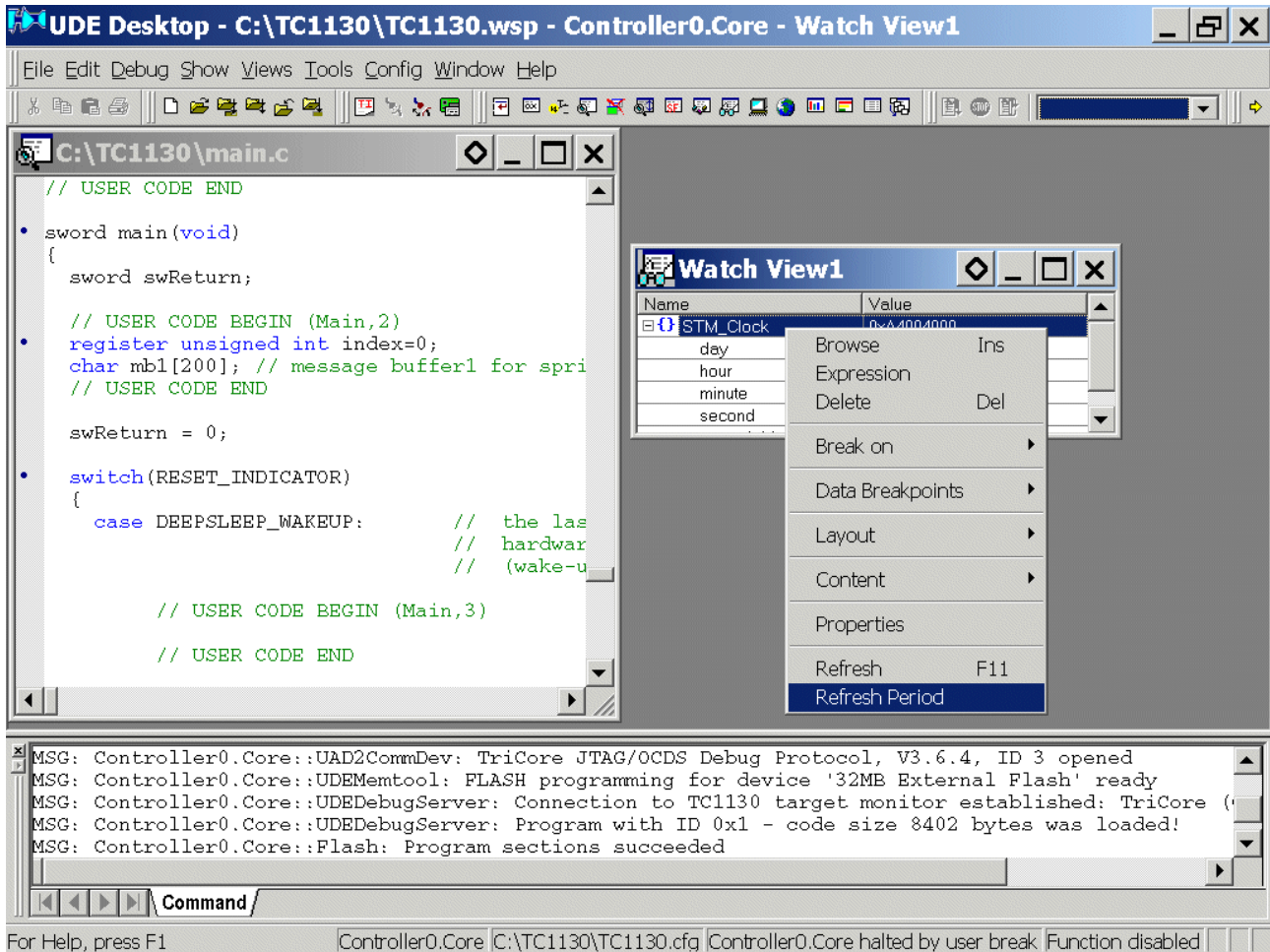
Name	Value
STM_Clock	0xA4004000
day	0
hour	0
minute	0
second	0

The Command window at the bottom shows the following messages:

```
MSG: Controller0.Core::UAD2CommDev: TriCore JTAG/OCDS Debug Protocol, V3.6.4, ID 3 opened
MSG: Controller0.Core::UDEMemtool: 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
```

The status bar at the bottom indicates: Controller0.Core C:\TC1130\TC1130.cfg Ln 545 Controller0.Core halted by user break Fun

Right mouse button click at “STM_Clock” select Refresh Period



The screenshot shows the UDE Desktop interface. The main window displays the C code for `main.c`. A `Watch View1` window is open, showing a table with the following data:

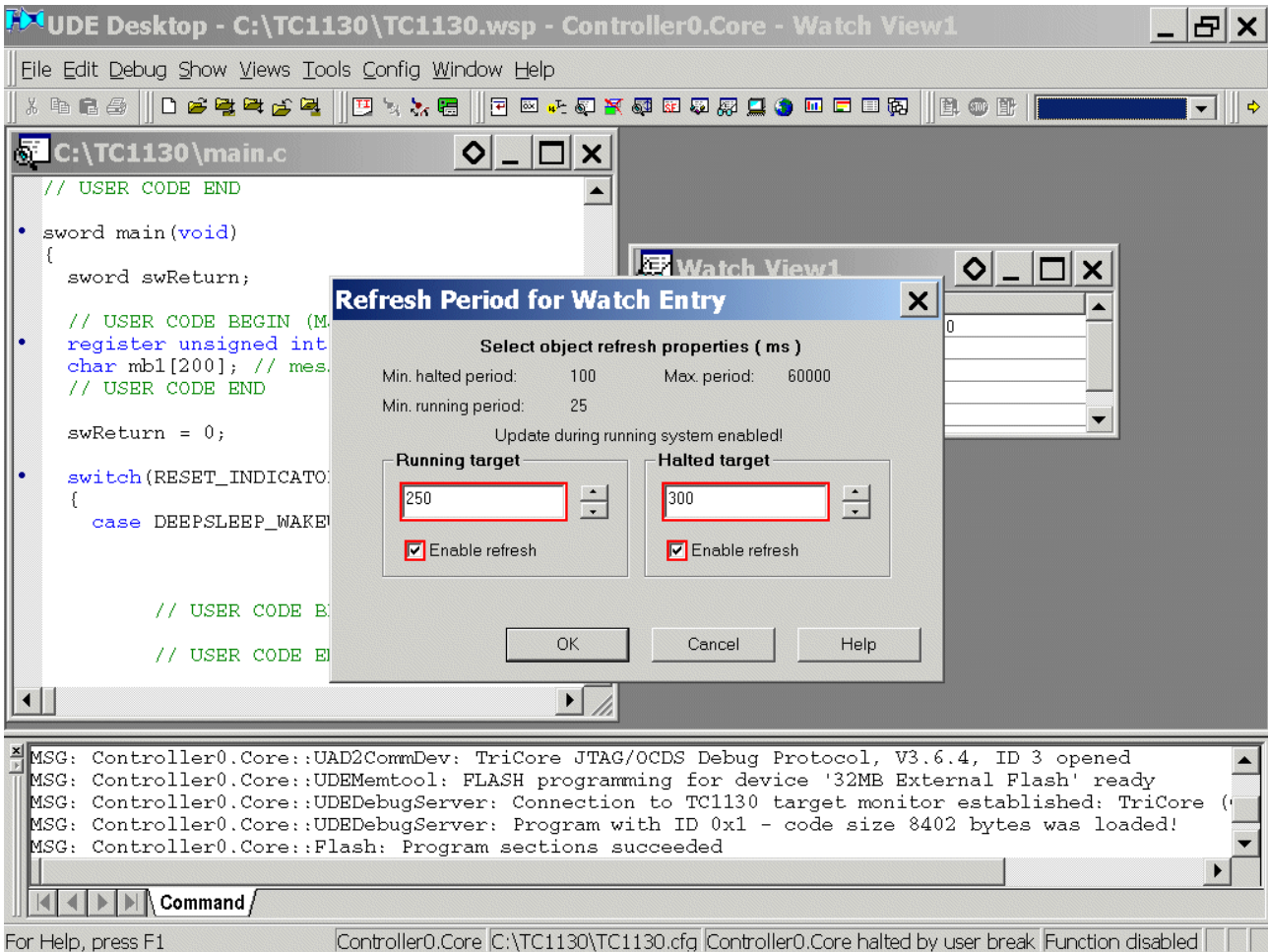
Name	Value
STM_Clock	0x24004000
day	
hour	
minute	
second	

A context menu is open over the `STM_Clock` entry, with the `Refresh Period` option selected. The command window at the bottom shows the following messages:

```
MSG: Controller0.Core::UAD2CommDev: TriCore JTAG/OCDS Debug Protocol, V3.6.4, ID 3 opened
MSG: Controller0.Core::UDEMentool: 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
```

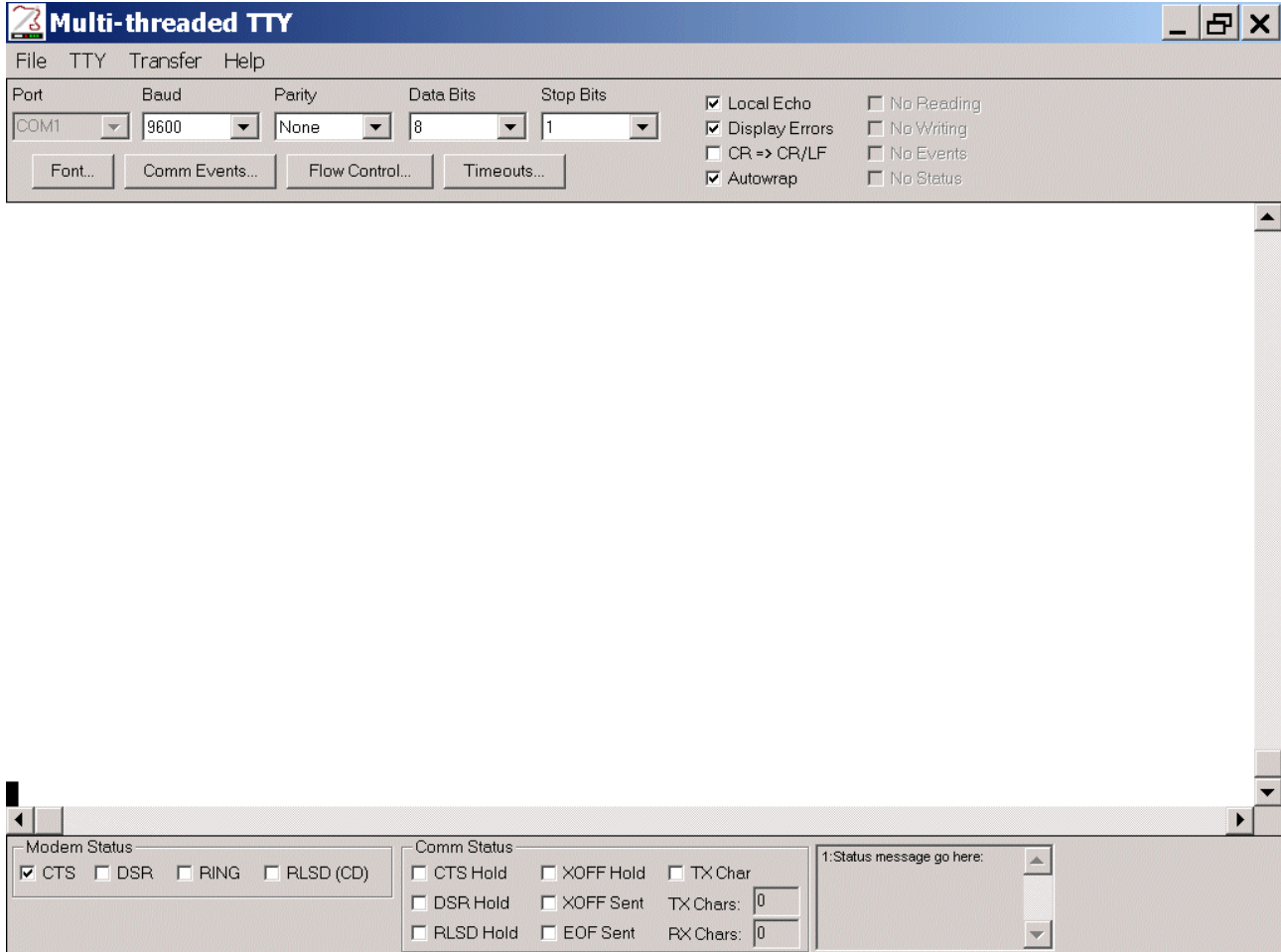
The status bar at the bottom indicates: "For Help, press F1 | Controller0.Core | C:\TC1130\TC1130.cfg | Controller0.Core halted by user break | Function disabled |"

- Running target: **click** ✓ Enable refresh
- Running target: **select** 250
- Halted target: **click** ✓ Enable refresh
- Halted target: **select** 300

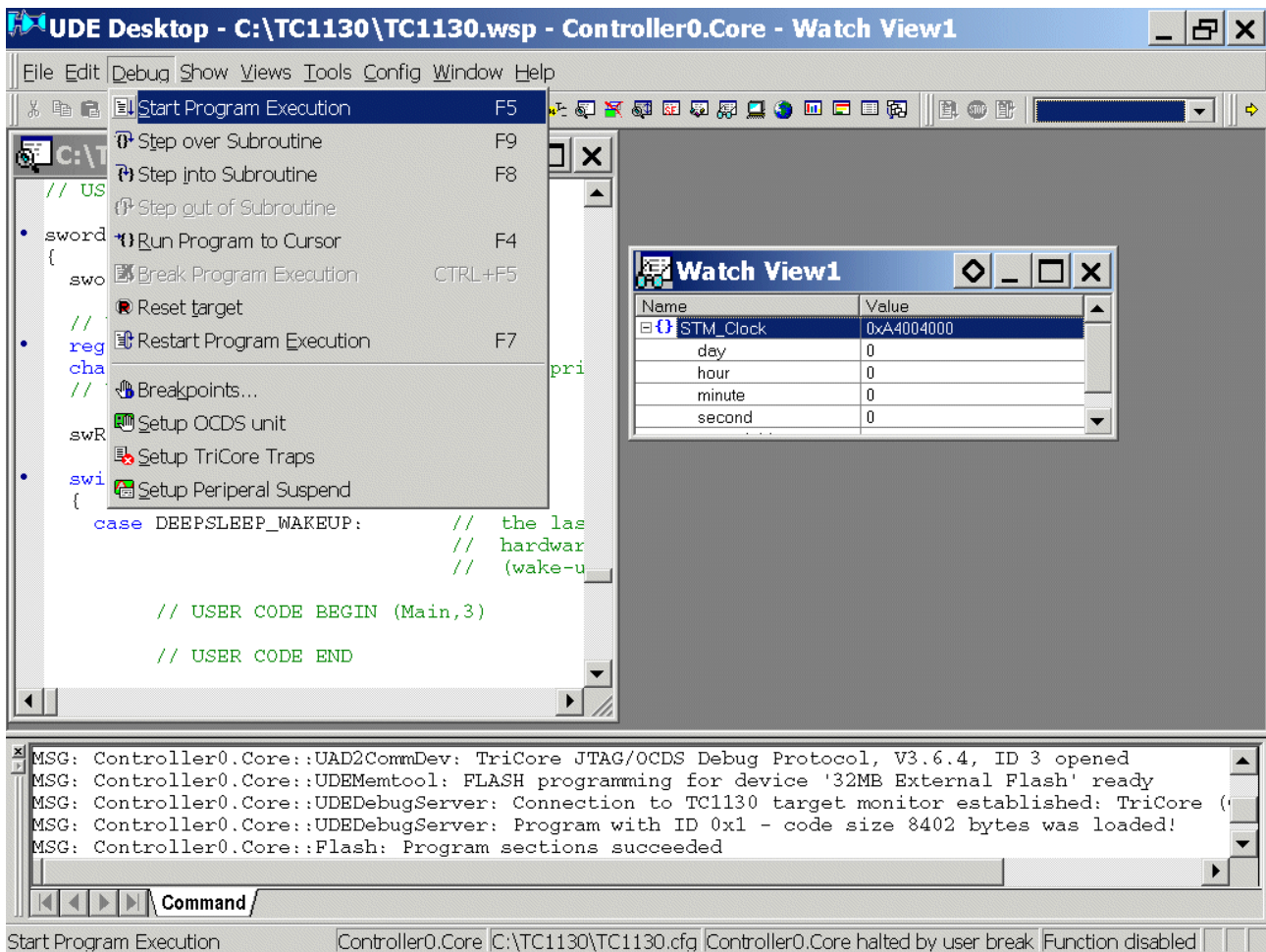


OK

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



Debug – Start Program Execution



The screenshot shows the UDE Desktop interface for the TC1130 Controller0.Core. The 'Start Program Execution' menu is open, displaying various debugging options. The 'Watch View1' window is also visible, showing the value of the STM_Clock variable.

Name	Value
STM_Clock	0xA4004000
day	0
hour	0
minute	0
second	0

The main editor window shows the following code snippet:

```

// US
sword
{
  swo
  //
  // reg
  // cha
  //
  swR
  //
  swi
  {
    case DBEPSLEEP_WAKEUP: // the las
                          // hardwar
                          // (wake-u
                          //
                          // USER CODE BEGIN (Main,3)
                          //
                          // USER CODE END
  }
}

```

The command window at the bottom displays the following messages:

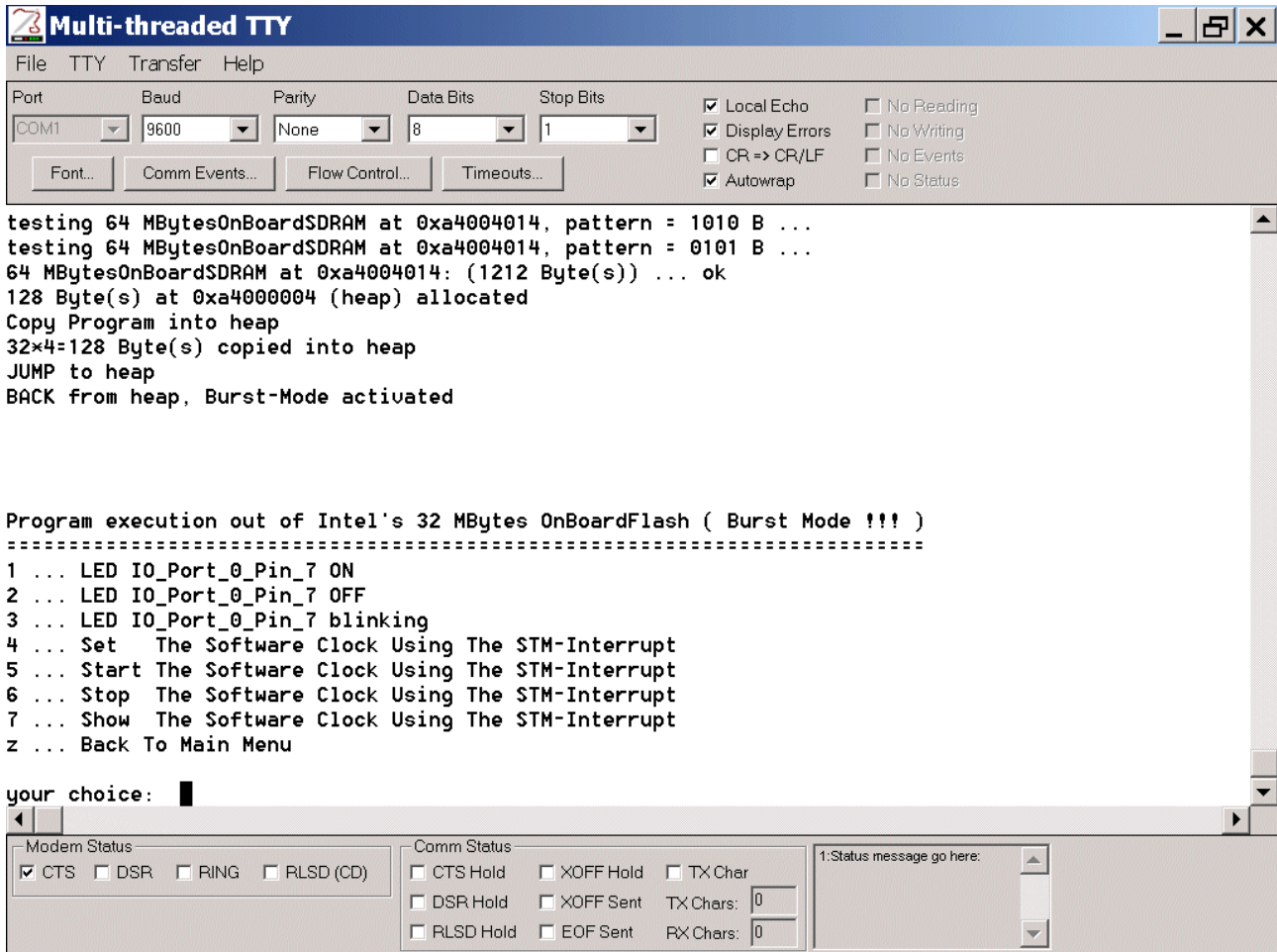
```

MSG: Controller0.Core::UAD2CommDev: TriCore JTAG/OCDS Debug Protocol, V3.6.4, ID 3 opened
MSG: Controller0.Core::UDEMentool: 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

```

The status bar at the bottom indicates: Start Program Execution | Controller0.Core | C:\TC1130\TC1130.cfg | Controller0.Core halted by user break | Function disabled

And see the result:



Multi-threaded TTY

File TTY Transfer Help

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

Local Echo No Reading
 Display Errors No Writing
 CR => CR/LF No Events
 Autowrap No Status

Font... Comm Events... Flow Control... Timeouts...

```

testing 64 MBytesOnBoardSDRAM at 0xa4004014, pattern = 1010 B ...
testing 64 MBytesOnBoardSDRAM at 0xa4004014, pattern = 0101 B ...
64 MBytesOnBoardSDRAM at 0xa4004014: (1212 Byte(s)) ... ok
128 Byte(s) at 0xa4000004 (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
z ... Back To Main Menu

your choice: █
  
```

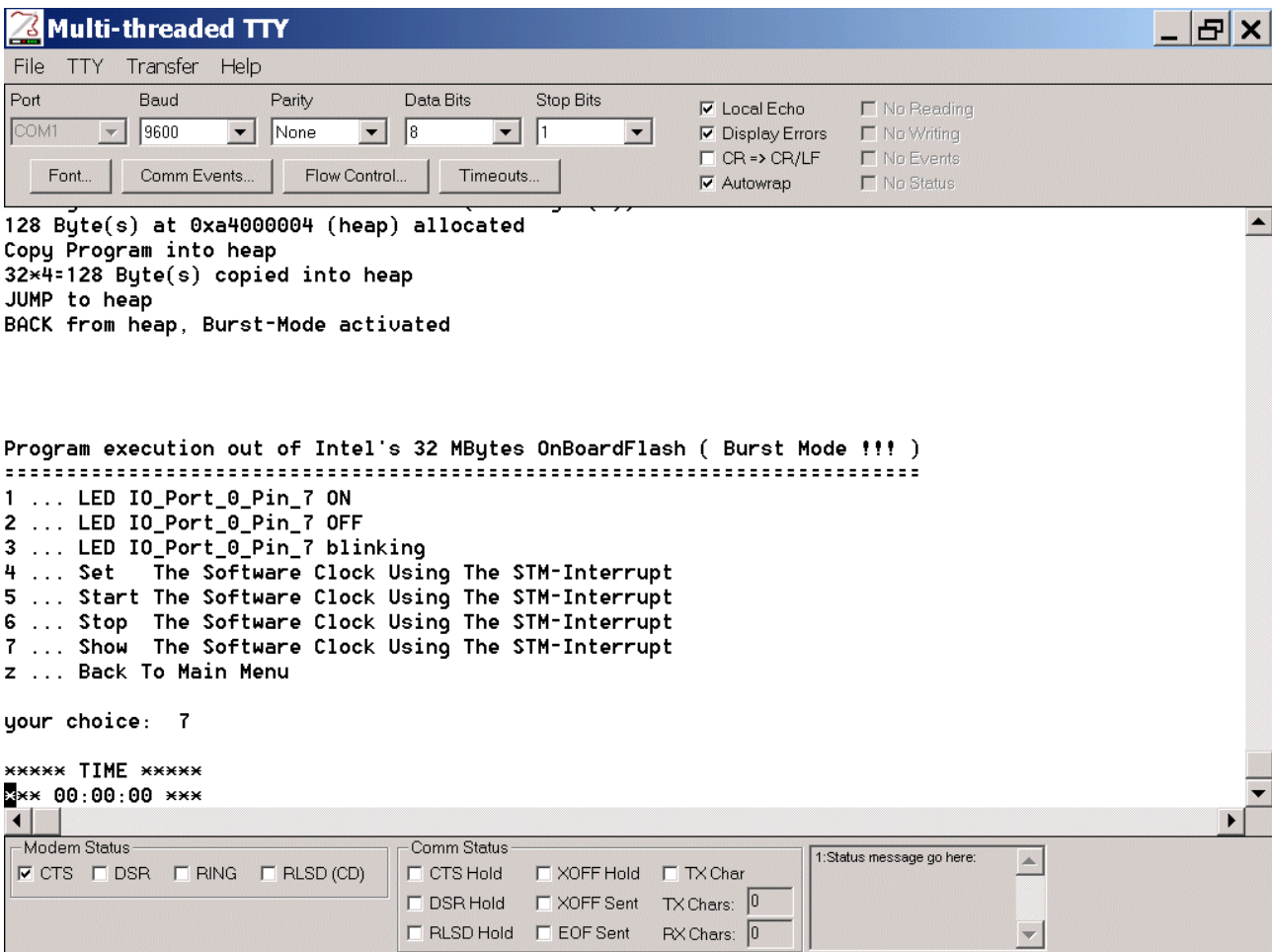
Modem Status: CTS DSR RING RLSD (CD)

Comm Status: CTS Hold XOFF Hold TX Char
 DSR Hold XOFF Sent TX Chars: 0
 RLSD Hold EOF Sent RX Chars: 0

1: Status message go here:



Insert 7



The screenshot shows a terminal window titled "Multi-threaded TTY" with a menu bar (File, TTY, Transfer, Help) and a configuration area. The configuration includes: Port (COM1), Baud (9600), Parity (None), Data Bits (8), Stop Bits (1), and checkboxes for Local Echo, Display Errors, CR => CR/LF, Autowrap, No Reading, No Writing, No Events, and No Status. The main text area contains the following output:

```

128 Byte(s) at 0xa4000004 (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 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

**** TIME ****
*** 00:00:00 ***

```

At the bottom, there are sections for Modem Status (CTS, DSR, RING, RLSD (CD)), Comm Status (CTS Hold, XOFF Hold, TX Char, DSR Hold, XOFF Sent, TX Chars: 0, RLSD Hold, EOF Sent, RX Chars: 0), and a status message field.

Insert z



Insert 4

Multi-threaded TTY

File TTY Transfer Help

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

Local Echo No Reading
 Display Errors No Writing
 CR => CR/LF No Events
 Autowrap No Status

Font... Comm Events... Flow Control... Timeouts...

```

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 ****

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

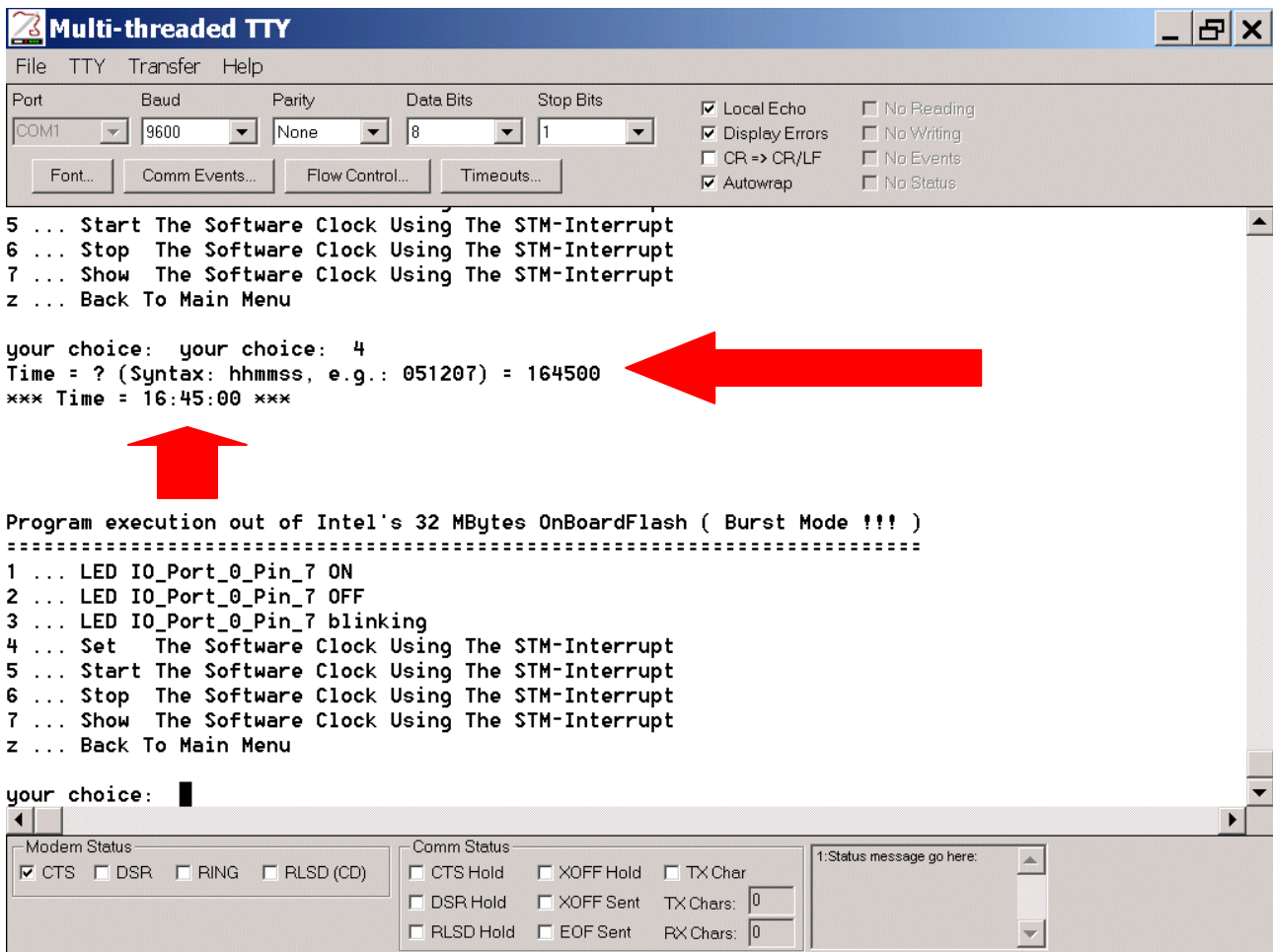
your choice: your choice: 4
Time = ? (Syntax: hhmmss, e.g.: 051207) = █
  
```

Modem Status: CTS DSR RING RLSD (CD)
 Comm Status: CTS Hold XOFF Hold TX Char
 DSR Hold XOFF Sent TX Chars: 0
 RLSD Hold EOF Sent RX Chars: 0

1: Status message go here:



Insert 164500



Multi-threaded TTY

File TTY Transfer Help

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

Font... Comm Events... Flow Control... Timeouts...

Local Echo No Reading
 Display Errors No Writing
 CR => CR/LF No Events
 Autowrap No Status

```

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: hhmmss, 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: █

```

Modem Status: CTS DSR RING RLSD (CD)

Comm Status: CTS Hold XOFF Hold TX Char
 DSR Hold XOFF Sent TX Chars: 0
 RLSD Hold EOF Sent FX Chars: 0

1: Status message go here:



@ 16:45:00 insert 5

```

Multi-threaded TTY
File TTY Transfer Help
Port COM1 Baud 9600 Parity None Data Bits 8 Stop Bits 1
Local Echo checked, Display Errors checked, CR => CR/LF unchecked, Autowrap checked, No Reading unchecked, No Writing unchecked, No Events unchecked, No Status unchecked
Font... Comm Events... Flow Control... Timeouts...

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: █

Modem Status: CTS checked, DSR unchecked, RING unchecked, RLSD (CD) unchecked
Comm Status: CTS Hold unchecked, DSR Hold unchecked, RLSD Hold unchecked, XOFF Hold unchecked, XOFF Sent unchecked, EOF Sent unchecked, TX Char unchecked, TX Chars: 0, RX Chars: 0
1:Status message go here:
  
```



Insert 7

Multi-threaded TTY

File TTY Transfer Help

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

Local Echo No Reading
 Display Errors No Writing
 CR => CR/LF No Events
 Autowrap No Status

Font... Comm Events... Flow Control... Timeouts...

```

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

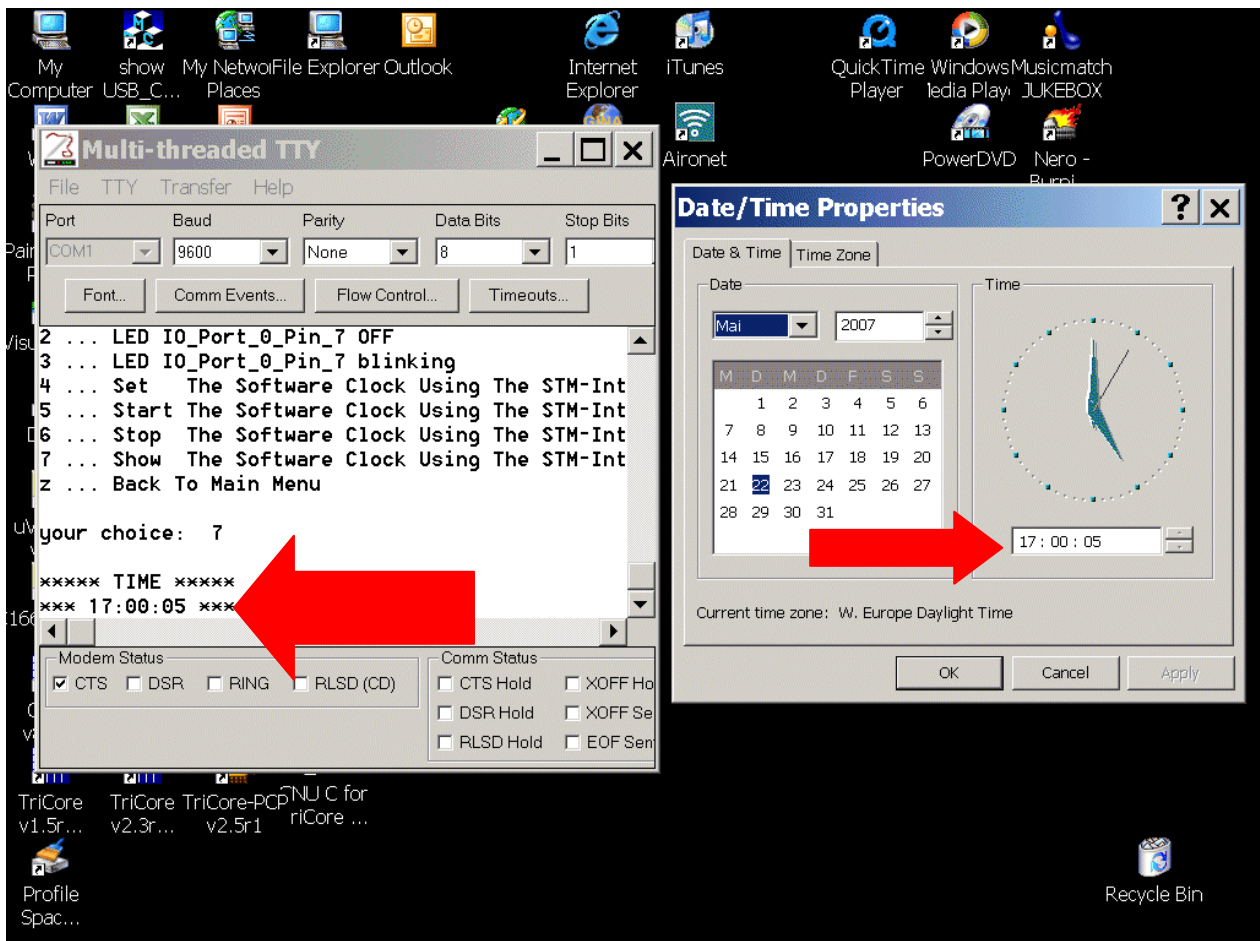
**** TIME ****
*** 16:46:17 ***
  
```

Modem Status: CTS DSR RING RLSD (CD)

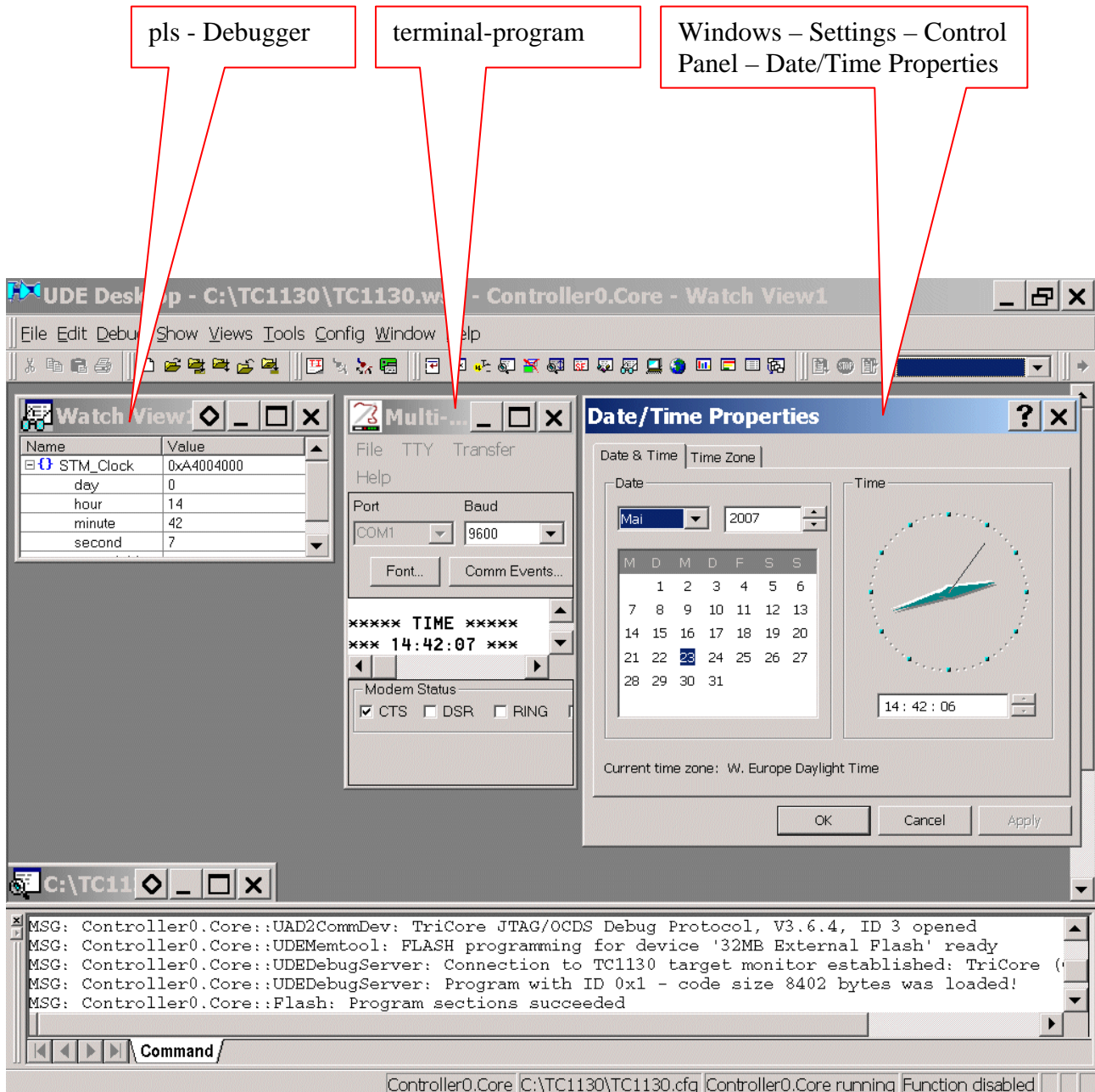
Comm Status: CTS Hold XOFF Hold TX Char
 DSR Hold XOFF Sent TX Chars: 0
 RLSD Hold EOF Sent RX Chars: 0

1:Status message go here:

Compare with your "computer time":



After 1 day (Compare with your "computer time" and with your debugger):



The screenshot shows the UDE Desktop interface with three windows open:

- Watch View:** A table showing the value of the STM_Clock register over time.

Name	Value
STM_Clock	0xA4004000
day	0
hour	14
minute	42
second	7
- Multi-...:** A terminal window displaying the time:


```

      ***** TIME *****
      *** 14:42:07 ***
      
```
- Date/Time Properties:** A dialog box showing the system date and time. The date is May 23, 2007, and the time is 14:42:06. The current time zone is W. Europe Daylight Time.

Red callout boxes indicate the following paths:

- pls - Debugger:** Points to the Watch View window.
- terminal-program:** Points to the Multi-... terminal window.
- Windows – Settings – Control Panel – Date/Time Properties:** Points to the Date/Time Properties dialog box.

The bottom of the screenshot shows a command window with the following messages:

```

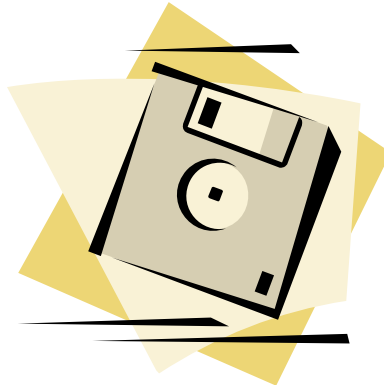
MSG: Controller0.Core::UAD2CommDev: TriCore JTAG/OCDS Debug Protocol, V3.6.4, ID 3 opened
MSG: Controller0.Core::UDEMentool: FLASH programming for device '32MB External Flash' ready
MSG: Controller0.Core::UDEDebugServer: Connection to TC1130 target monitor established: TriCore (
MSG: Controller0.Core::UDEDebugServer: Program with ID 0x1 - code size 8402 bytes was loaded!
MSG: Controller0.Core::Flash: Program sections succeeded
  
```

Close the pls-Debugger:

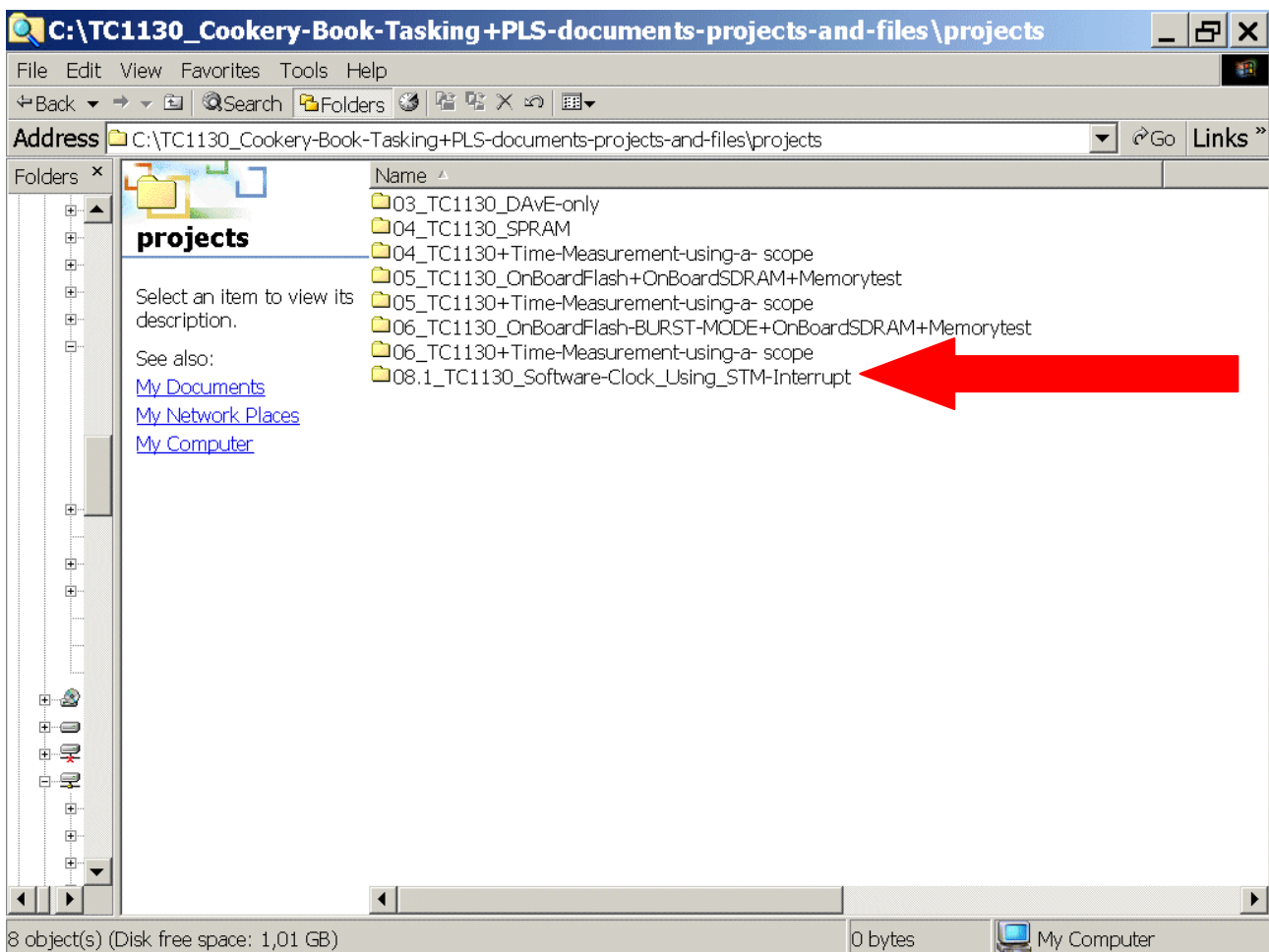
File – Close Workspace

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

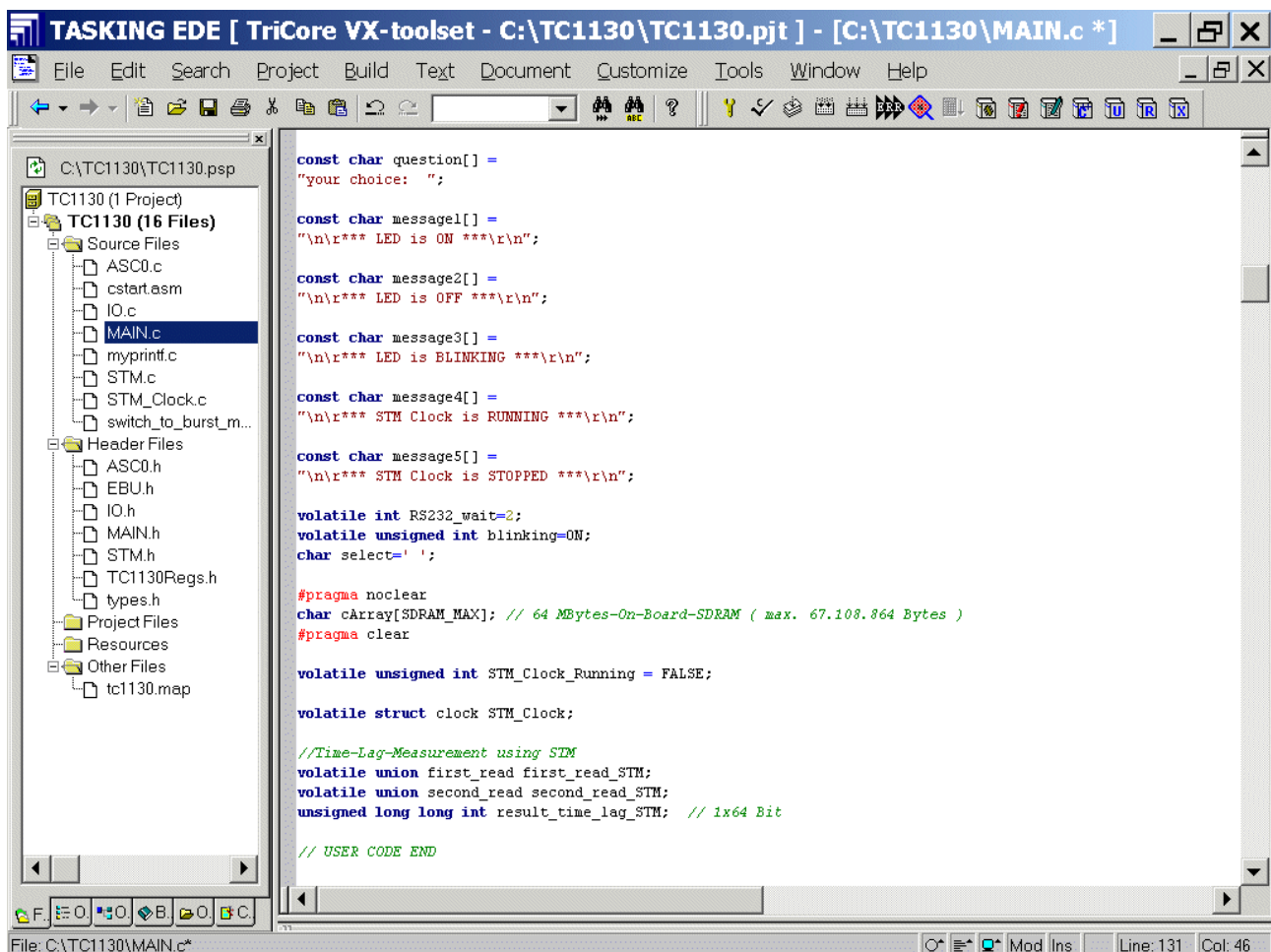
Look in: **select** C:\TC1130

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
```



The screenshot shows the TASKING EDE IDE interface. The left pane displays the project structure for 'TC1130 (1 Project)' with 16 files. The main editor window shows the source code for 'MAIN.c'. The code includes several character arrays for messages, integer variables for wait times and blinking, and a union for time-lag measurement. The user code section is highlighted in green.

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

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

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

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

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

const char message5[] =
"\n\r*** STM Clock is STOPPED ***\r\n";

volatile int RS232_wait=2;
volatile unsigned int blinking=0N;
char select=' ';

#pragma noclear
char cArray[SDRAM_MAX]; // 64 MBytes-On-Board-SDRAM ( max. 67.108.864 Bytes )
#pragma clear

volatile unsigned int STM_Clock_Running = FALSE;

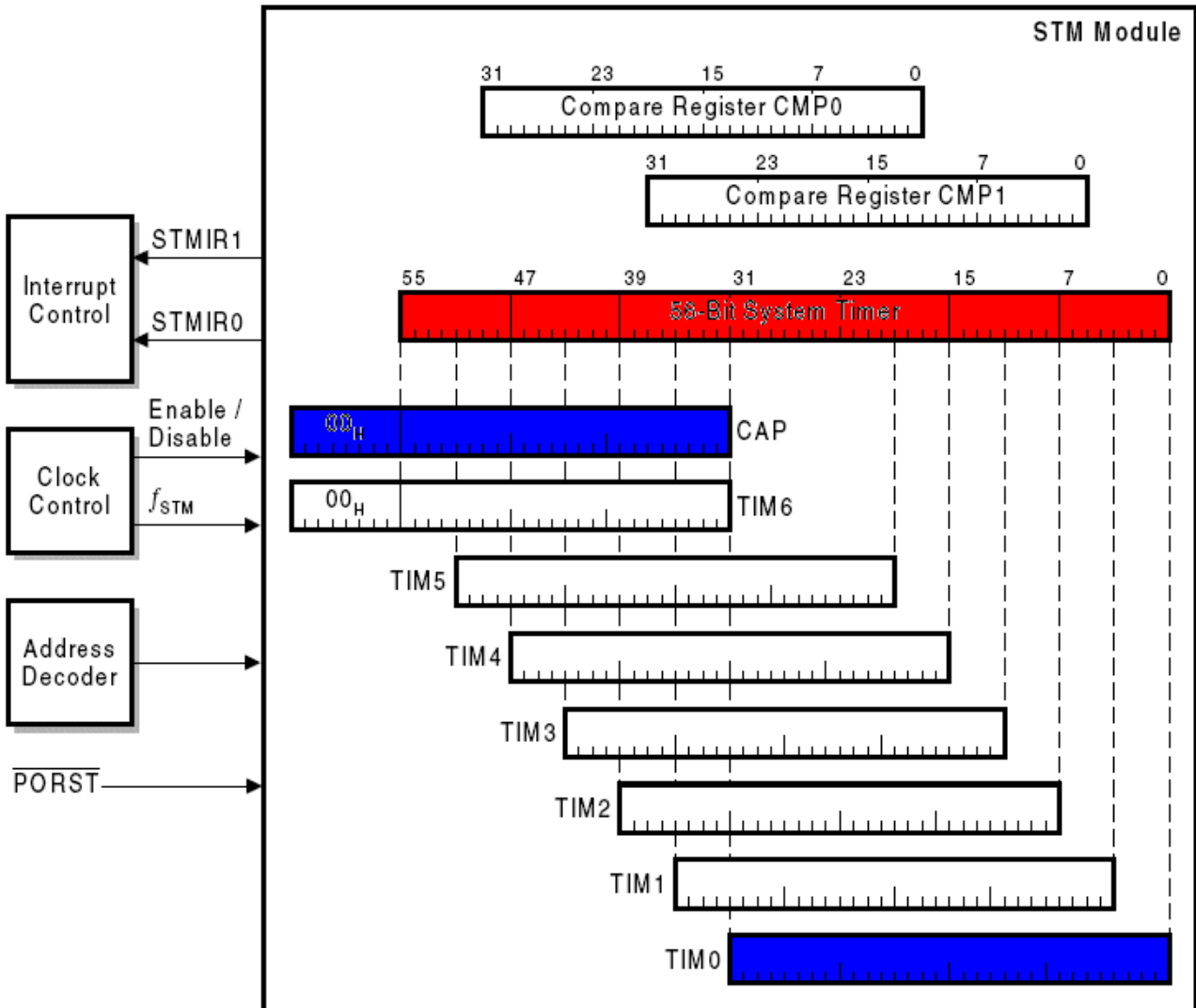
volatile struct clock STM_Clock;

//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

// USER CODE END
```



Additional Information: System Timer (STM):



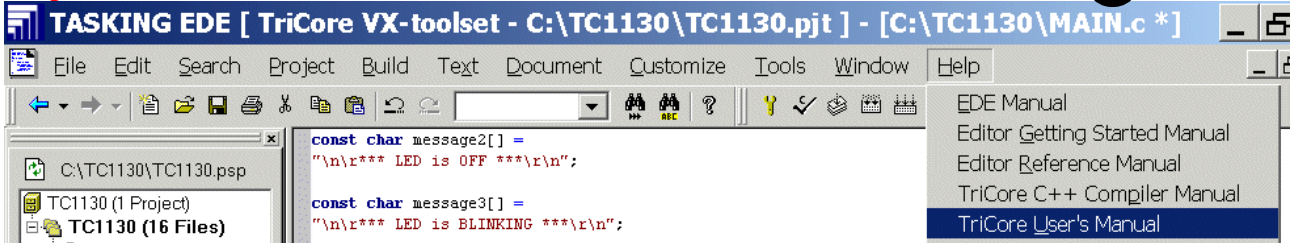
Note:

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.



Additional Information: fundamental data types:

Help – TriCore User’s Manual

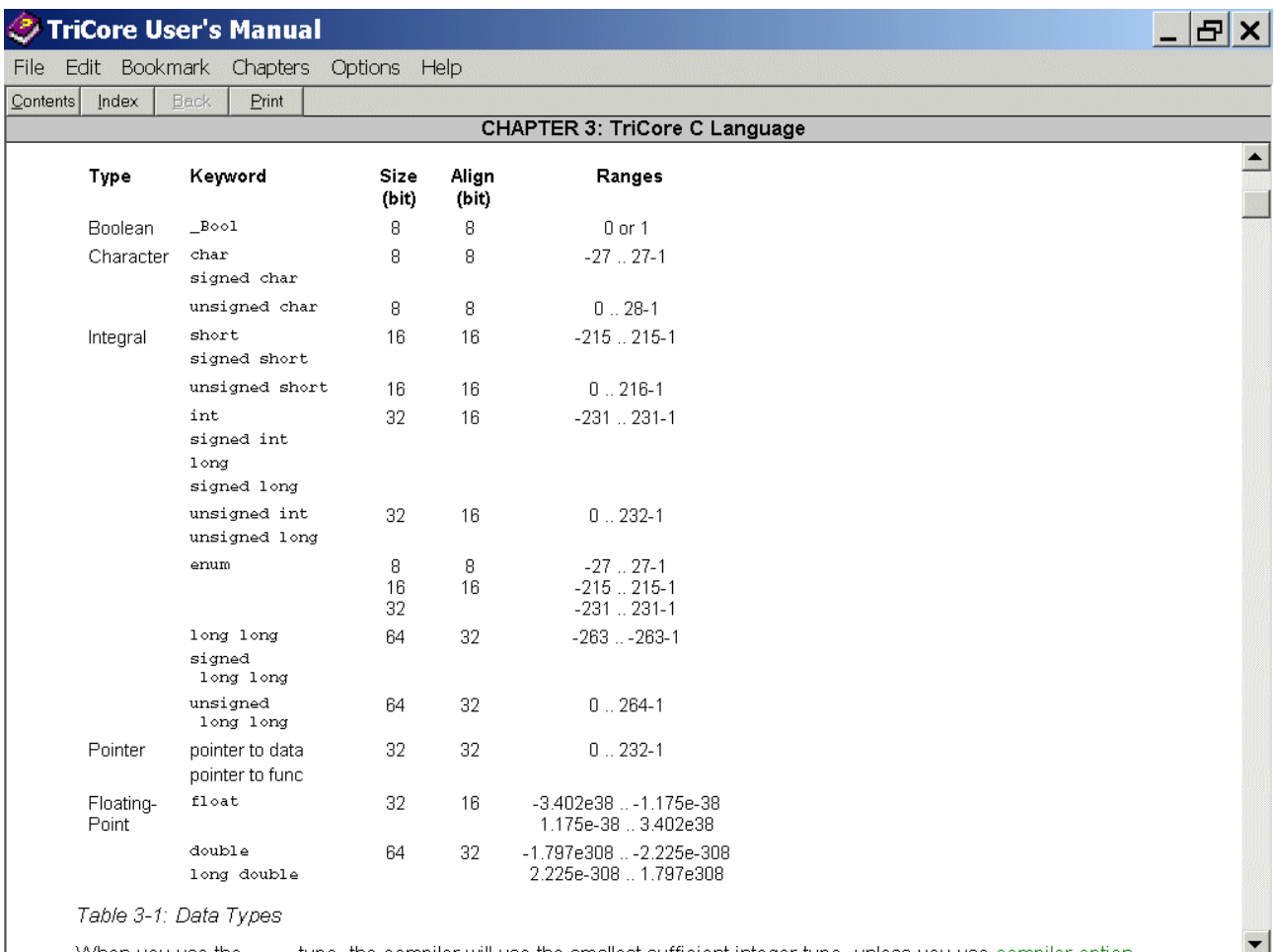


Note:

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:



Type	Keyword	Size (bit)	Align (bit)	Ranges	
Boolean	_Bool	8	8	0 or 1	
Character	char	8	8	-27 .. 27-1	
	signed char				
	unsigned char	8	8	0 .. 28-1	
Integral	short	16	16	-215 .. 215-1	
	signed short				
	unsigned short	16	16	0 .. 216-1	
	int	32	16	-231 .. 231-1	
	signed int				
	long				
	signed long				
	unsigned int	32	16	0 .. 232-1	
	unsigned long				
	enum		8	8	-27 .. 27-1
Pointer	pointer to data	16	16	-215 .. 215-1	
	pointer to func	32	16	-231 .. 231-1	
	long long	64	32	-263 .. -263-1	
	signed long long				
	unsigned long long	64	32	0 .. 264-1	
	pointer to data	32	32	0 .. 232-1	
	pointer to func				
	Floating-Point	float	32	16	-3.402e38 .. -1.175e-38 1.175e-38 .. 3.402e38
	double	64	32	-1.797e308 .. -2.225e-308 2.225e-308 .. 1.797e308	
	long double				

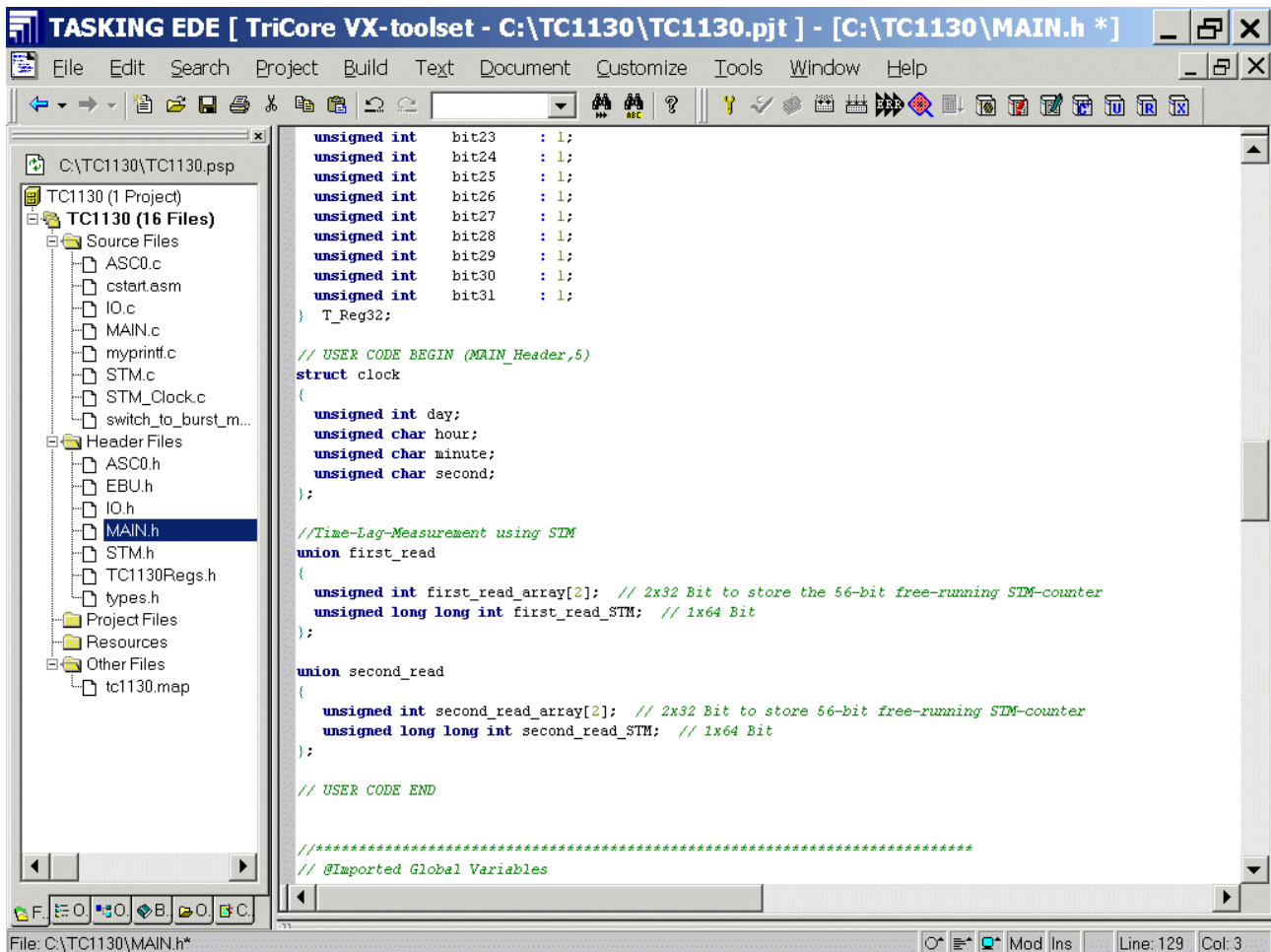
Table 3-1: Data Types

When you use the enum type, the compiler will use the smallest sufficient integer type, unless you use compiler option --

Double click: [Main.h](#) and insert Typedefs:

```
//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
};
```



```

unsigned int    bit23    : 1;
unsigned int    bit24    : 1;
unsigned int    bit25    : 1;
unsigned int    bit26    : 1;
unsigned int    bit27    : 1;
unsigned int    bit28    : 1;
unsigned int    bit29    : 1;
unsigned int    bit30    : 1;
unsigned int    bit31    : 1;
} T_Reg32;

// USER CODE BEGIN (MAIN_Header,5)
struct clock
{
    unsigned int day;
    unsigned char hour;
    unsigned char minute;
    unsigned char second;
};

//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
};

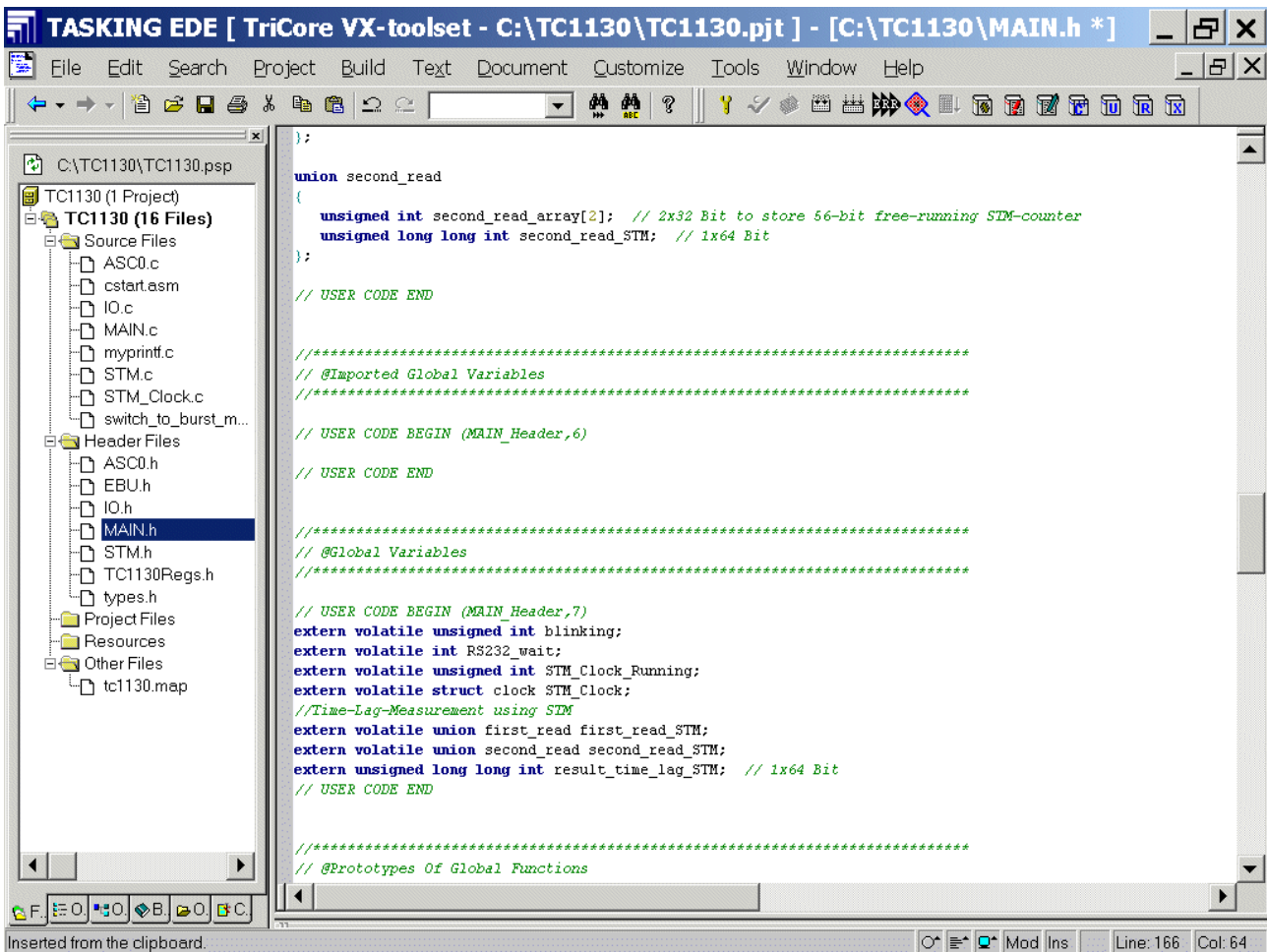
// USER CODE END

//*****
// @Imported Global Variables

```

Double click: [Main.h](#) and insert Prototypes of Global Variables (Extern Declaration):

```
//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
```



```
};
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
};

// USER CODE END

//*****
// @Imported Global Variables
//*****

// USER CODE BEGIN (MAIN_Header,6)

// USER CODE END

//*****
// @Global Variables
//*****

// USER CODE BEGIN (MAIN_Header,7)
extern volatile unsigned int blinking;
extern volatile int RS232_wait;
extern volatile unsigned int STM_Clock_Running;
extern volatile struct clock STM_Clock;
//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
// USER CODE END

//*****
// @Prototypes Of Global Functions
```




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:

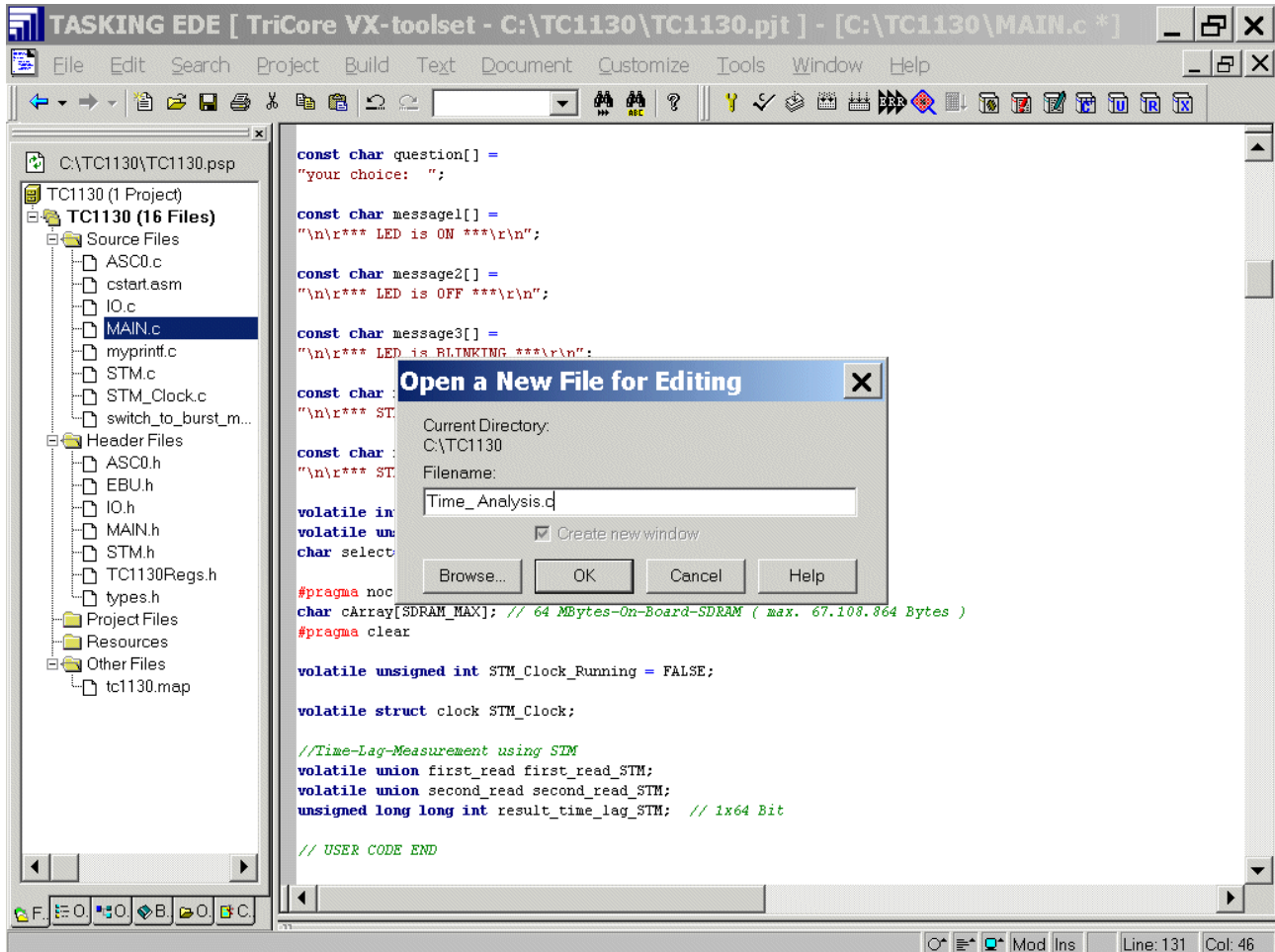
The screenshot shows the DAVE software interface with the 'System Timer (STM)' configuration window open. The window has tabs for 'Module Clock', 'Resolutions', 'Control', 'Interrupts', 'Functions', 'Parameters', and 'Notes'. The 'Resolutions' tab is selected.

The configuration is divided into two sections:

- 56-Bit System Timer:**
 - 56-bit system timer (TIM0, 6): resolution [μs] is 0,013; range [years] is 30,466.
- Additional Parts of the System Timer:**
 - System timer 0 (TIM0): resolution [μs] is 0,013; range [s] is 57,266.
 - System timer 1 (TIM1): resolution [μs] is 0,213; range [min] is 15,271.
 - System timer 2 (TIM2): resolution [μs] is 3,413; range [h] is 4,072.
 - System timer 3 (TIM3): resolution [μs] is 54,613; range [days] is 2,715.
 - System timer 4 (TIM4): resolution [μs] is 873,813; range [days] is 43,437.
 - System timer 5 (TIM5): resolution [ms] is 13,981; range [years] is 1,904.
 - System timer 6 (TIM6): resolution [s] is 57,266; range [years] is 30,466.

Two red arrows point to the 'range [years]' field for the 56-bit timer and the 'range [years]' field for System timer 6 (TIM6), both showing a value of 30,466.

File – New
Insert Time_Analysis.c



OK

Insert User Code:

```
#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;

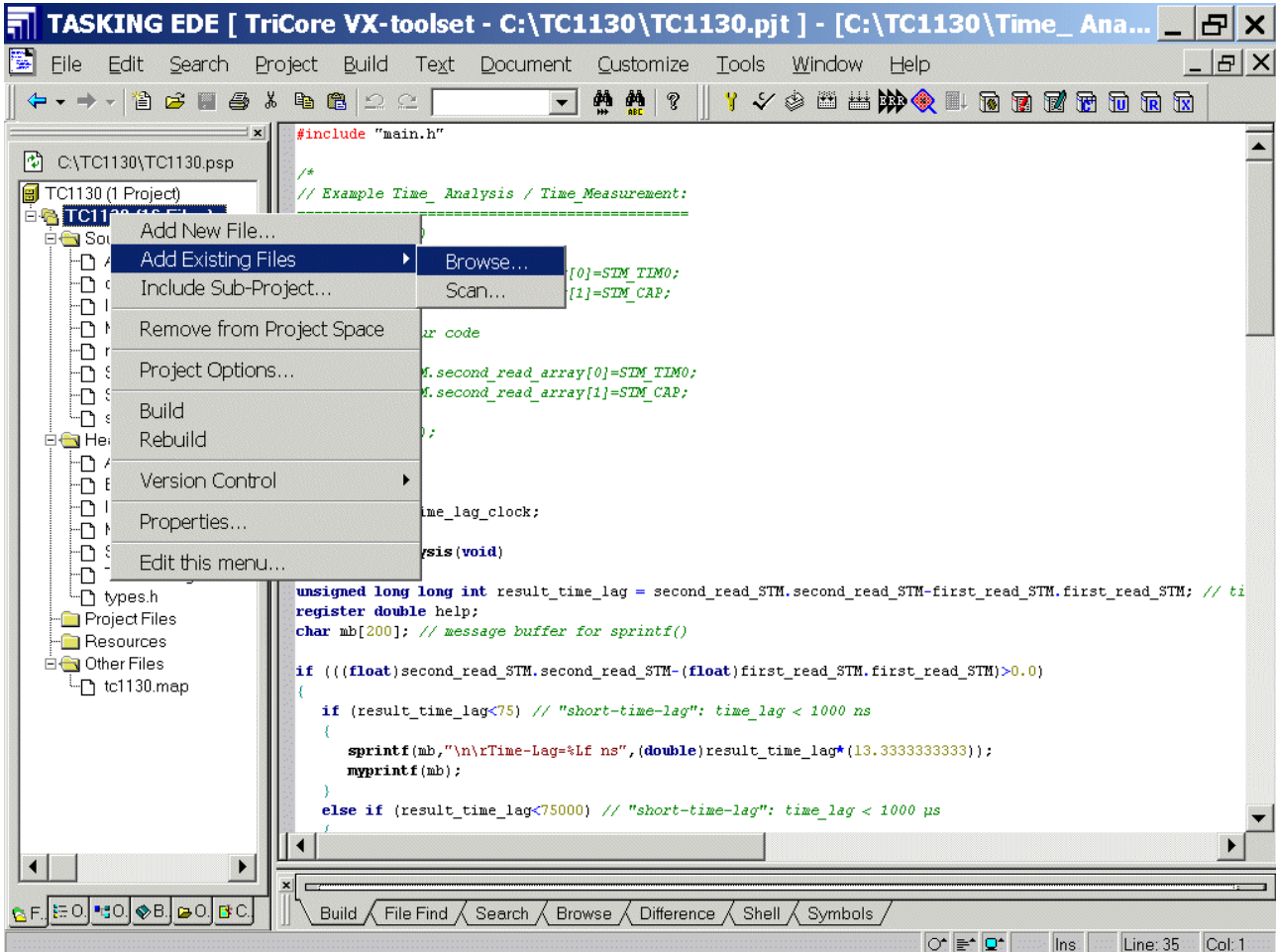
void Time_Analysis(void)
{
unsigned long long int result_time_lag = second_read_STM.second_read_STM-
first_read_STM.first_read_STM; // time-lag in STM-Timer-Ticks
register double help;
char mb[200]; // message buffer for sprintf()

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*(13.3333333333));
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*(13.3333333333*0.001));
myprintf(mb);
}
else if (result_time_lag<75000000) // "short-time-lag": time_lag < 1000 ms
{
sprintf(mb,"\n\rTime-Lag=%Lf
ms",(double)result_time_lag*(13.3333333333*0.001*0.001));
myprintf(mb);
}
else // "long-time-lag": time_lag >= 1000 ms
```

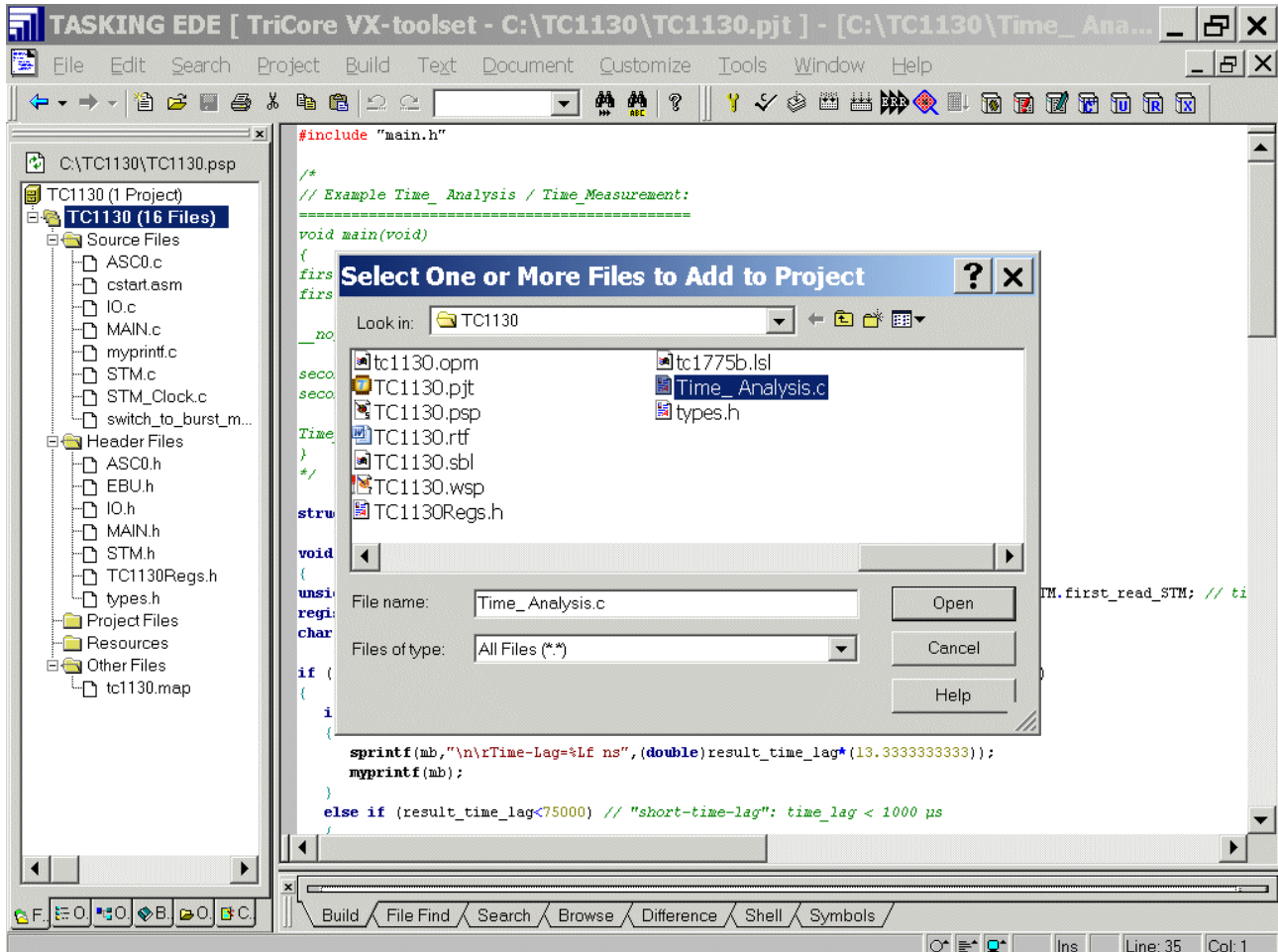
```
{
time_lag_clock.day=time_lag_clock.hour=time_lag_clock.minute=time_lag_clock.second=0;
help=(double)result_time_lag*(13.3333333333*0.001*0.001*0.001);
do
{
++time_lag_clock.second,help=help-1.0;
if (time_lag_clock.second==60)
{
time_lag_clock.second=0;
if (++time_lag_clock.minute==60)
{
time_lag_clock.minute=0;
if (++time_lag_clock.hour==24)
time_lag_clock.hour=0, ++time_lag_clock.day;
}
}
}while (help>0.0);
myprintf("\r\n\n*** TIME-LAG ***\r\n");
sprintf(mb,"*** %02u:%02u:%02u
***\r",time_lag_clock.hour,time_lag_clock.minute,time_lag_clock.second);
myprintf(mb);
}
}
}
```

File
Save All

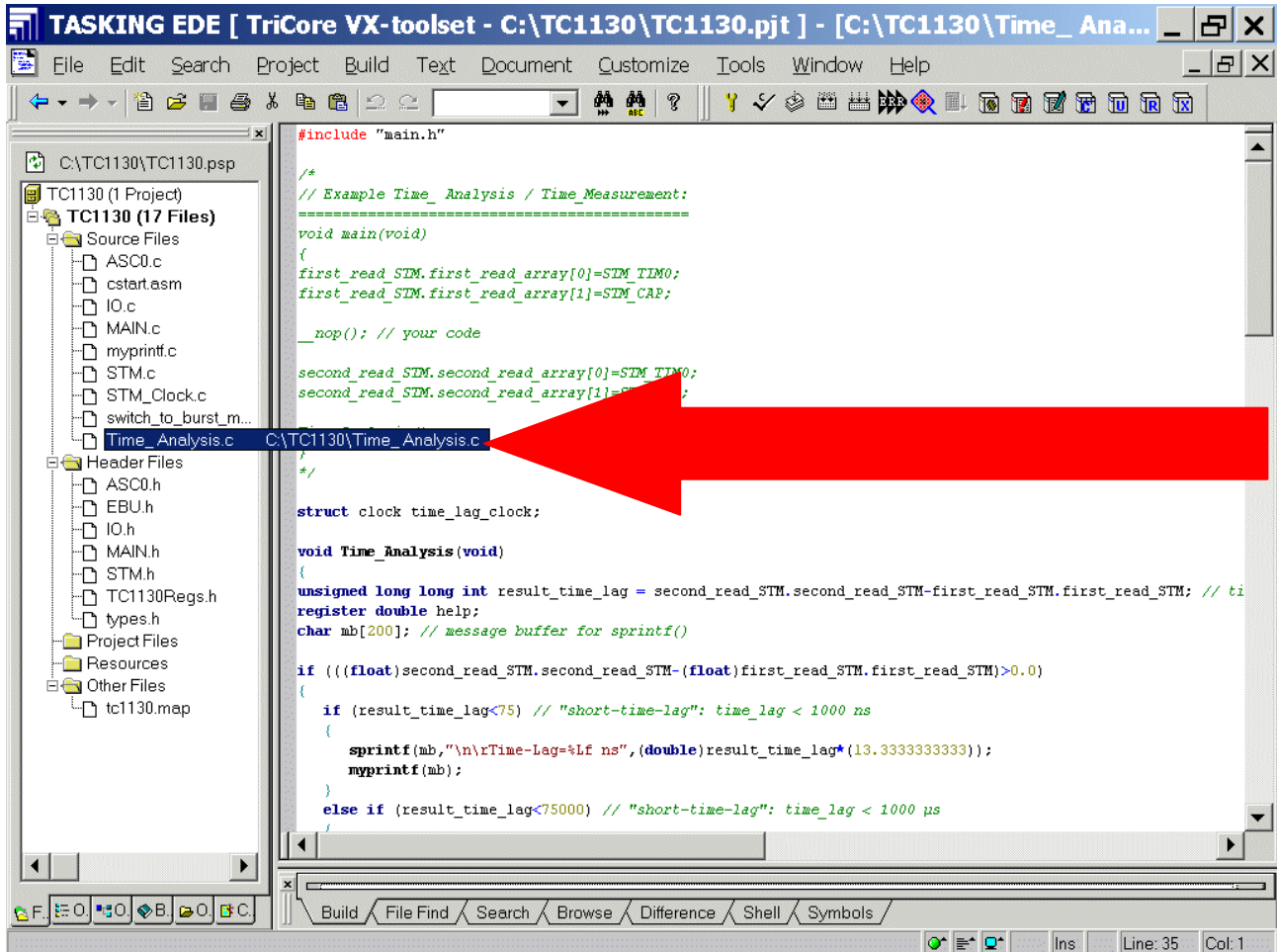
(Project Window **File View**) – TC1130 (Files) – **right mouse button click** – Add Existing Files – Browse



Select Time_Analysis.c



Open - OK



TASKING EDE [TriCore VX-toolset - C:\TC1130\TC1130.pjt] - [C:\TC1130\Time_Ana...

File Edit Search Project Build Text Document Customize Tools Window Help

C:\TC1130\TC1130.psp

- TC1130 (1 Project)
 - TC1130 (17 Files)
 - Source Files
 - ASC0.c
 - cstart.asm
 - IO.c
 - MAIN.c
 - myprintf.c
 - STM.c
 - STM_Clock.c
 - switch_to_burst_m...
 - Time_Analysis.c**
 - Header Files
 - ASC0.h
 - EBU.h
 - IO.h
 - MAIN.h
 - STM.h
 - TC1130Regs.h
 - types.h
 - Project Files
 - Resources
 - Other Files
 - tc1130.map

```

#include "main.h"

/*
// Example Time_Analysis / Time_Measurement:
-----
void main(void)
{
  first_read_STM.first_read_array[0]=STM_TLM0;
  first_read_STM.first_read_array[1]=STM_CAP;

  __nop(); // your code

  second_read_STM.second_read_array[0]=STM_TLM0;
  second_read_STM.second_read_array[1]=STM_CAP;
}

*/

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; // ti
  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*(13.3333333333));
      myprintf(mb);
    }
    else if (result_time_lag<75000) // "short-time-lag": time_lag < 1000 μs
    {

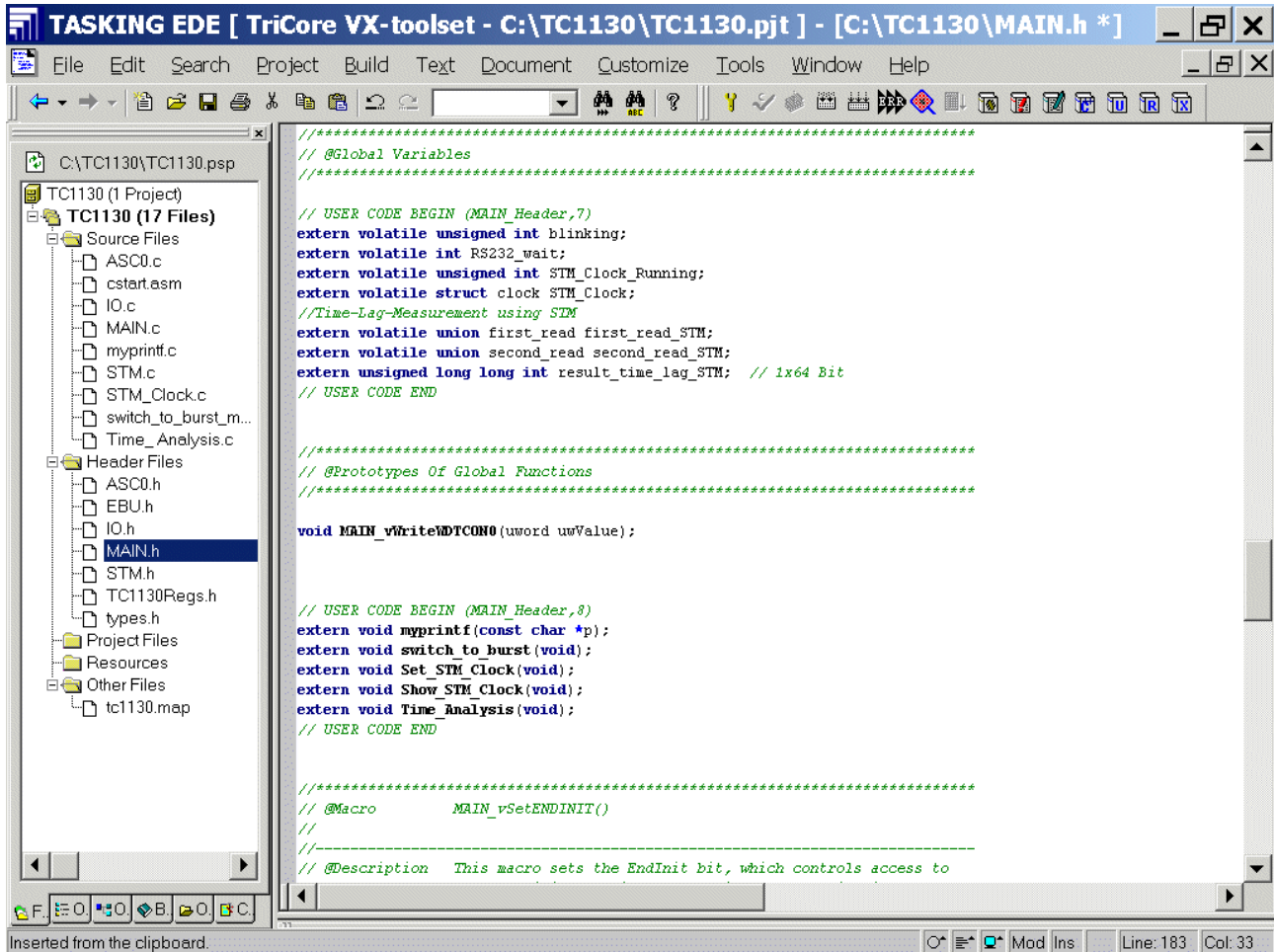
```

Build File Find Search Browse Difference Shell Symbols

Ins Line: 35 Col: 1

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

extern void Time_Analysis(void);



```

//*****
//Global Variables
//*****

// USER CODE BEGIN (MAIN_Header,7)
extern volatile unsigned int blinking;
extern volatile int RS232_wait;
extern volatile unsigned int STM_Clock_Running;
extern volatile struct clock STM_Clock;
//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
// USER CODE END

//*****
//Prototypes Of Global Functions
//*****

void MAIN_vWriteWDTCON0(uword uwValue);

// USER CODE BEGIN (MAIN_Header,8)
extern void myprintf(const char *p);
extern void switch_to_burst(void);
extern void Set_STM_Clock(void);
extern void Show_STM_Clock(void);
extern void Time_Analysis(void);
// USER CODE END

//*****
// @Macro      MAIN_vSetENDINIT()
//
//-----
// @Description This macro sets the EndInit bit, which controls access to

```

Inserted from the clipboard. Mod Ins Line: 183 Col: 33

Double click: [Main.c](#) and change Code ["void main (void)" - Function] from

```

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
}

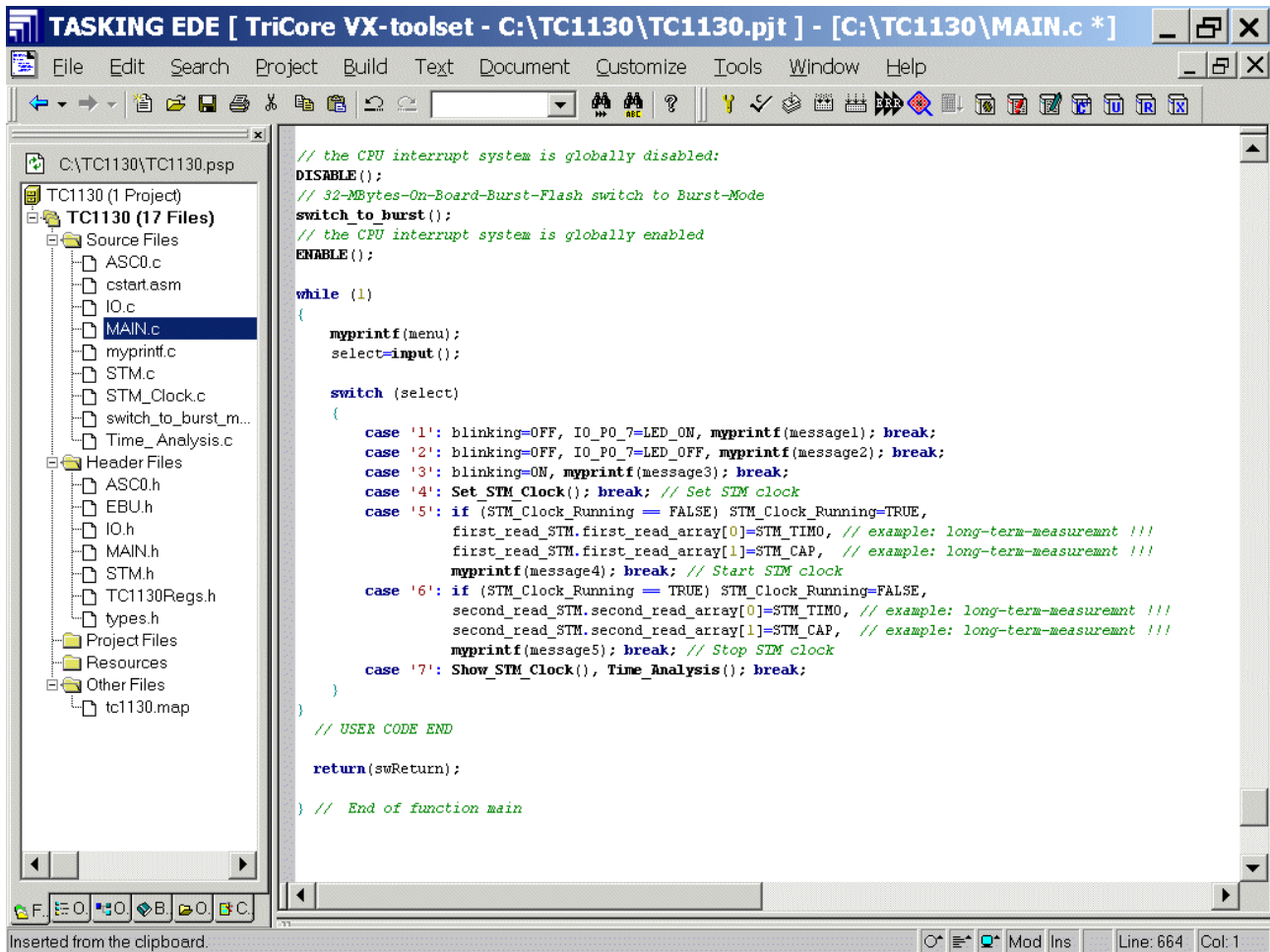
```

to

```

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,
        first_read_STM.first_read_array[0]=STM_TIM0, //
example: long-term-measurmnt !!!
        first_read_STM.first_read_array[1]=STM_CAP, //
example: long-term-measurmnt !!!
        myprintf(message4); break; // Start STM clock
    case '6': if (STM_Clock_Running == TRUE)
STM_Clock_Running=FALSE,
        second_read_STM.second_read_array[0]=STM_TIM0, // example: long-
term-measurmnt !!!
        second_read_STM.second_read_array[1]=STM_CAP, //
example: long-term-measurmnt !!!
        myprintf(message5); break; // Stop STM clock
    case '7': Show_STM_Clock(), Time_Analysis(); break;
}

```



```

// 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();

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;
        case '4': Set_STM_Clock(); break; // Set STM clock
        case '5': if (STM_Clock_Running == FALSE) STM_Clock_Running=TRUE,
            first_read_STM.first_read_array[0]=STM_TIMO, // example: long-term-measremnt !!!
            first_read_STM.first_read_array[1]=STM_CAP, // example: long-term-measremnt !!!
            myprintf(message4); break; // Start STM clock
        case '6': if (STM_Clock_Running == TRUE) STM_Clock_Running=FALSE,
            second_read_STM.second_read_array[0]=STM_TIMO, // example: long-term-measremnt !!!
            second_read_STM.second_read_array[1]=STM_CAP, // example: long-term-measremnt !!!
            myprintf(message5); break; // Stop STM clock
        case '7': Show_STM_Clock(), Time_Analysis(); break;
    }
}
// USER CODE END

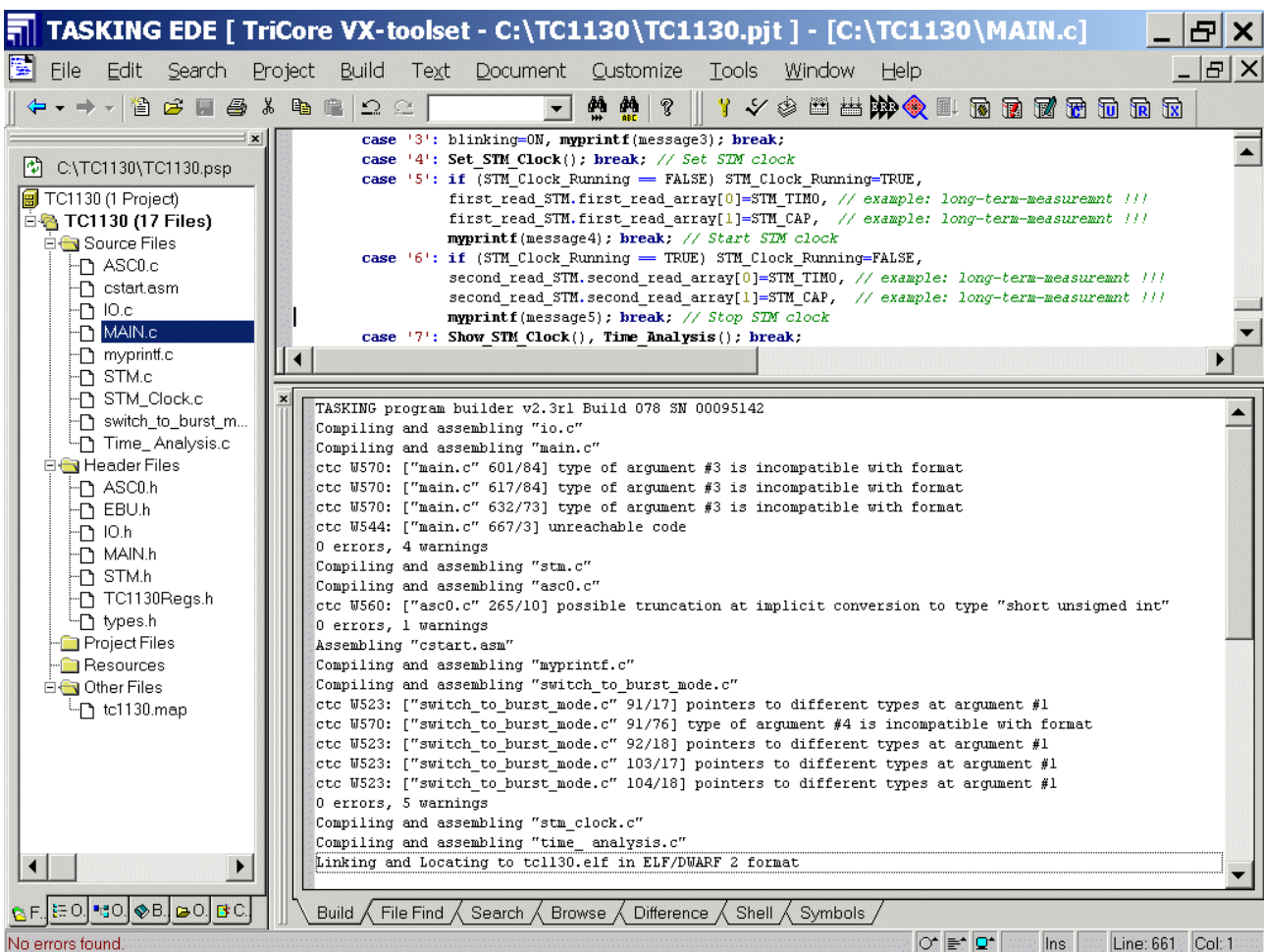
return(swReturn);
} // End of function main
    
```

Inserted from the clipboard. Mod Ins Line: 664 Col: 1

Generate your application program:

Build
Rebuild

or



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

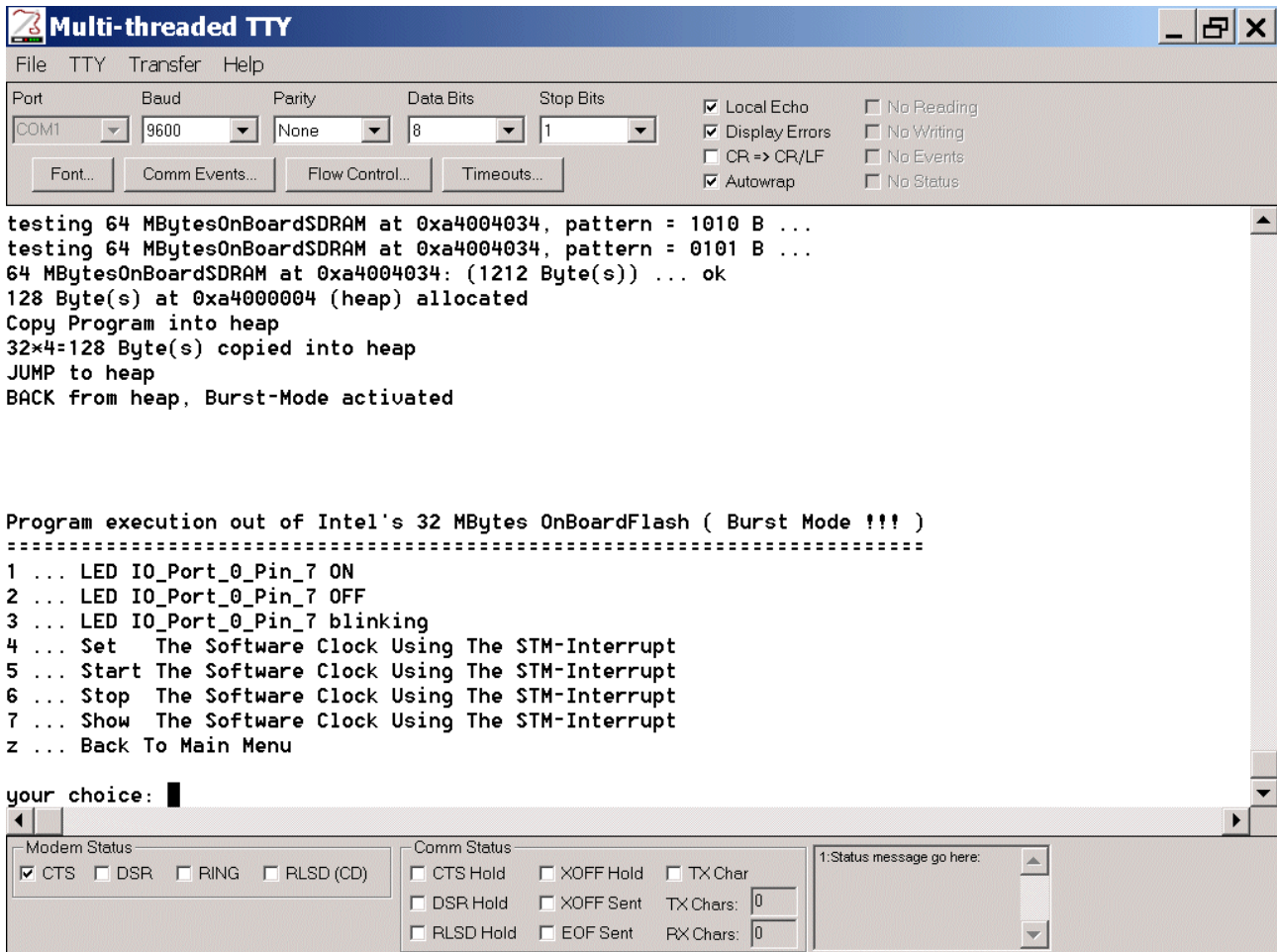
File – Close Workspace

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:



The screenshot shows a terminal window titled "Multi-threaded TTY" with a menu bar (File, TTY, Transfer, Help) and a configuration area. The configuration includes Port (COM1), Baud (9600), Parity (None), Data Bits (8), and Stop Bits (1). There are also checkboxes for Local Echo, Display Errors, CR => CR/LF, Autowrap, No Reading, No Writing, No Events, and No Status. The main text area contains the following output:

```

testing 64 MBytesOnBoardSDRAM at 0xa4004034, pattern = 1010 B ...
testing 64 MBytesOnBoardSDRAM at 0xa4004034, pattern = 0101 B ...
64 MBytesOnBoardSDRAM at 0xa4004034: (1212 Byte(s)) ... ok
128 Byte(s) at 0xa4000004 (heap) allocated
Copy Program into heap
32x4=128 Byte(s) copied into heap
JUMP to heap
BACK from heap, Burst-Mode activated

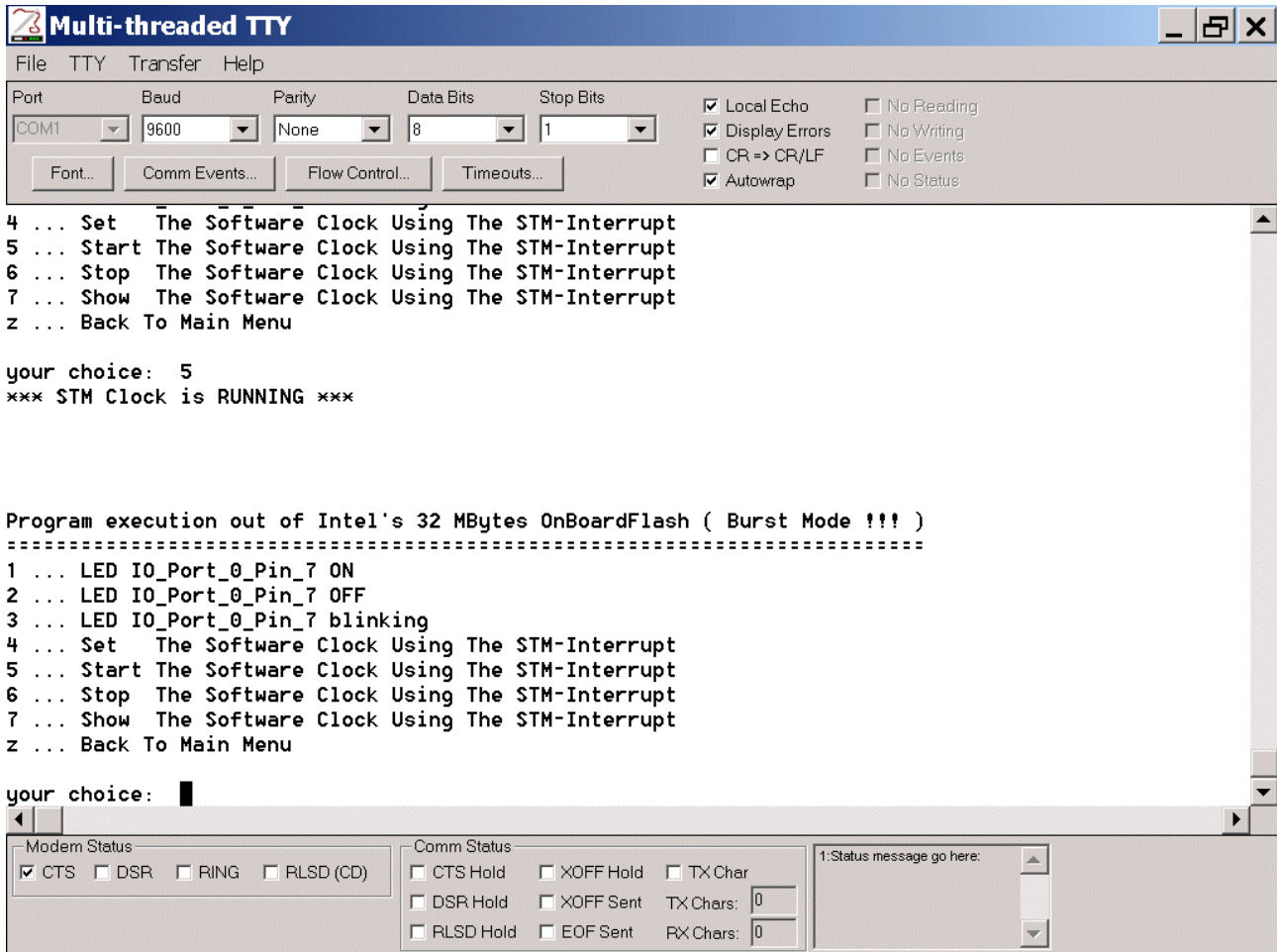
Program execution out of Intel's 32 MBytes OnBoardFlash ( Burst Mode !!! )
=====
1 ... LED IO_Port_0_Pin_7 ON
2 ... LED IO_Port_0_Pin_7 OFF
3 ... LED IO_Port_0_Pin_7 blinking
4 ... Set The Software Clock Using The STM-Interrupt
5 ... Start The Software Clock Using The STM-Interrupt
6 ... Stop The Software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

your choice: █

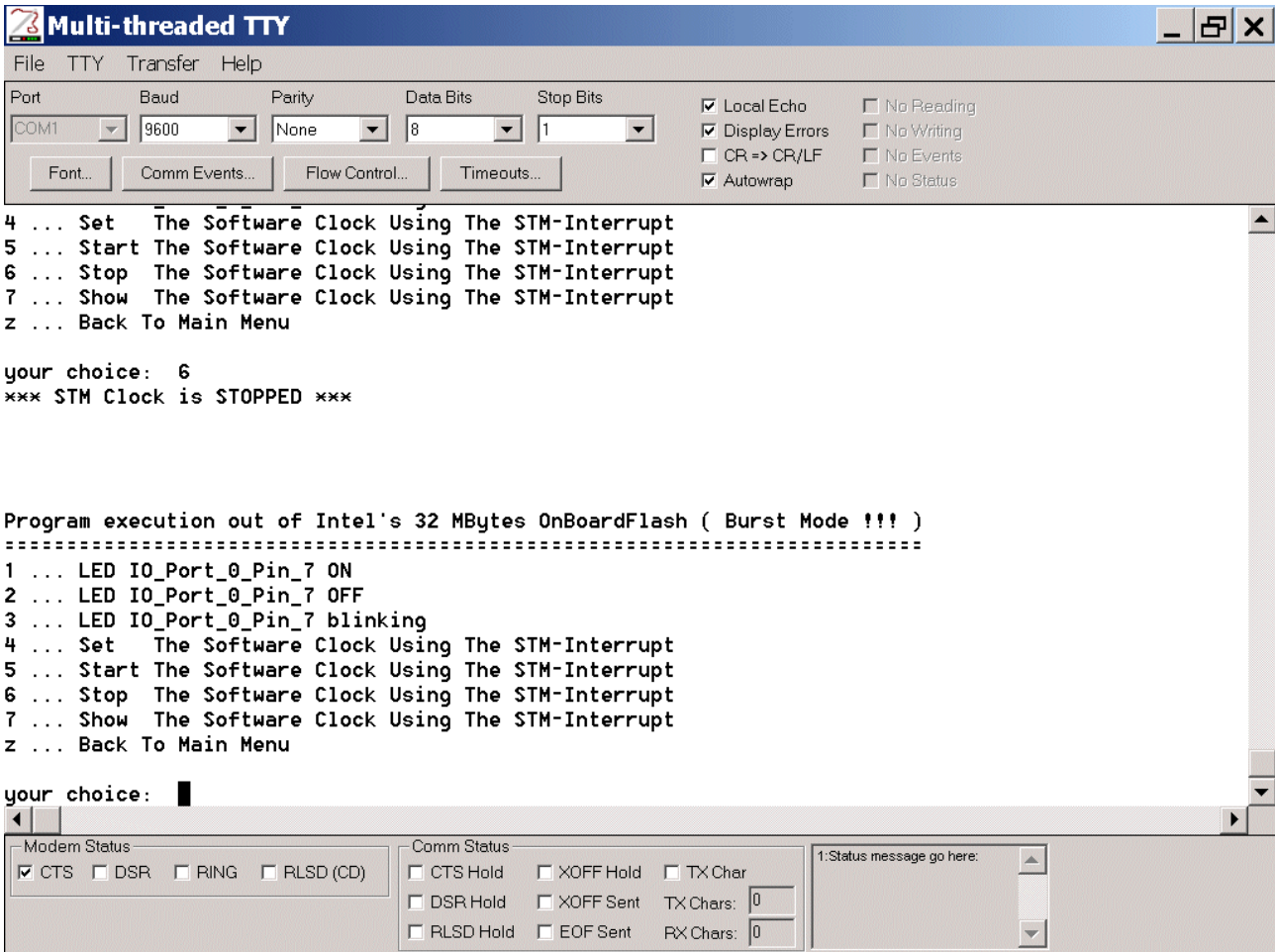
```

At the bottom, there are sections for Modem Status (CTS, DSR, RING, RLSD (CD)) and Comm Status (CTS Hold, XOFF Hold, TX Char, DSR Hold, XOFF Sent, TX Chars: 0, RLSD Hold, EOF Sent, RX Chars: 0). A status message field is also present.

Insert 5



Insert 6



Multi-threaded TTY

File TTY Transfer Help

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

Local Echo No Reading
 Display Errors No Writing
 CR => CR/LF No Events
 Autowrap No Status

Font... Comm Events... Flow Control... Timeouts...

```

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 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

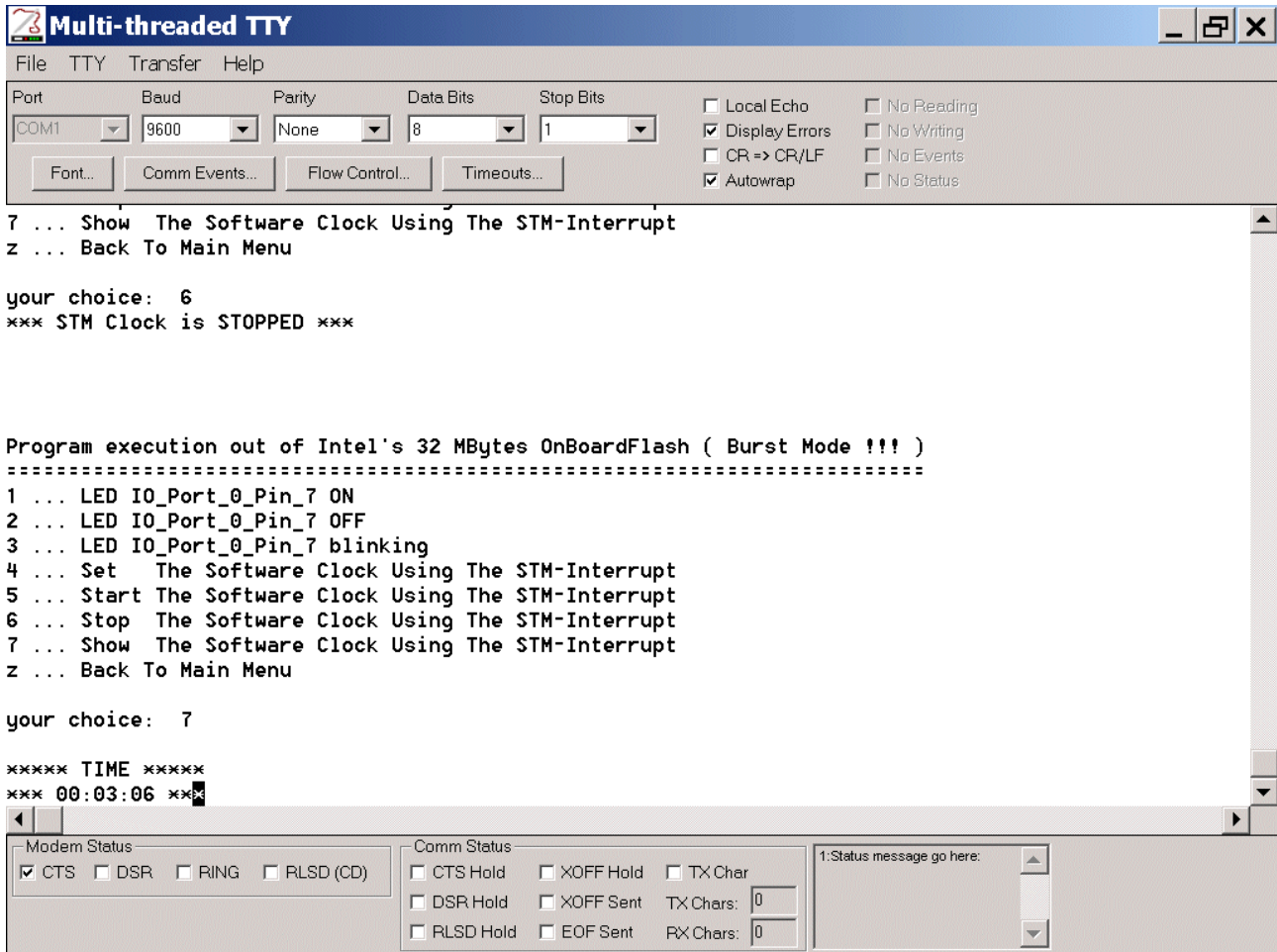
your choice: █
  
```

Modem Status: CTS DSR RING RLSD (CD)

Comm Status: CTS Hold XOFF Hold TX Char
 DSR Hold XOFF Sent TX Chars: 0
 RLSD Hold EOF Sent RX Chars: 0

1: Status message go here:

Insert 7



Multi-threaded TTY

File TTY Transfer Help

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

Local Echo No Reading
 Display Errors No Writing
 CR => CR/LF No Events
 Autowrap No Status

Font... Comm Events... Flow Control... Timeouts...

```

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 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

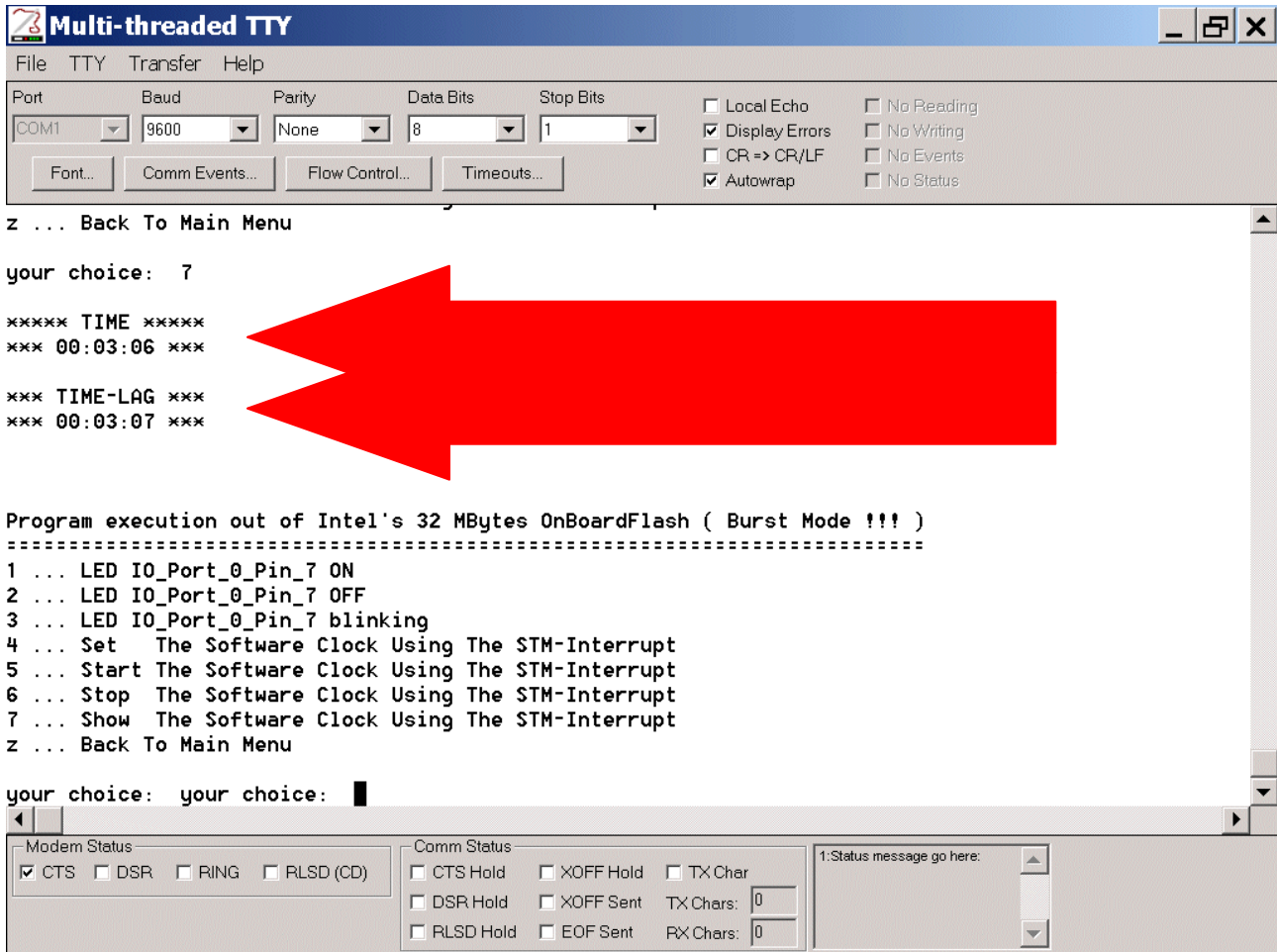
***** TIME *****
*** 00:03:06 ***
  
```

Modem Status: CTS DSR RING RLSD (CD)

Comm Status: CTS Hold XOFF Hold TX Char
 DSR Hold XOFF Sent TX Chars: 0
 RLSD Hold EOF Sent RX Chars: 0

1: Status message go here:

Insert z



Multi-threaded TTY

File TTY Transfer Help

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

Local Echo No Reading
 Display Errors No Writing
 CR => CR/LF No Events
 Autowrap No Status

Font... Comm Events... Flow Control... Timeouts...

```

z ... Back To Main Menu

your choice: 7

**** TIME ****
*** 00:03:06 ***

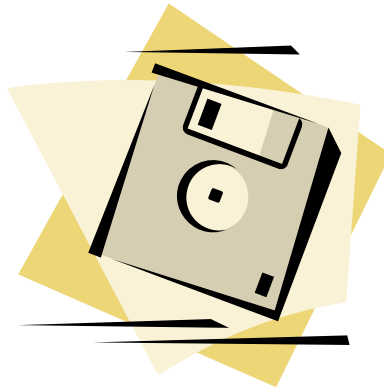
*** TIME-LAG ***
*** 00:03:07 ***

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

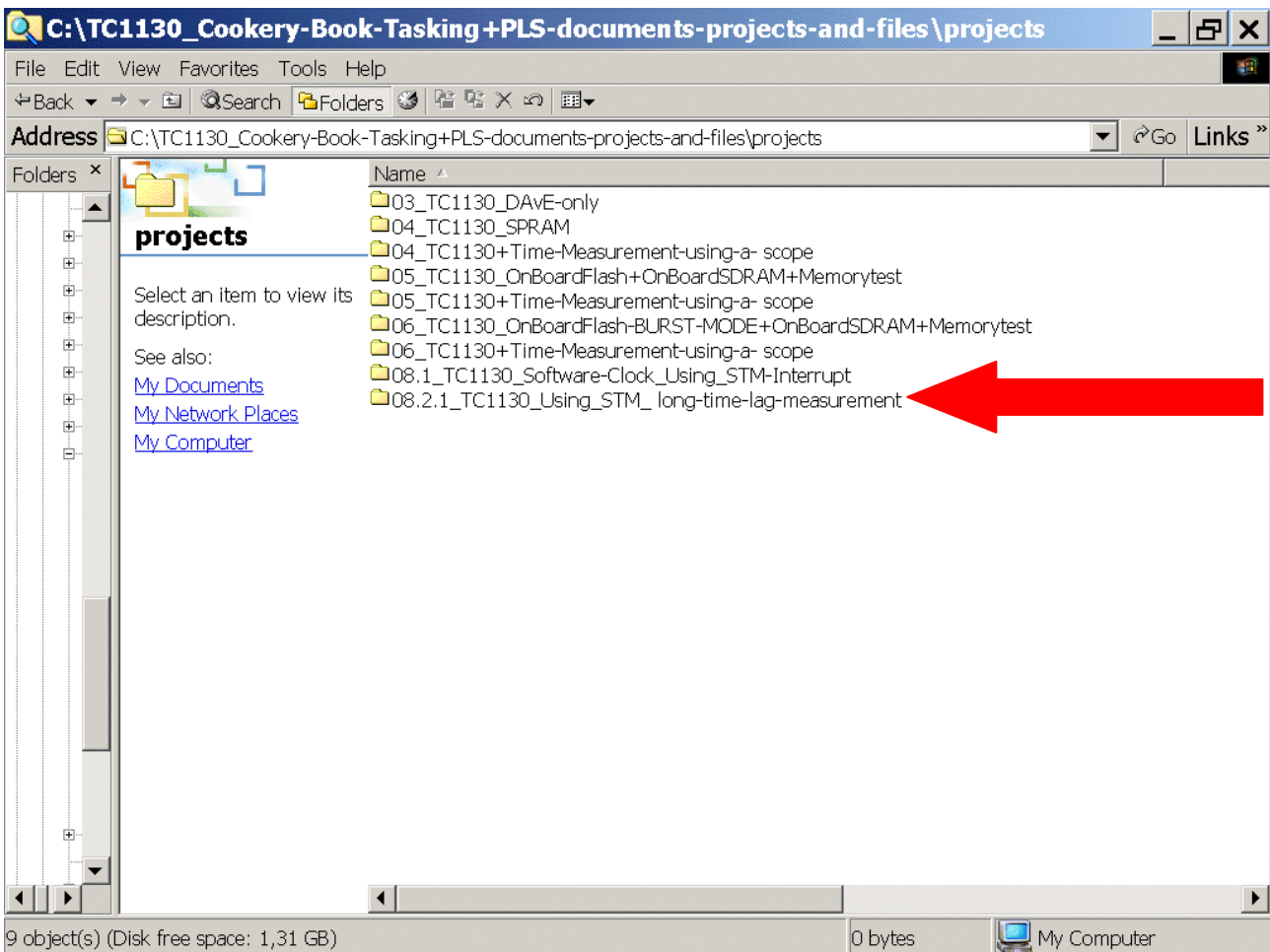
your choice: your choice: █
  
```

Modem Status: CTS DSR RING RLSD (CD)
 Comm Status: CTS Hold XOFF Hold TX Char
 DSR Hold XOFF Sent TX Chars: 0
 RLSD Hold EOF Sent RX Chars: 0

1: Status message go here:



We recommend now to **copy and store** your project-directory “C:\TC1130” to “08.2.1_TC1130_Using_STM_long-time-lag-measurement”:



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

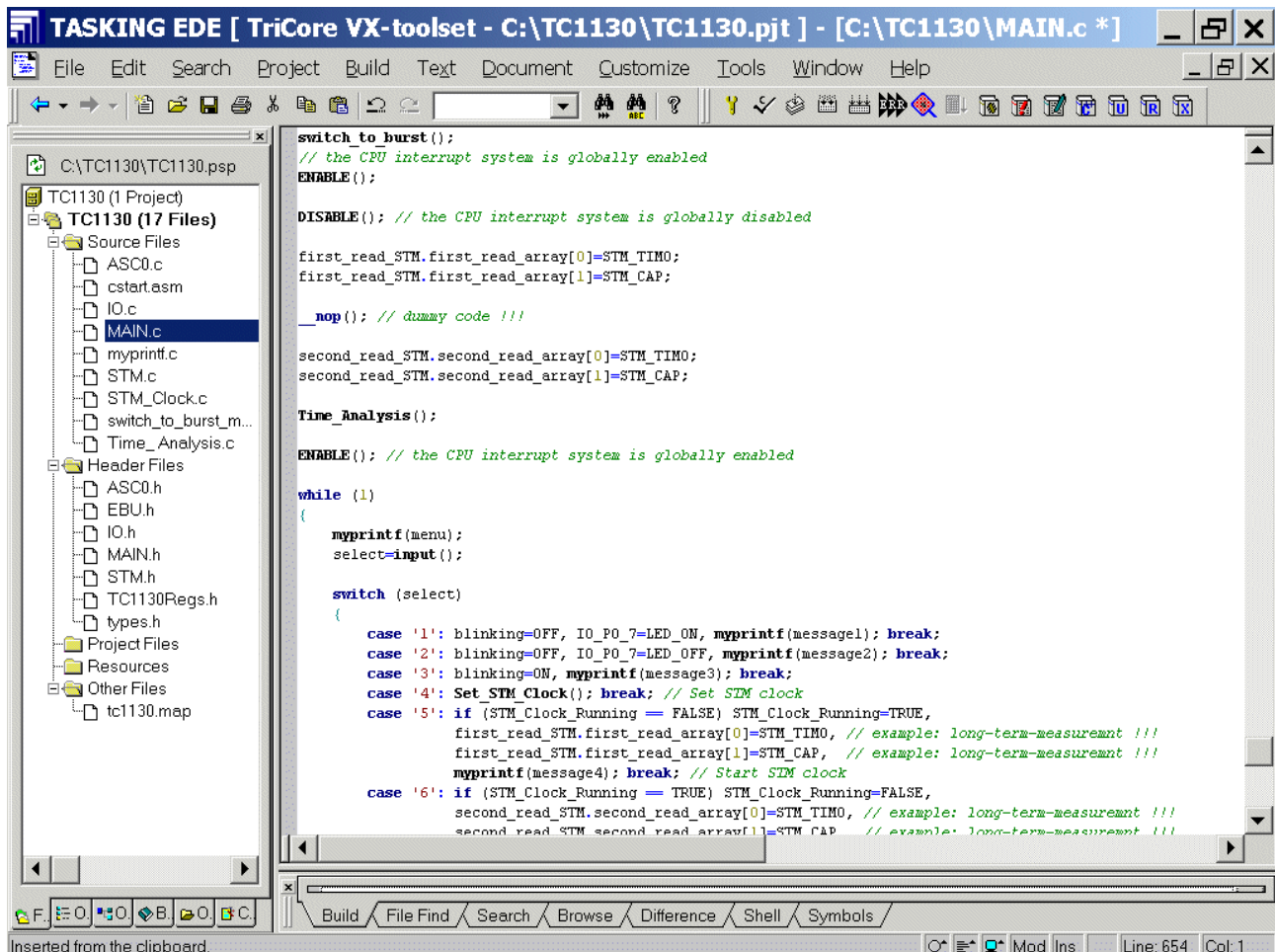
first_read_STM.first_read_array[0]=STM_TIM0;
first_read_STM.first_read_array[1]=STM_CAP;

__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
```



The screenshot shows the TASKING EDE IDE interface. The left pane displays the project structure for TC1130, with the 'Source Files' folder expanded to show 'MAIN.c' selected. The main editor window displays the following C code:

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

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

while (1)
{
    myprintf(menu);
    select=input();

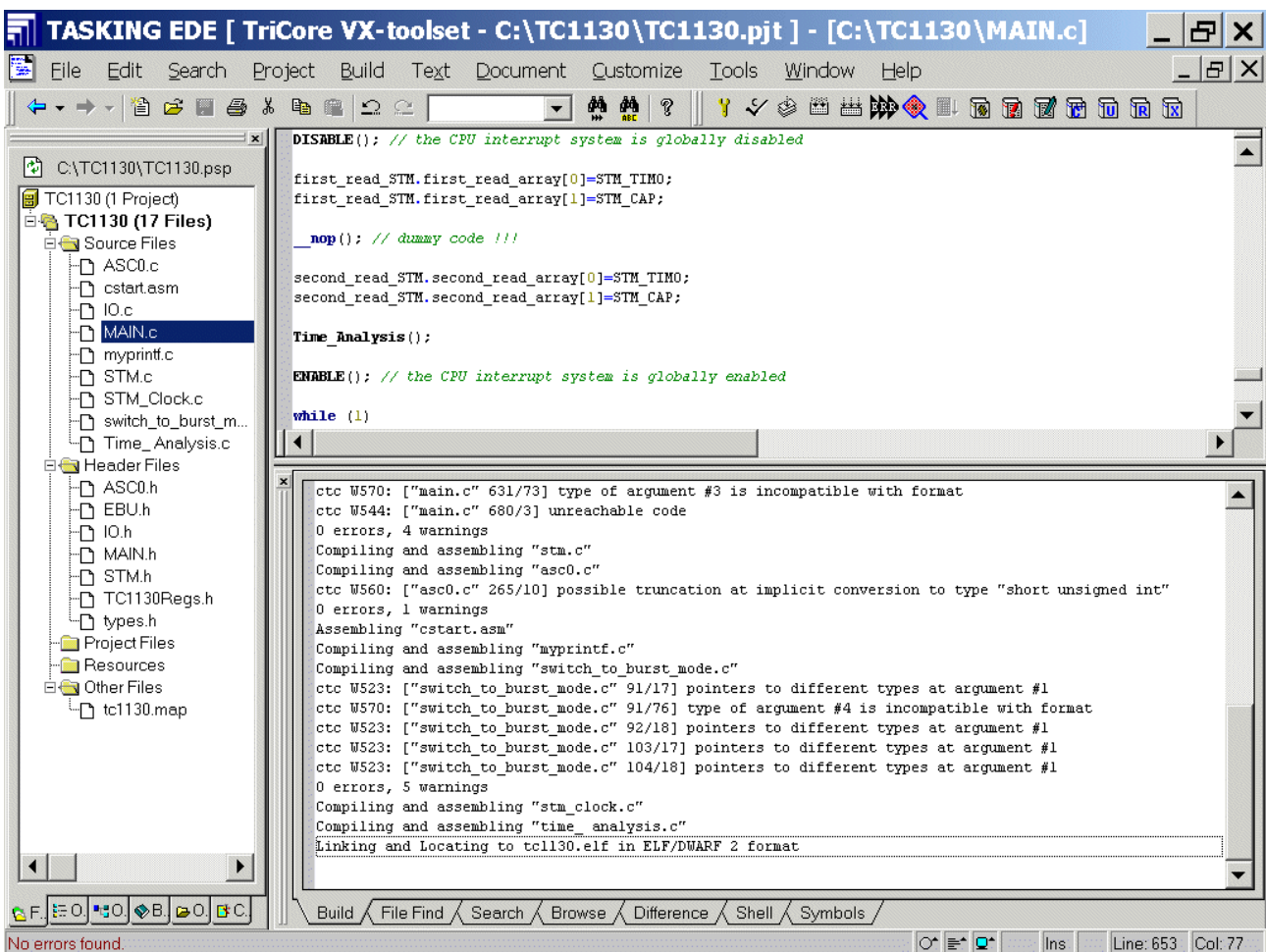
    switch (select)
    {
        case '1': blinking=OFF, IO_PO_7=LED_ON, myprintf(message1); break;
        case '2': blinking=OFF, IO_PO_7=LED_OFF, myprintf(message2); break;
        case '3': blinking=ON, myprintf(message3); break;
        case '4': Set_STM_Clock(); break; // Set STM clock
        case '5': if (STM_Clock_Running == FALSE) STM_Clock_Running=TRUE,
            first_read_STM.first_read_array[0]=STM_TIM0, // example: long-term-measurmnt !!!
            first_read_STM.first_read_array[1]=STM_CAP, // example: long-term-measurmnt !!!
            myprintf(message4); break; // Start STM clock
        case '6': if (STM_Clock_Running == TRUE) STM_Clock_Running=FALSE,
            second_read_STM.second_read_array[0]=STM_TIM0, // example: long-term-measurmnt !!!
            second_read_STM.second_read_array[1]=STM_CAP // example: long-term-measurmnt !!!
    }
}
```

The status bar at the bottom indicates the cursor is at Line: 654, Col: 1.

Generate your application program:

Build
Rebuild

OR



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

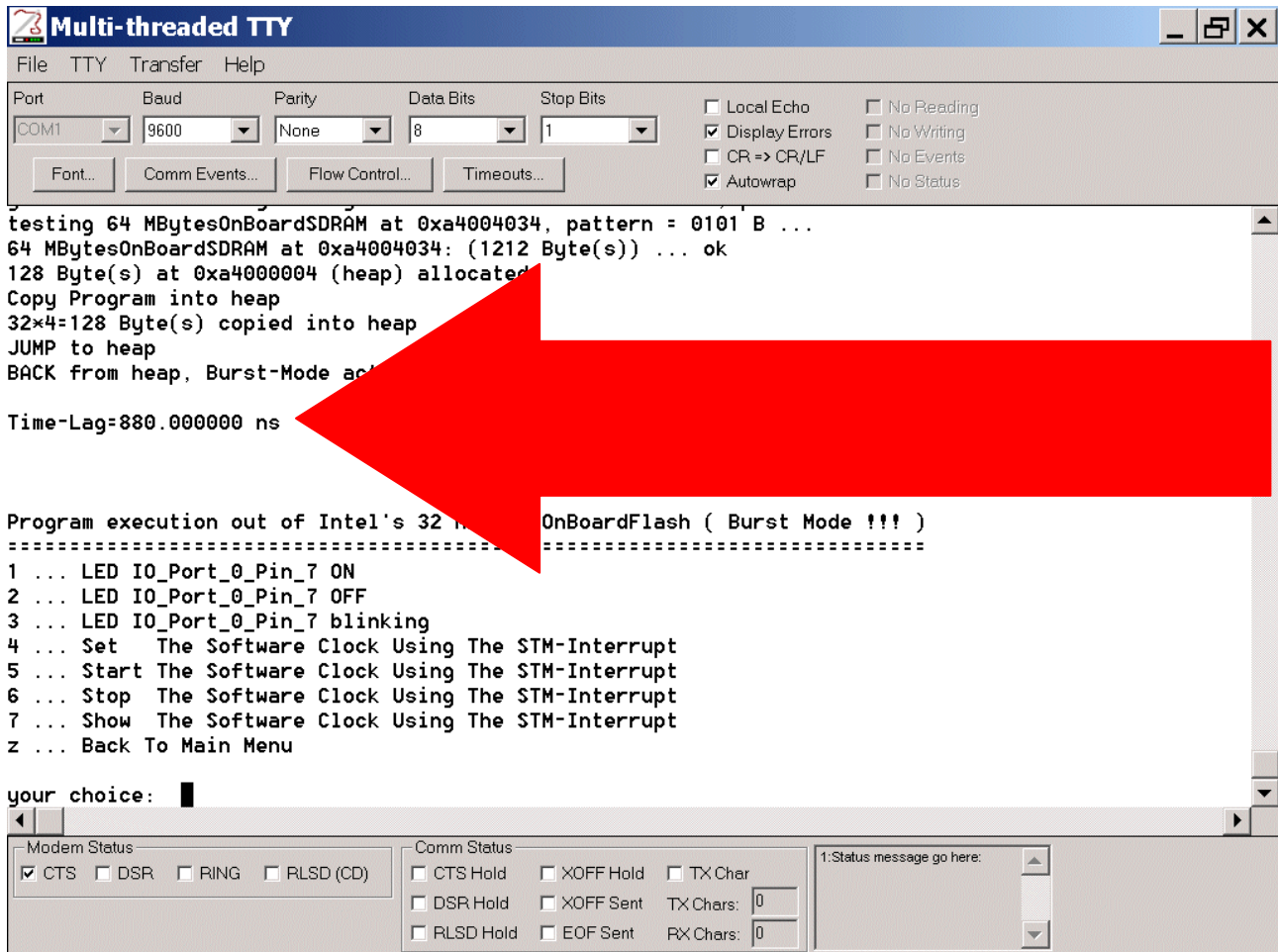
File – Close Workspace

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

File TTY Transfer Help

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

Local Echo No Reading
 Display Errors No Writing
 CR => CR/LF No Events
 Autowrap No Status

Font... Comm Events... Flow Control... Timeouts...

```

testing 64 MBytesOnBoardSDRAM at 0xa4004034, pattern = 0101 B ...
64 MBytesOnBoardSDRAM at 0xa4004034: (1212 Byte(s)) ... ok
128 Byte(s) at 0xa4000004 (heap) allocated
Copy Program into heap
32x4=128 Byte(s) copied into heap
JUMP to heap
BACK from heap, Burst-Mode act

Time-Lag=880.000000 ns

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: █
  
```

-----Modem Status----- -----Comm Status----- 1:Status message go here:

CTS DSR RING RLSD (CD) CTS Hold XOFF Hold TX Char TX Chars: 0
 DSR Hold XOFF Sent TX Chars: 0
 RLSD Hold EOF Sent RX Chars: 0

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:

```
// Additionally: 200 "NOPs" - should be about 1,333 µs (200*6,667ns)
__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();__nop();__nop();__nop();
```

TASKING EDE [TriCore VX-toolset - C:\TC1130\TC1130.pjt] - [C:\TC1130\MAIN.c *]

File Edit Search Project Build Text Document Customize Tools Window Help

```

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 !!!

// 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();

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

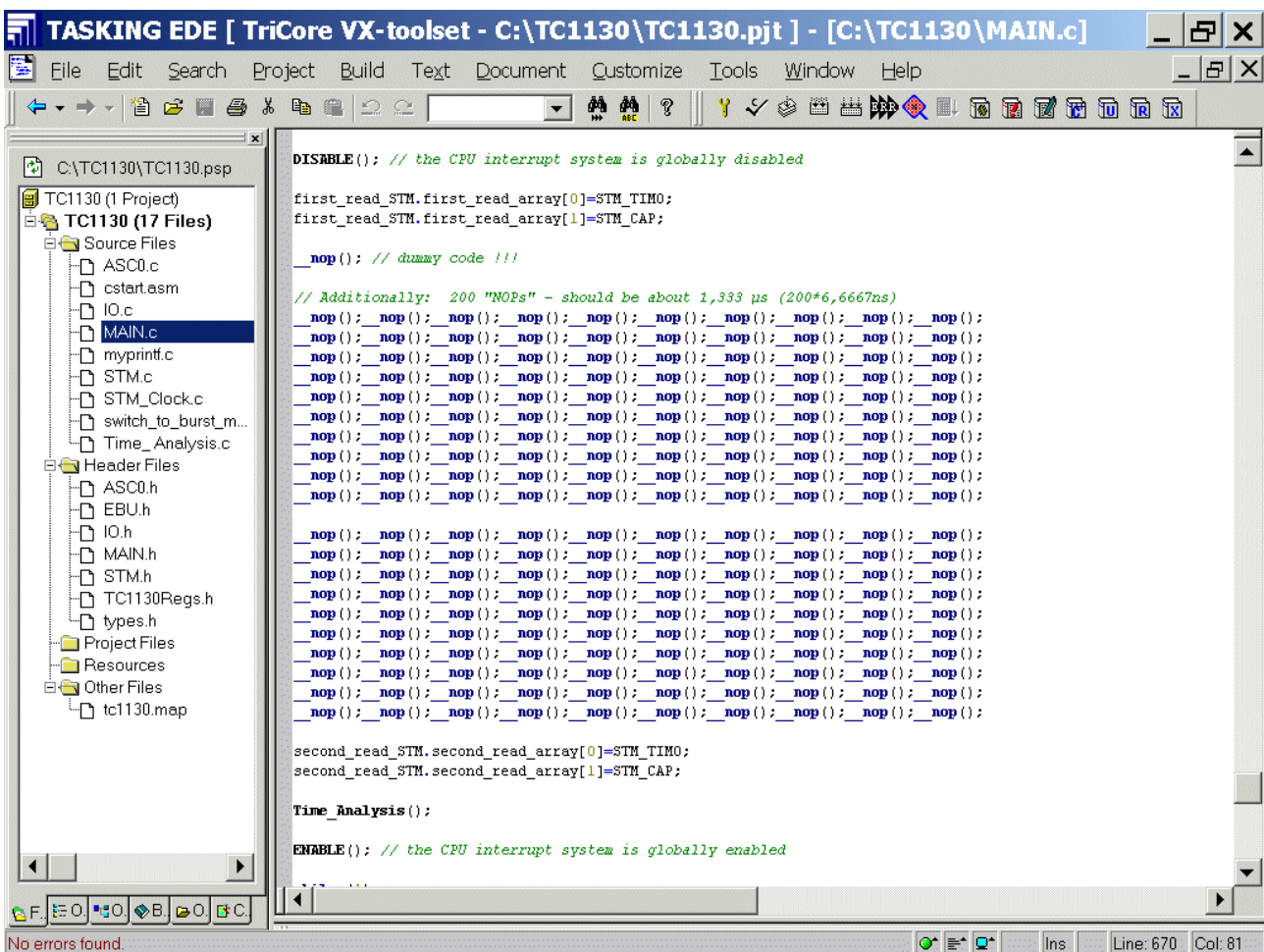
```

Inserted from the clipboard. Mad |ns Line: 670 Col: 81

Generate your application program:

Build
Rebuild

OR



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

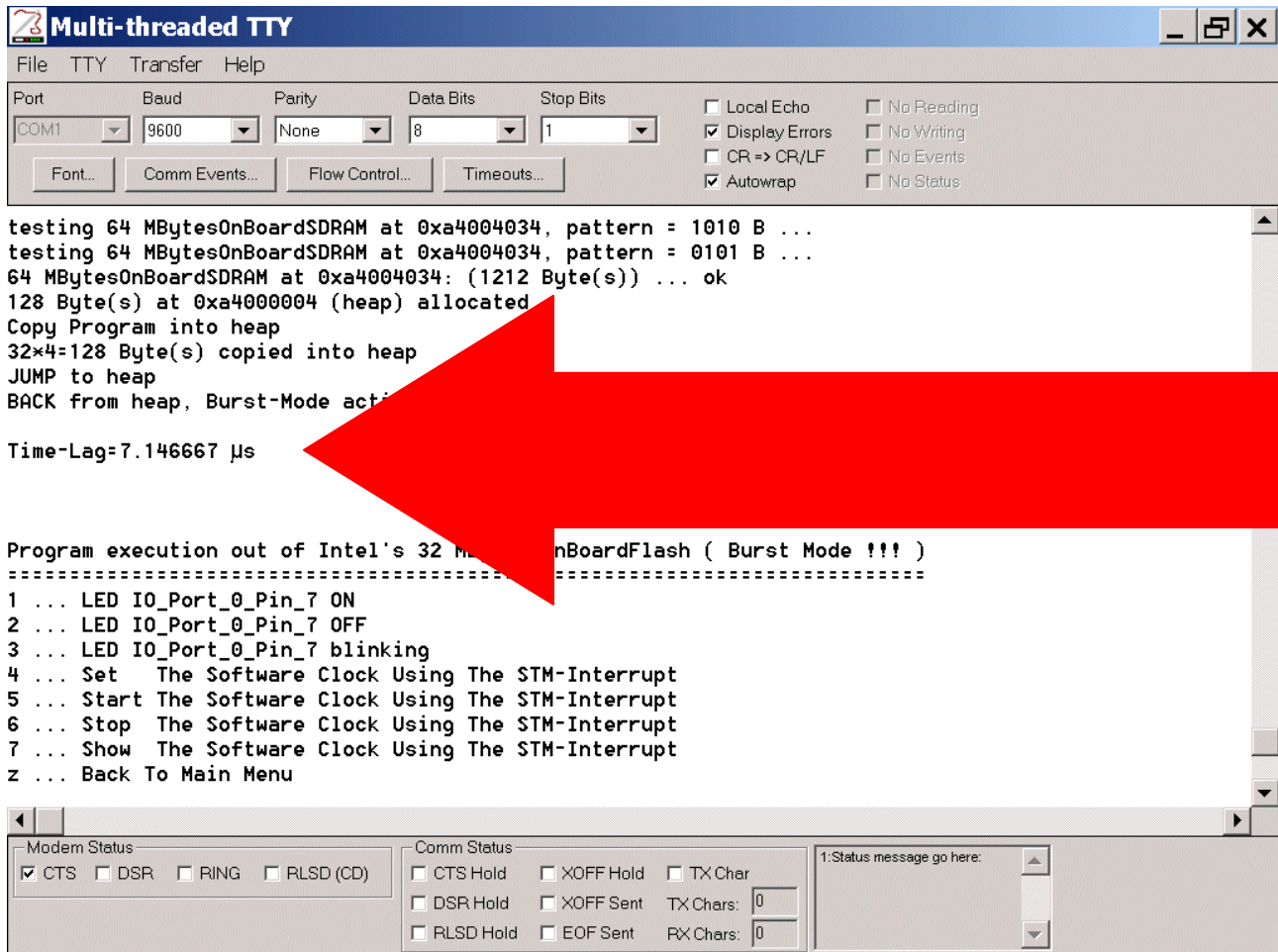
File – Close Workspace

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

File TTY Transfer Help

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

Local Echo No Reading
 Display Errors No Writing
 CR => CR/LF No Events
 Autowrap No Status

testing 64 MBytesOnBoardSDRAM at 0xa4004034, pattern = 1010 B ...
testing 64 MBytesOnBoardSDRAM at 0xa4004034, pattern = 0101 B ...
64 MBytesOnBoardSDRAM at 0xa4004034: (1212 Byte(s)) ... ok
128 Byte(s) at 0xa4000004 (heap) allocated
Copy Program into heap
32*4=128 Byte(s) copied into heap
JUMP to heap
BACK from heap, Burst-Mode acti

Time-Lag=7.146667 µs

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

```

Modem Status: CTS DSR RING RLSD (CD)

Comm Status: CTS Hold XOFF Hold TX Char
 DSR Hold XOFF Sent TX Chars: 0
 RLSD Hold EOF Sent RX Chars: 0

1:Status message go here:



Conclusion:

We expected **1,33 µs** for additional 200 NOPs ($200 * 6.667 \text{ ns}$).

We got $7,146 \mu\text{s} - 880 \text{ ns} = 6,3 \mu\text{s}$ instead of **1,33 µs** [Cause: LMB Bus Arbitration, EBU].

Just to satisfy curiosity:

Is it possible to measure even CPU clocks ($150 \text{ MHz} \rightarrow 6,6667 \text{ ns}$) ?

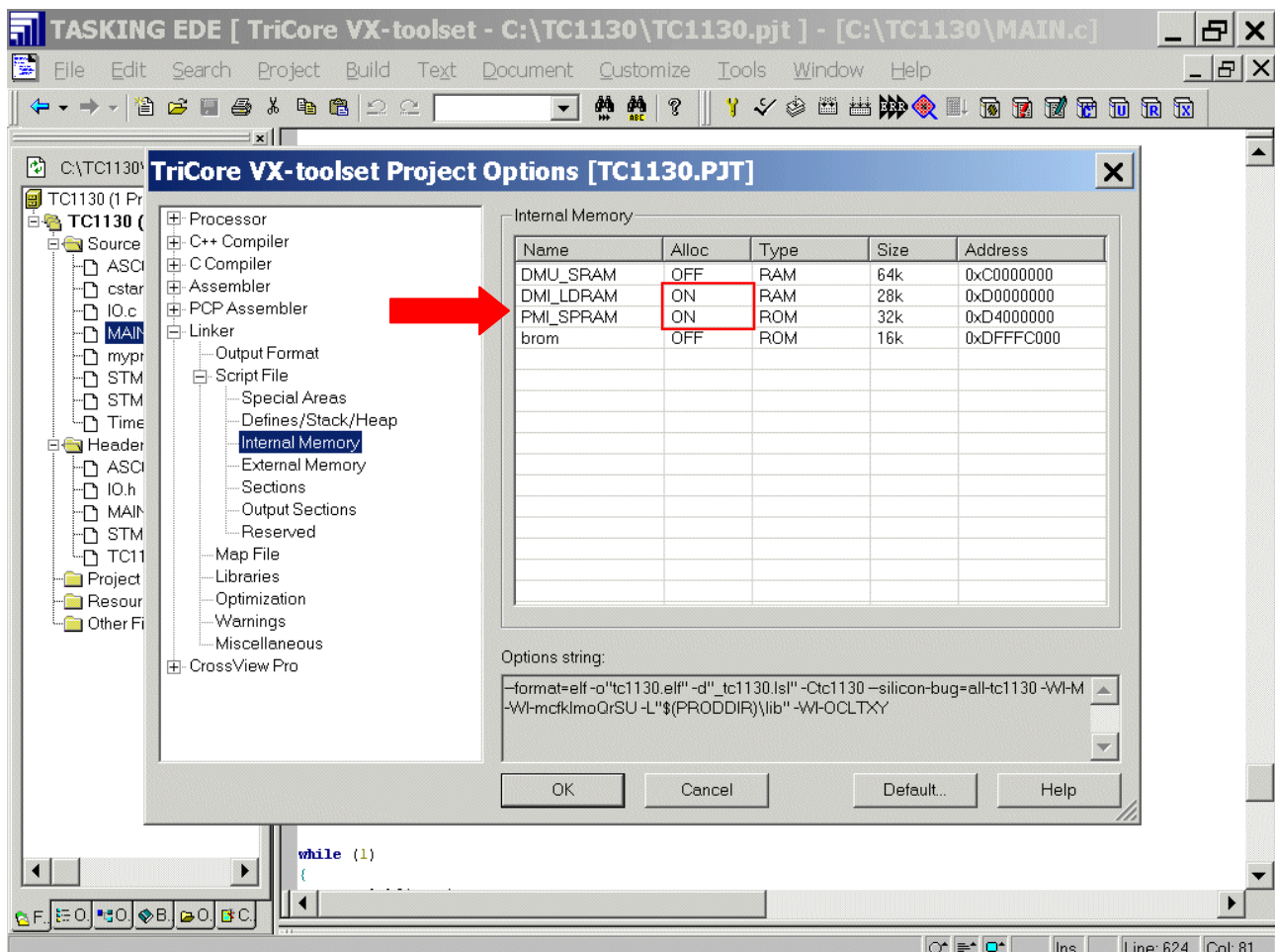
We used the programming example “04_TC1130_SPRAM”, deactivated all memories - except PMI_SPRAM and DMI_SPRAM, added software from chapter 8.1. upwards and got the following values:

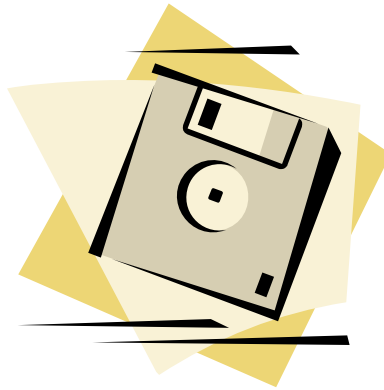
$1,546667 \mu\text{s} - 213 \text{ ns} = 1,3337 \mu\text{s}$ – THIS IS EXACTLY THE TIME WE CALCULATED FOR 200 NOPs ($1,3337 \mu\text{s} / 200 = 6,67 \text{ ns} = \text{ONE CPU CLOCK}$).

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

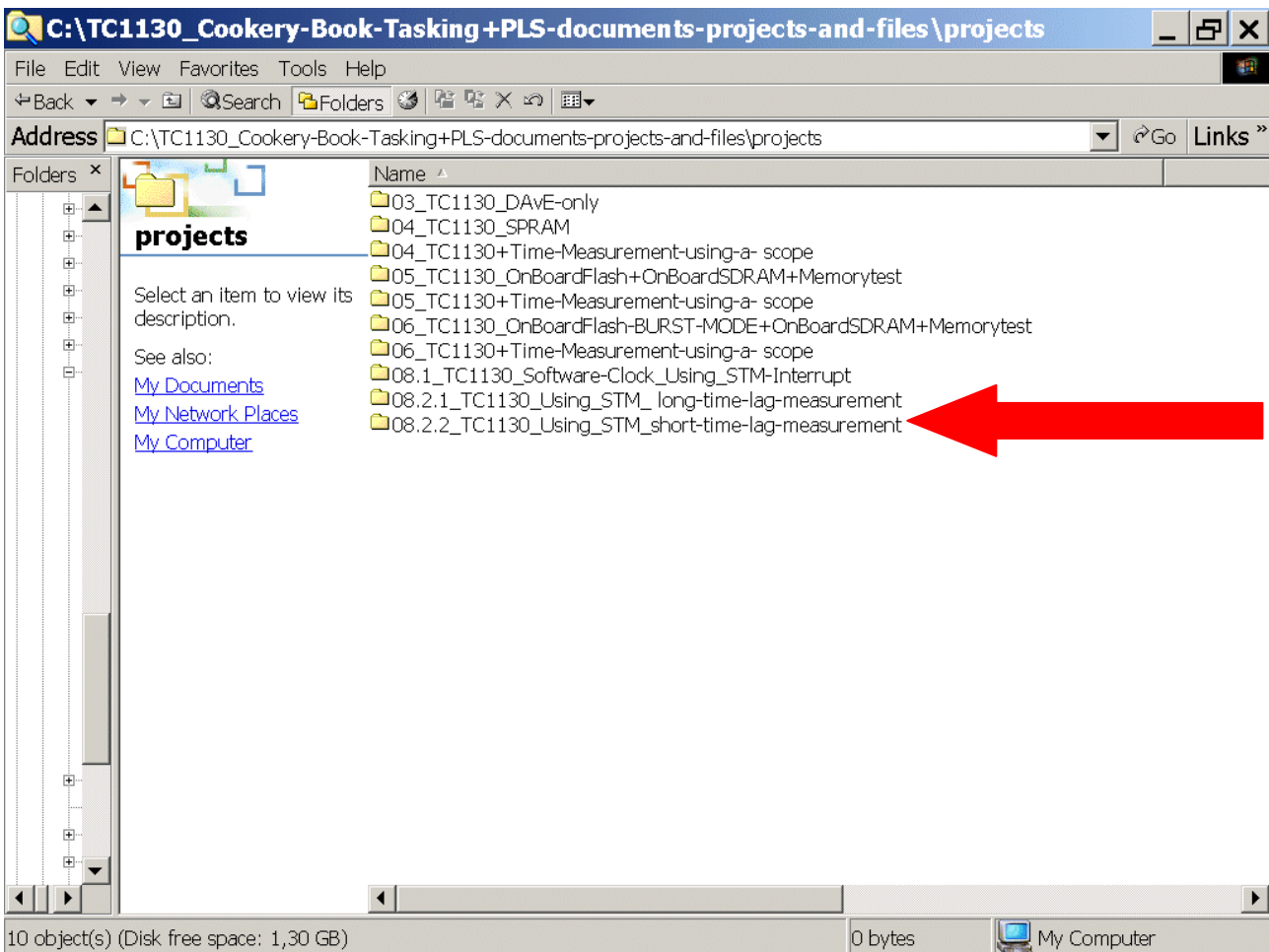
What we have learnt:

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.

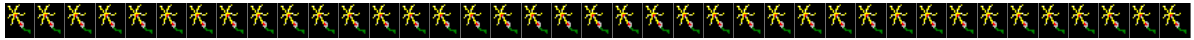




We recommend now to **copy and store** your project-directory “C:\TC1130” to “08.2.2_TC1130_Using_STM_short-time-lag-measurement”:



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:

<http://www.infineon.com>

Published by Infineon Technologies AG