AP08081

XC878CM-16FF

XC878 Easy Kit: "Cookery Book" for a hello world application.

Using DAvE (Code Generator) and

DAvE Bench (Open Platform for Free Tools: IDE, Compiler, Debugger, Utility Tools).

Microcontrollers

Never stop thinking

Edition 2010-02-25

Published by
Infineon Technologies AG
81726 München, Germany

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.

AP08048		
Revision History:	2010-03	V2.0
Previous Version:	none	

Page	Subjects (major changes since last revision)

We Listen to Your Comments

Any information within this document that you feel is wrong, unclear or missing at all?
Your feedback will help us to continuously improve the quality of this document.
Please send your proposal (including a reference to this document) to:

mcdocu.comments@infineon.com

Table of Contents **Page**

Note: Table of Contents see page 8.

Introduction:

This “Application Note / Appnote” is a Hands On Training / Cookery Book / step-by-step book.
It will help inexperienced users to get the XC878 Easy Kit up and running.

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

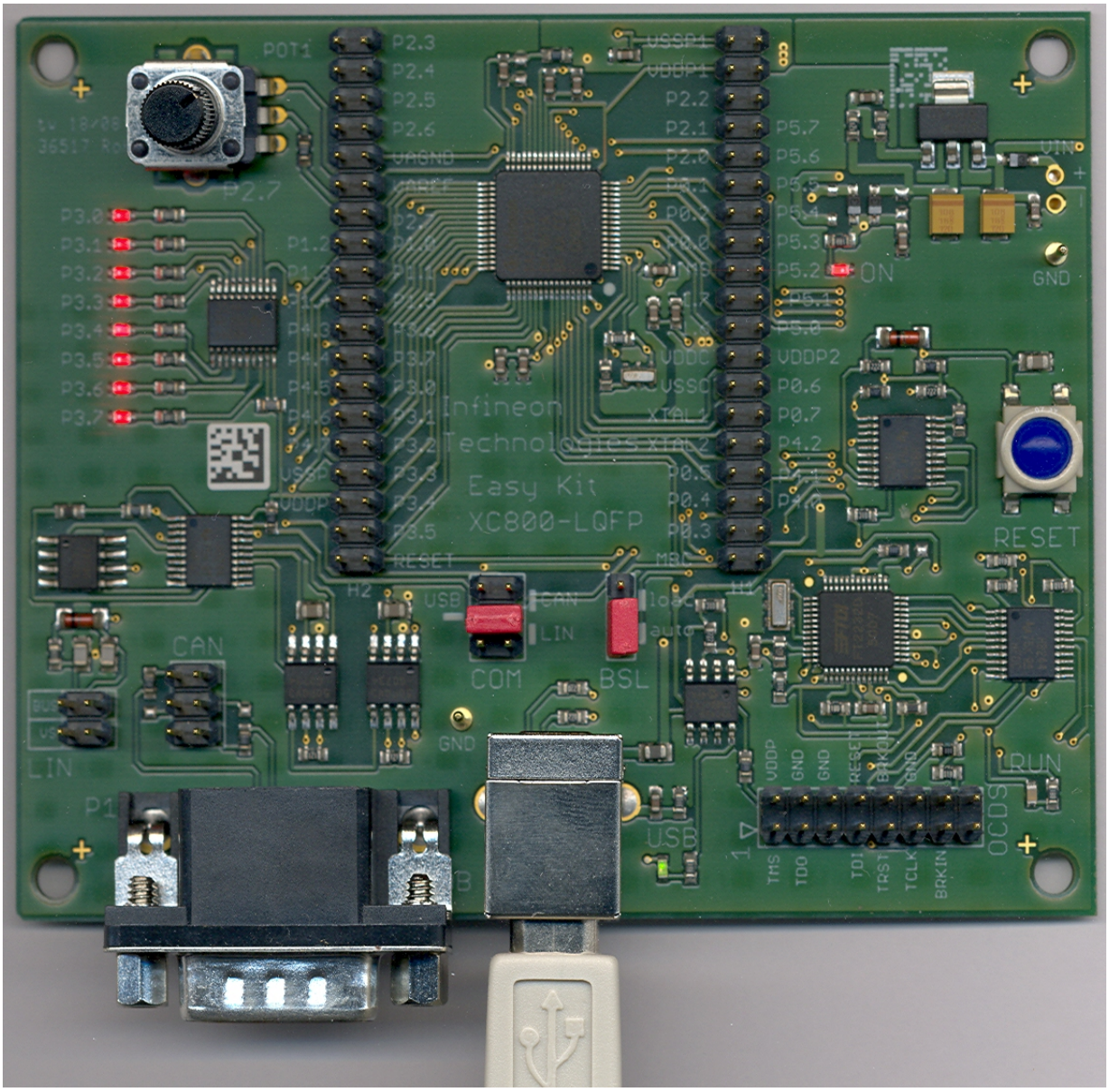
The purpose of this document is to gain know-how of the microcontroller and the tool-chain.
Additionally, the "hello world example" can easily be expanded to suit your needs.
You can connect either a part of - or your entire application to the XC878 Easy Kit.
You are also able to benchmark any of your algorithms to find out if the selected microcontroller fulfils all the required functions within the time frame needed.

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 the XC878 Easy Kit!

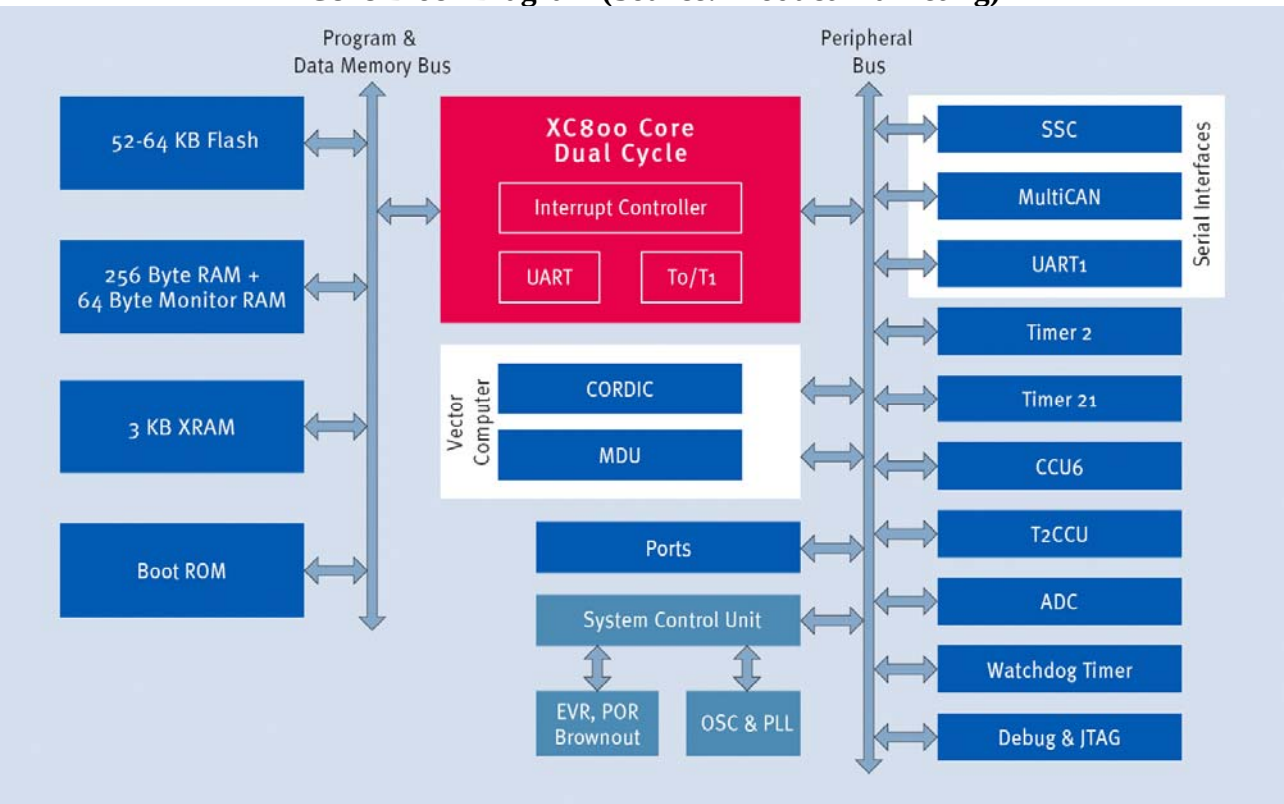
Programming Example

XC878 Easy Kit

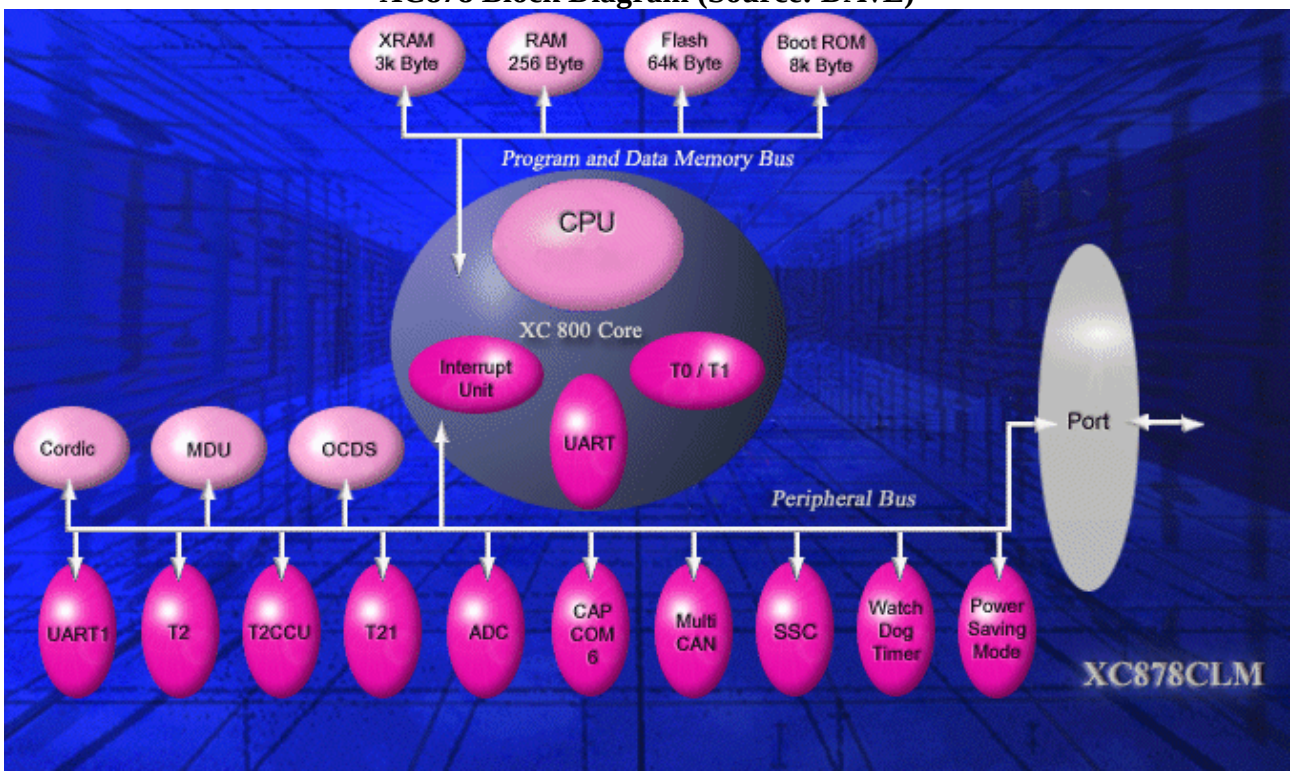


Used/selected microcontroller:

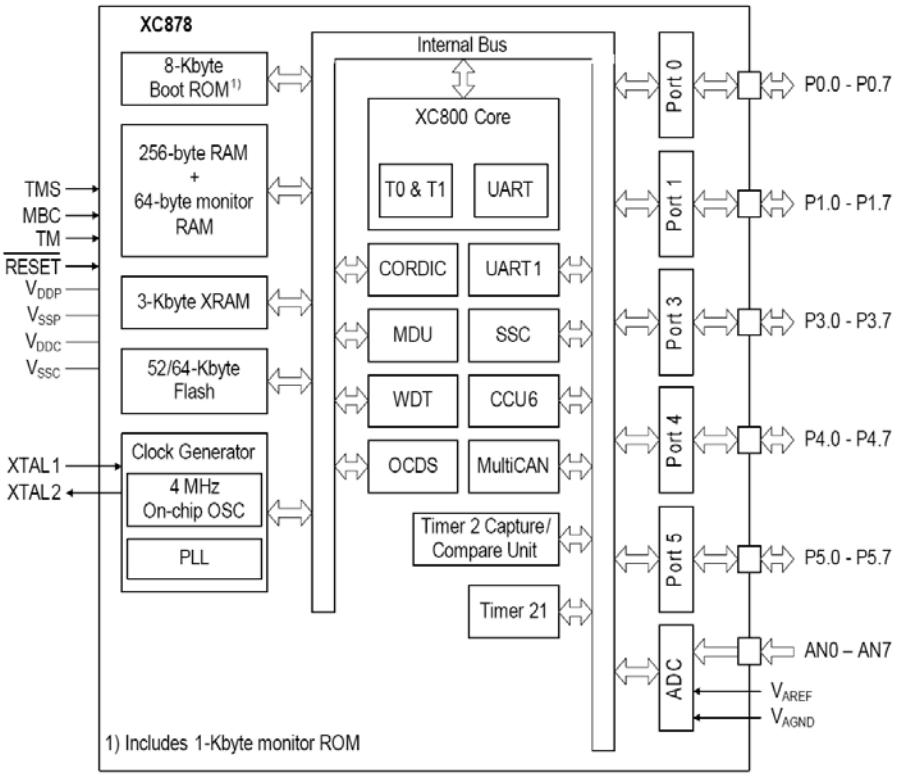
XC878 Block Diagram (Source: Product Marketing)



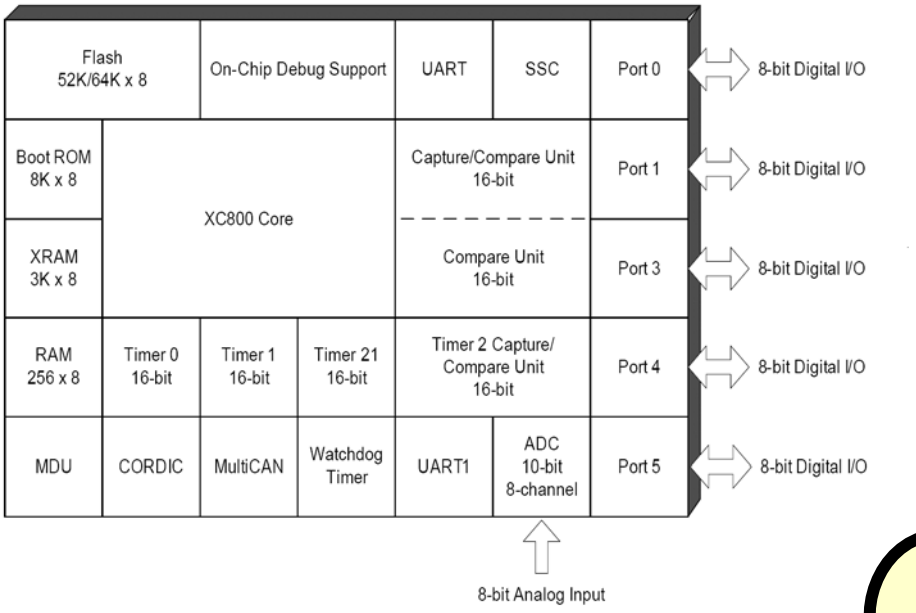
XC878 Block Diagram (Source: DAvE)



XC878 Block Diagram (Source: User's Manual)



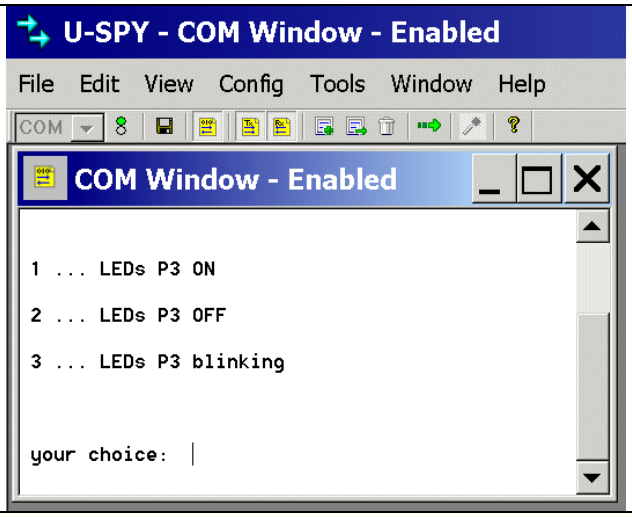
XC878 functional units (Source: User's Manual)



Note:
Just by comparing the different sources of block diagrams, you should be able to get a complete picture of the product and to answer some of your initial questions.

“Cookery book“

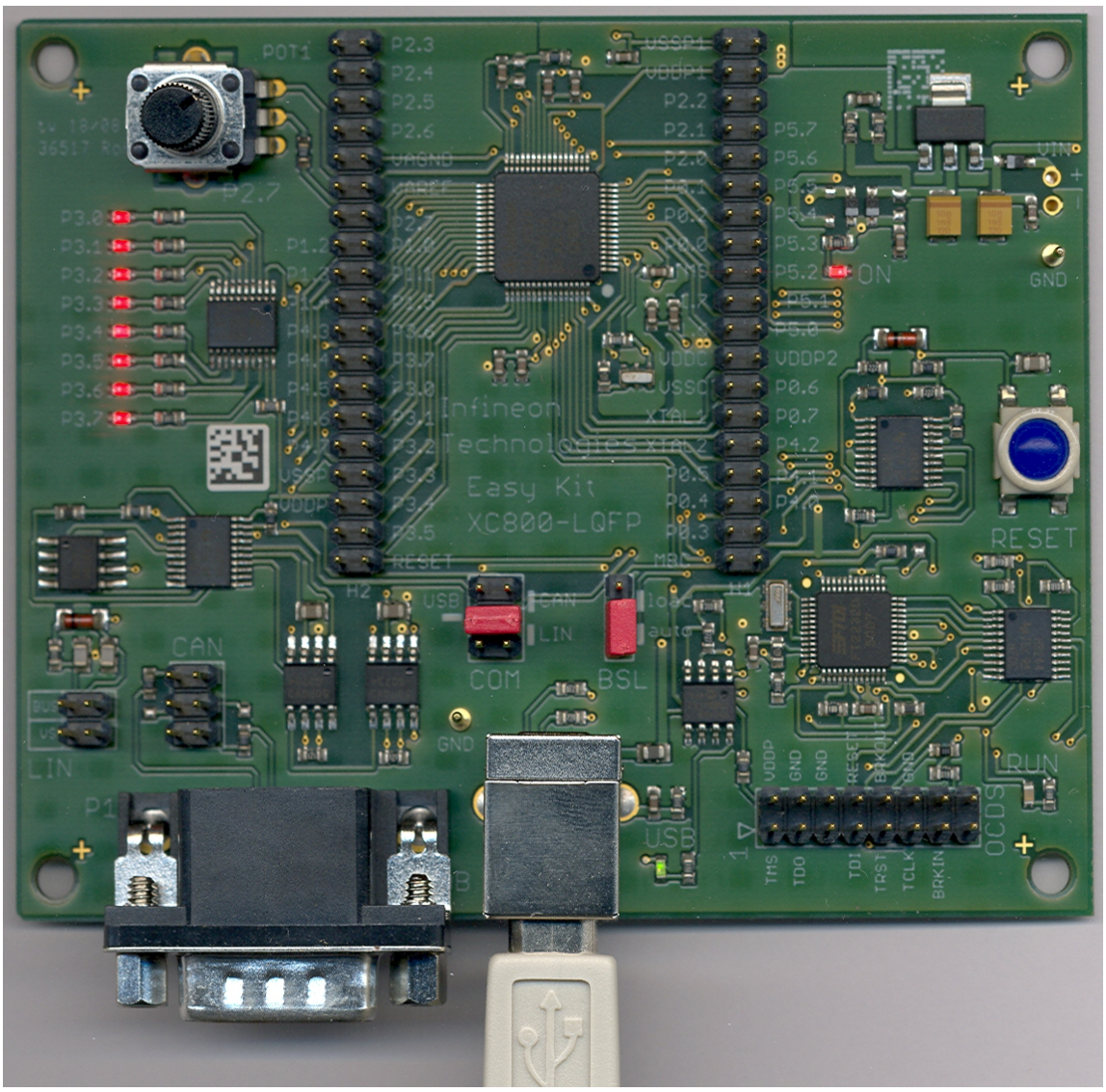
For your first programming example for the XC878 Easy Kit:

Your program:	
Chapter/ Step	***** Recipes *****
1.)	**DAS Installation + Connecting the XC878 Easy Kit**
2.)	**DAvE (program generator)** **DAvE Installation (mothersystem) + DAvE Update Installation (XC878.DIP) for XC878**
3.)	**Using DAvE** **Microcontroller initialization for your programming example**
4.)	**Using DAvE Bench** **Programming of your application (hello world) with the DAvE Bench tool chain**
5.)	**Using the debugger (DAvE Bench)**

Feedback:

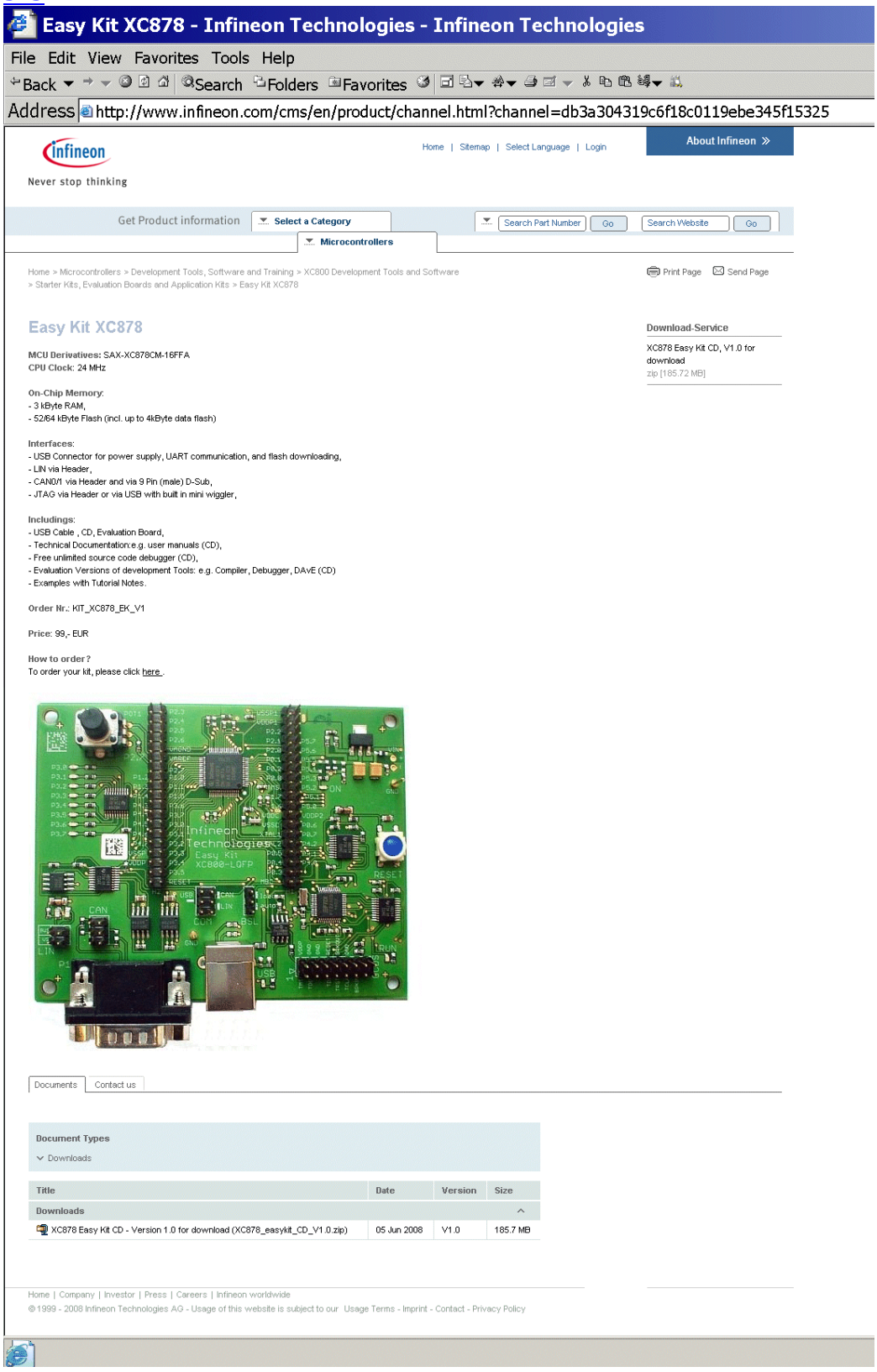
6.)	**Feedback**

1.) DAS Installation + Connecting the XC878 Easy Kit:



Screenshot of the XC878 Easy Kit Homepage:

http://www.infineon.com/cms/en/product/channel.html?channel=db3a304319c6f18c0119ebe345f15325



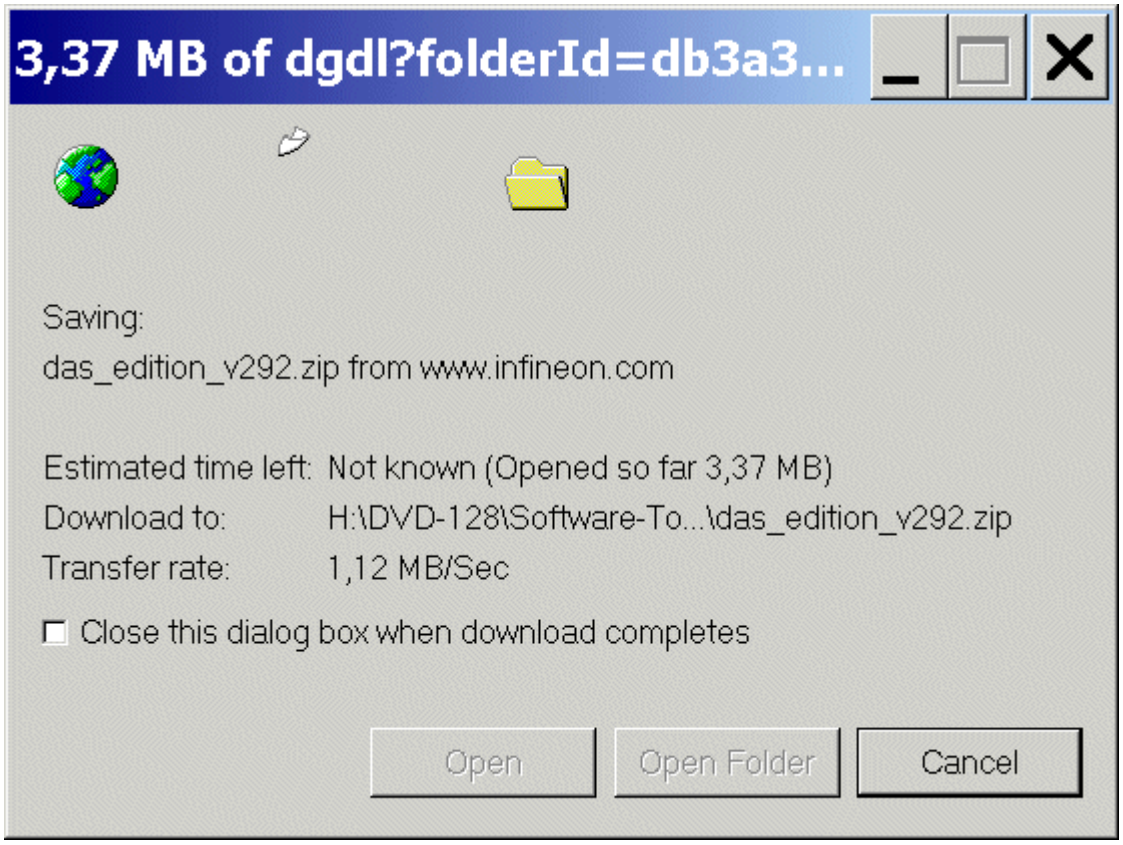
Note:
For further information, please refer to the XC878 Easy Kit Board Manual V1.0, April 2008.

Install the Infineon DAS (D e v i c e A c c e s s S e r v e r) Server:

Go to www.infineon.com/DAS:

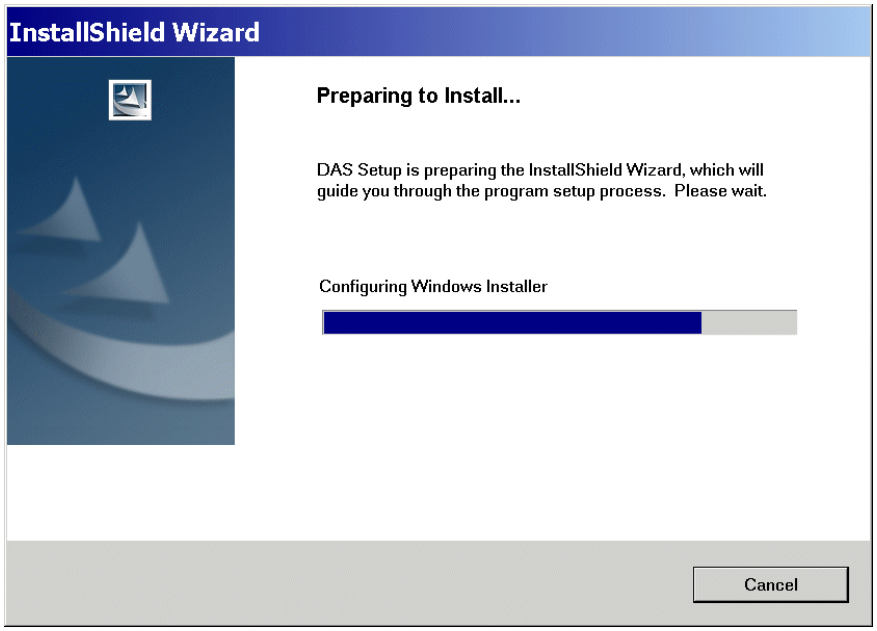
Note:
The DAS Server must be installed on your host computer!
The goal of the DAS software is to provide one single interface for all types of tools.
The USB Device driver communicates with the XC878 Easy Kit when connected to the host computer.
The USB Device driver for the XC878 Easy Kit USB interface is included in the DAS software.
A virtual COM port driver is also included.

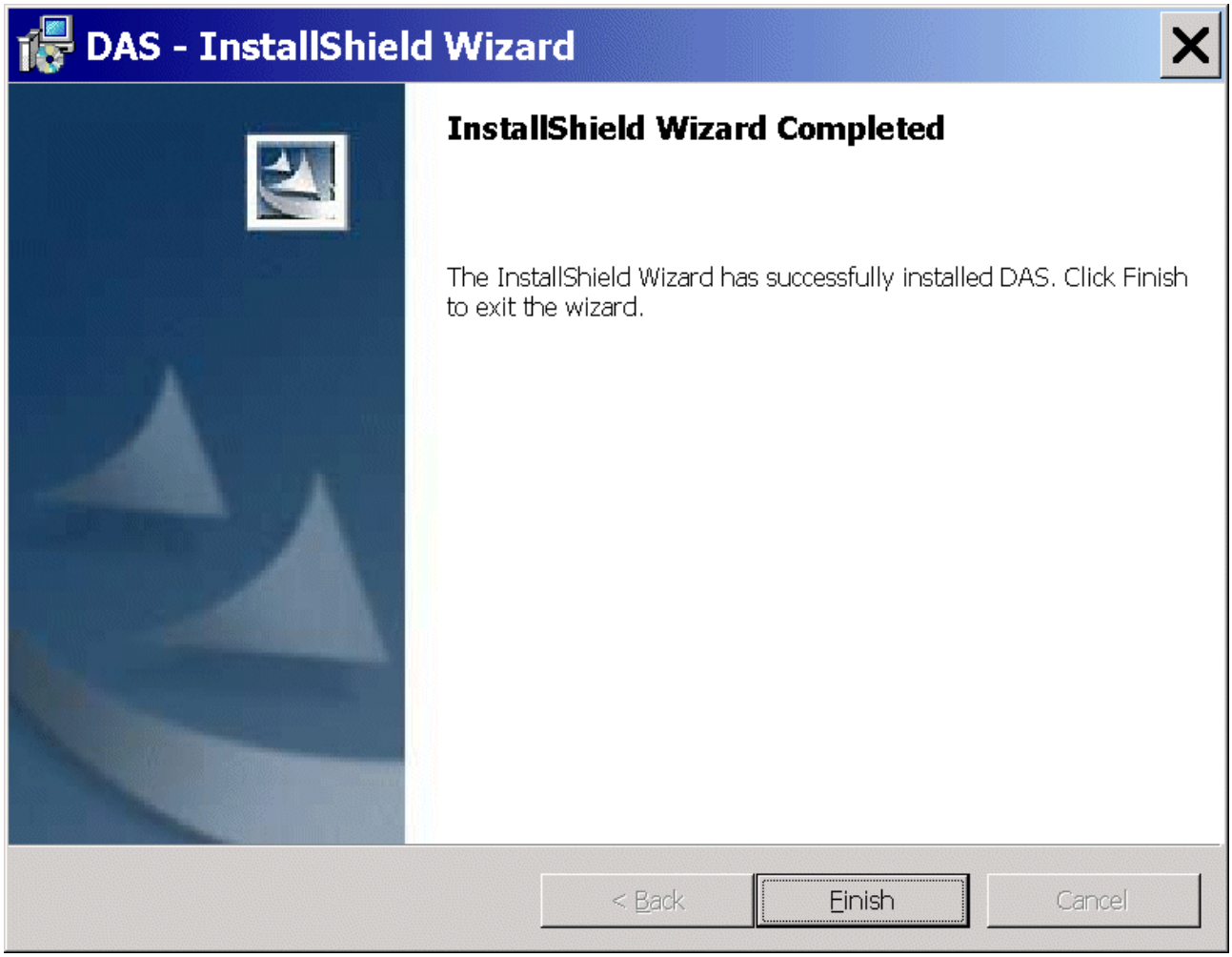
Download “The latest release version for download: DAS Edition v2.9.2”
(- or any higher version !!!):



Unzip das_edition_v292.zip and

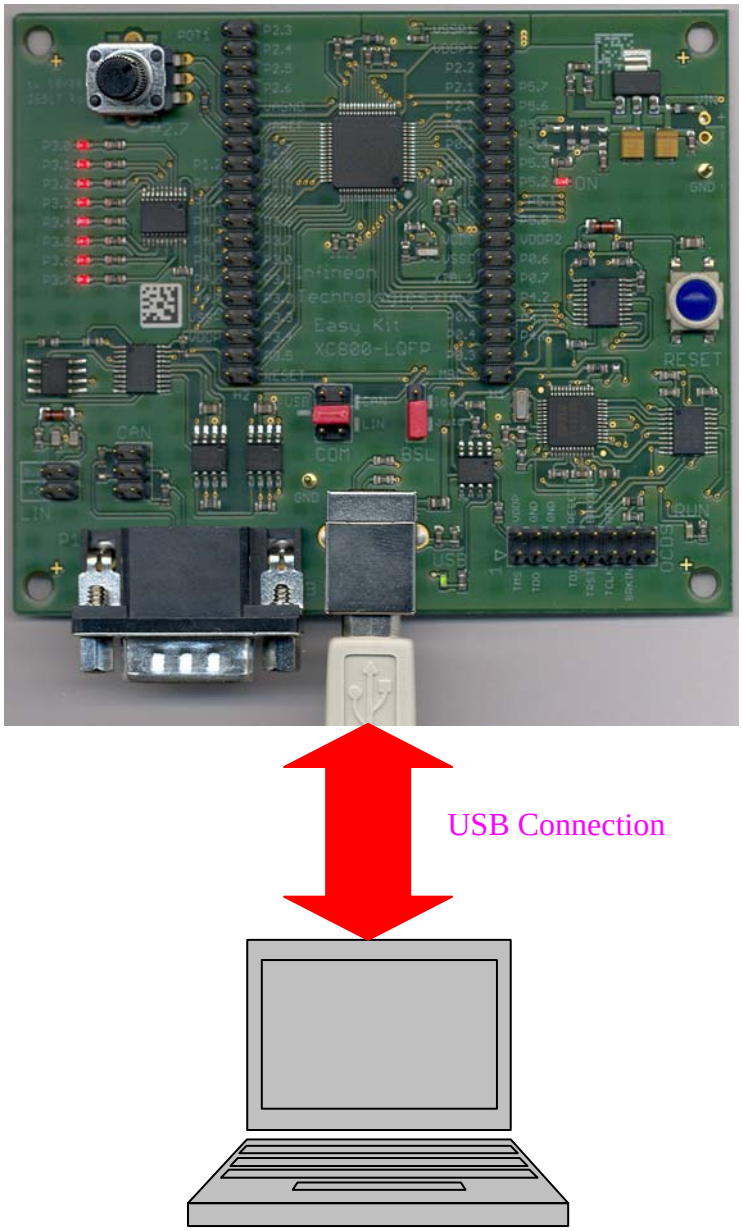
execute “DAS_v292_setup.exe” to install the DAS Server.



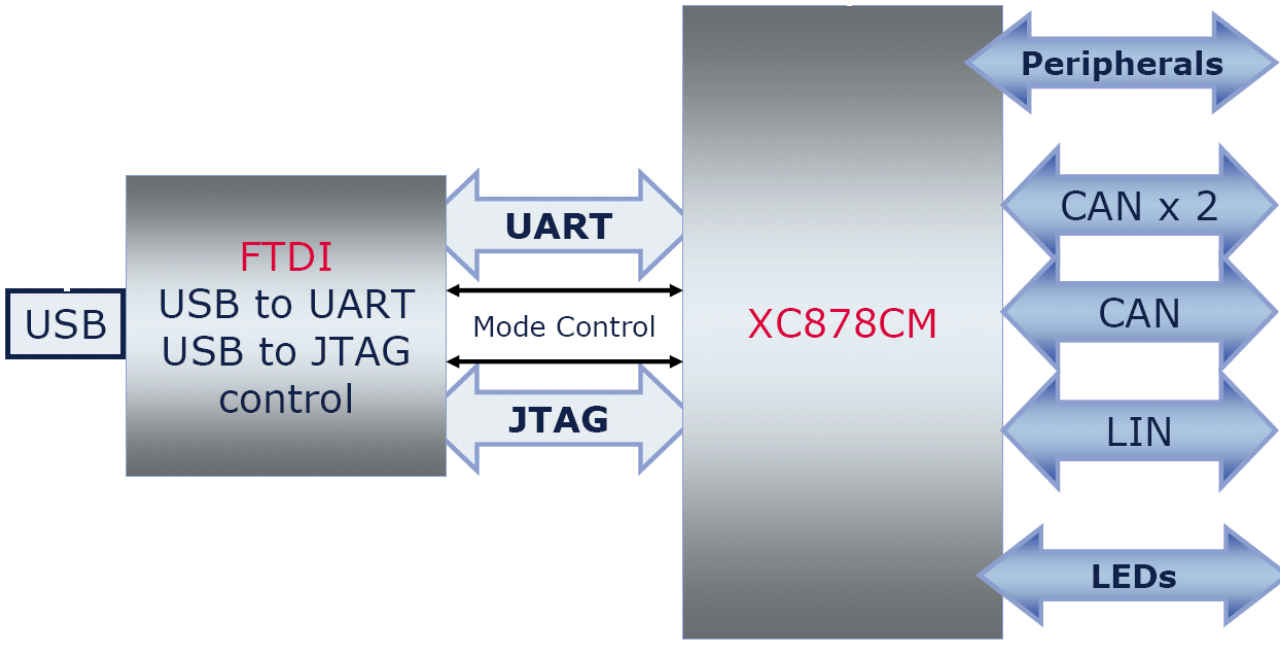


Click Finish

Connect the XC878 Easy Kit to the host computer:



USB Connection:
.) used for: UART communication (the UART/RS232/serial interface is available via USB as a virtual COM port of the second USB channel of the FTDI FT2232 Dual USB to UART/JTAG interface).
.) used for: On-Chip-Flash-Programming and Debugging (first USB channel of the FTDI FT2232 Dual USB to UART/JTAG interface).
.) the USB connection works also as the power supply.

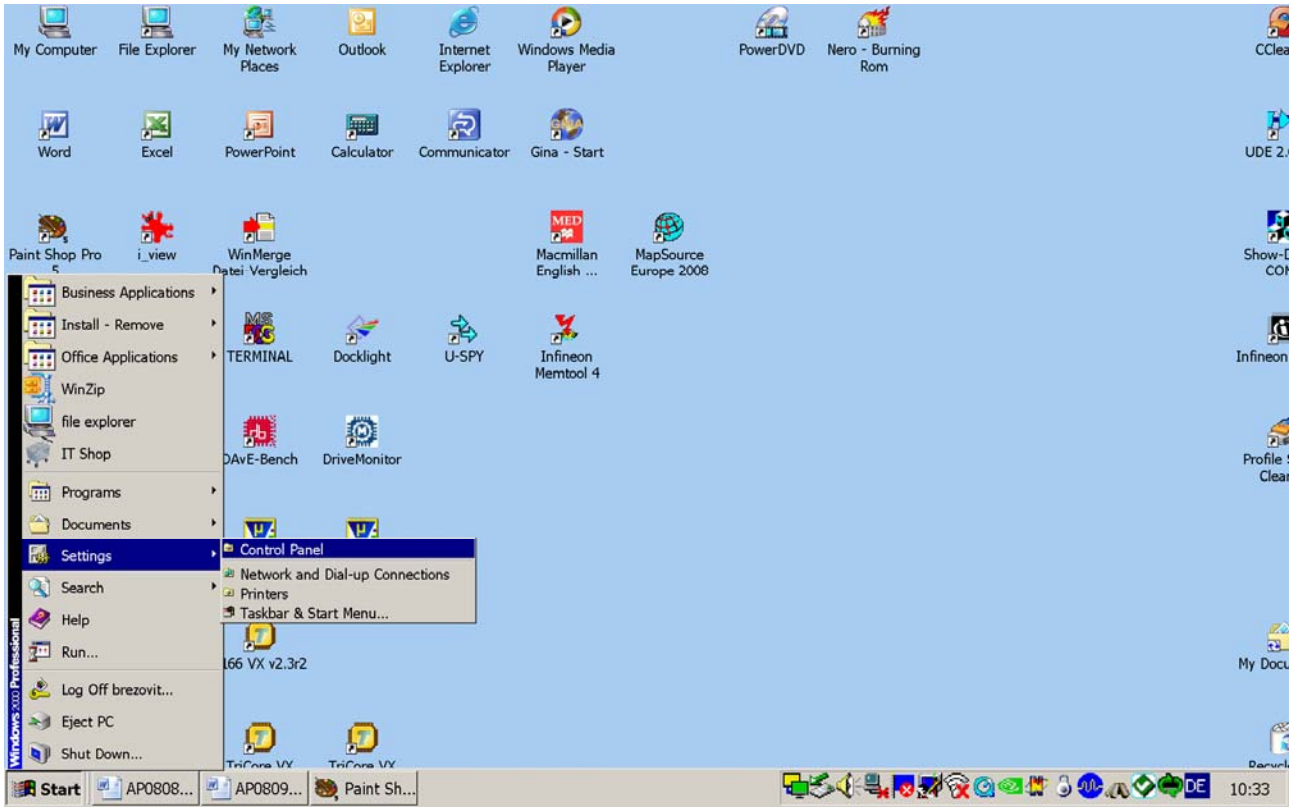

Peripherals
CAN x 2
CAN
LIN
LEDs
UART
Mode Control
JTAG
XC878CM
FTDI
USB to UART
USB to JTAG
control
USB

Note:
A USB driver is installed the first time while connecting the XC878 Easy Kit via USB to your host computer.

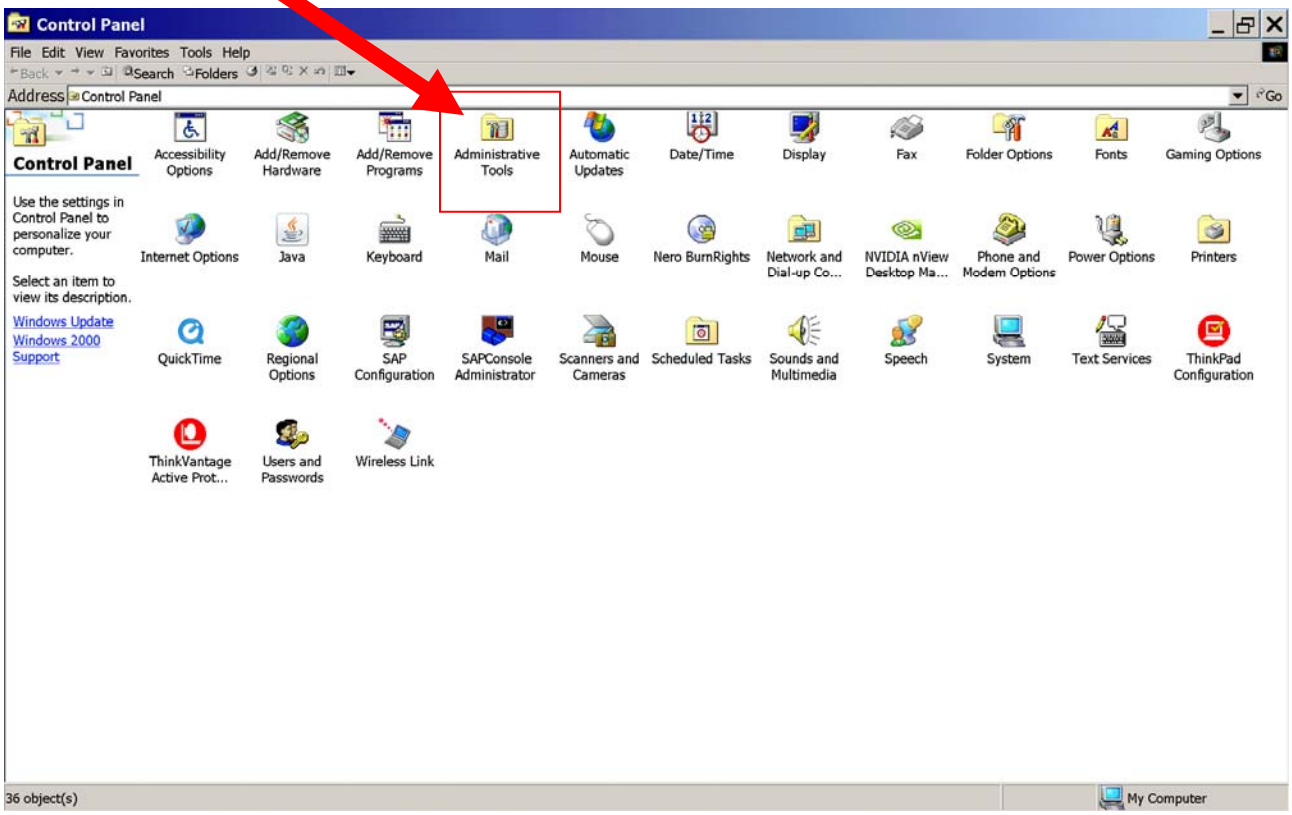
Note:
A default virtual COM Port is generated.

Using a Windows 2000 operating system, we are now going to search for the virtual COM Port which was generated after connecting our XC878 Easy Kit:

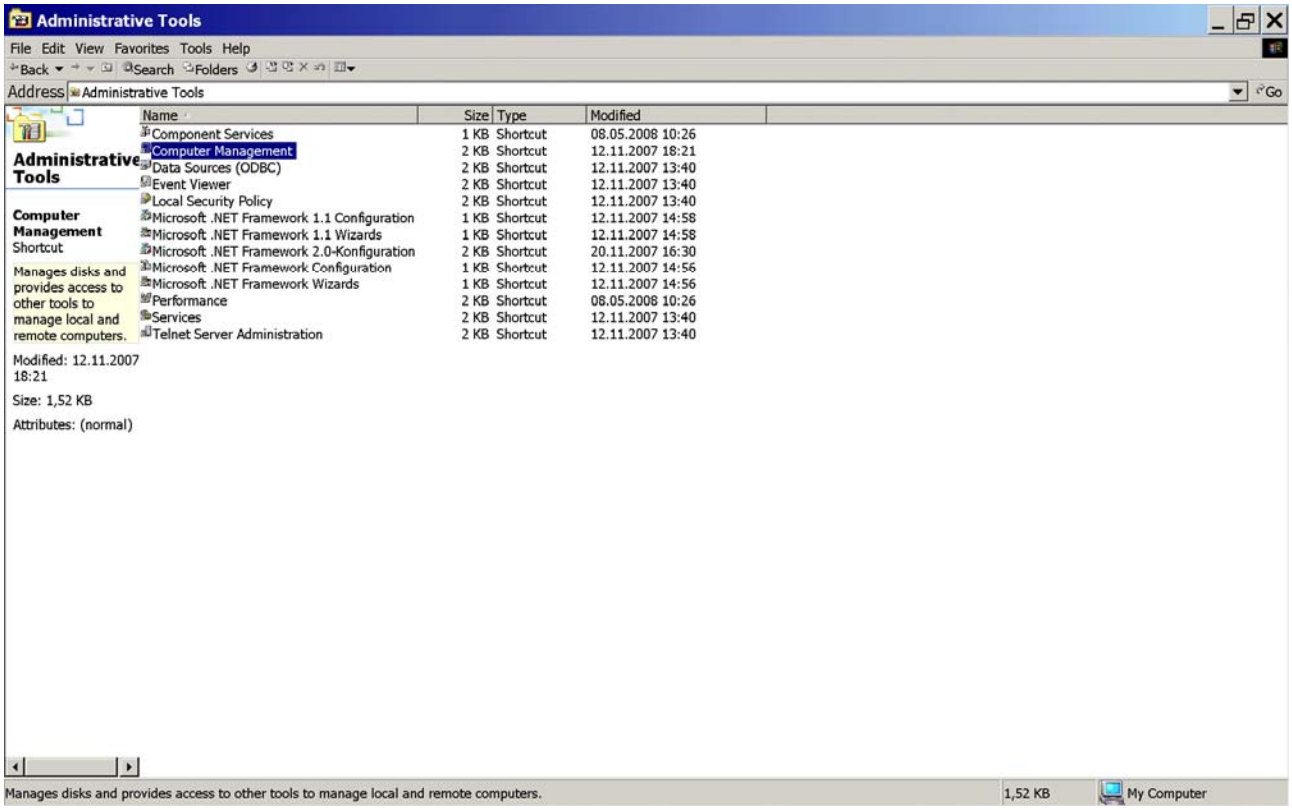
Start – Settings – Control Panel



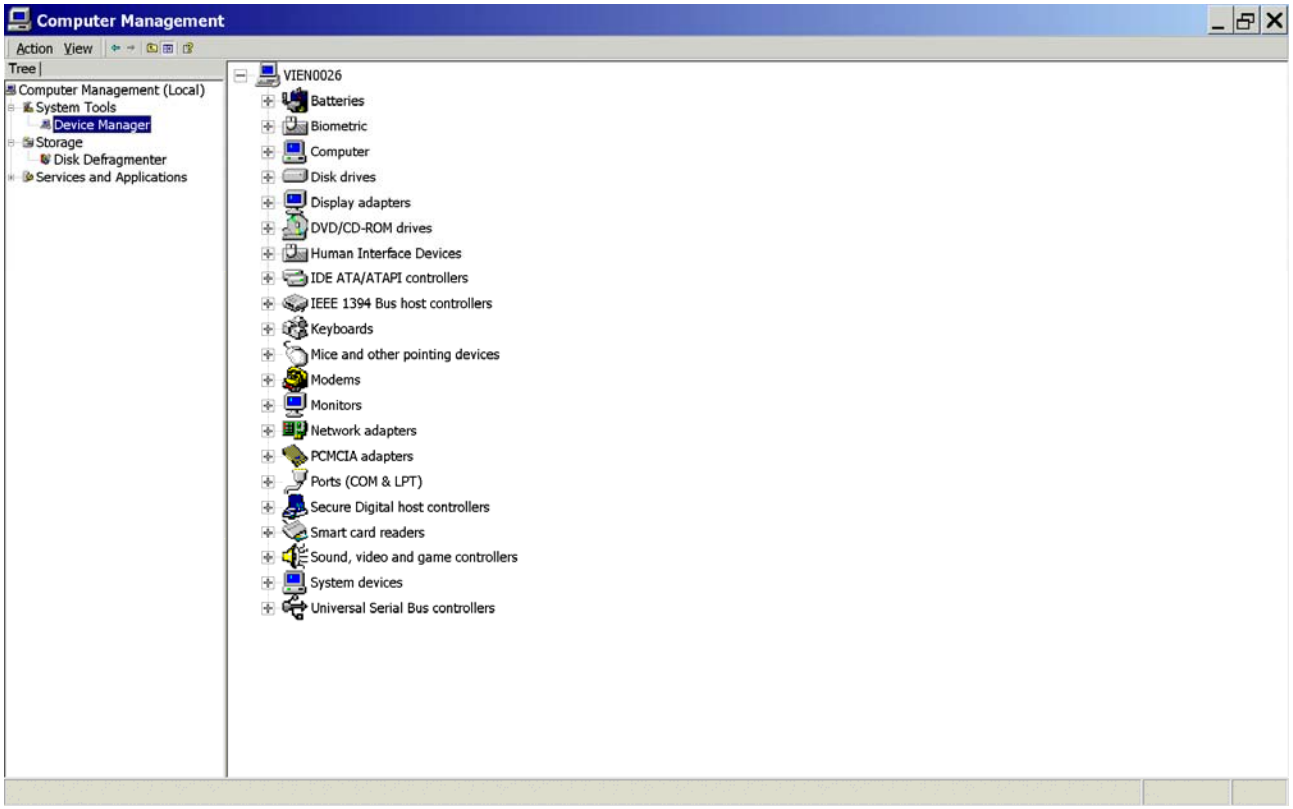
Double click: Administrative Tools



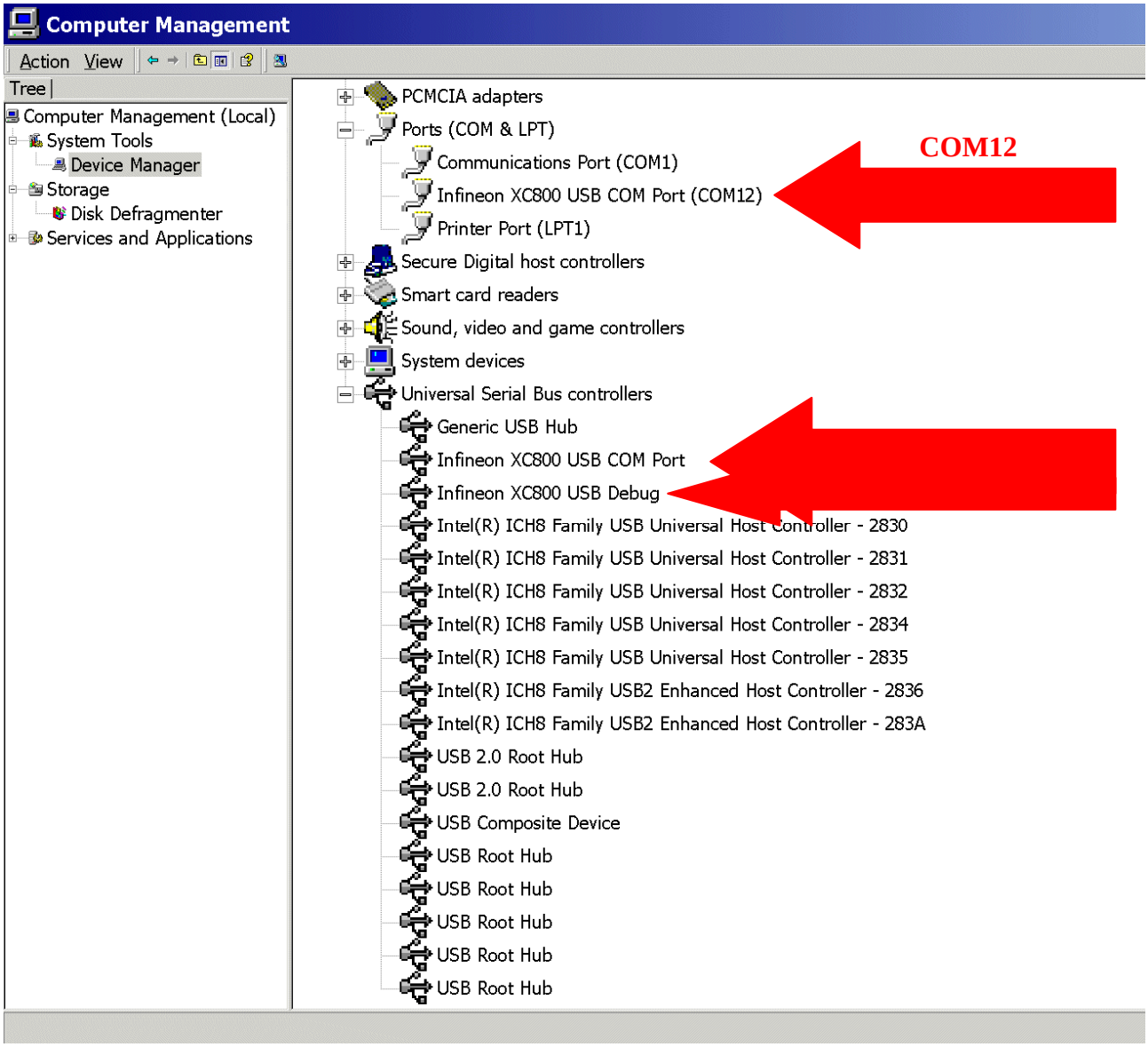
Double click: Computer Management



Click: Device Manager



Expand: Ports (COM & LPT):
Expand: Universal Serial Bus controllers:



Note:
As we can see:
our virtual COM Port for UART/RS232 communication with the XC878 Easy Kit via USB is COM12!

2.) DAvE – Installation for XC878 microcontrollers:

Install DAvE (mothersystem):

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

Title	Date	Version	Size
Tool Package			
DAvE - Mothersystem (DAvE_Mothersystem_v2_2r1.zip)	14 Dec 2009	V2.2	8.8 MB
DAvE - Mothersystem (setup.exe)	14 Dec 2009	V2.2	8.9 MB

and execute setup.exe to install DAvE .

Note:
Abort the installation of Acrobat Reader.

Install the XC878 microcontroller support/update (XC878 DIP file):

1.)

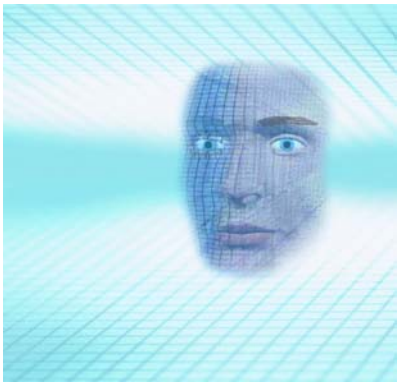
Download DAvE_XC878CLM_v2_1.zip (- or any higher version !!!)
- the DAvE-update-file (.DIP) for the required microcontroller
@ http://www.infineon.com/DAvE

Title	Date	Version	Size
Development Tools			
XC864 DIP file for DAvE (Microcontroller Configuration Tool)-latest version (DAvE_XC864_v1_2.zip)	14 Dec 2009	v1.2	5.6 MB
XC878 DIP file for DAvE (Microcontroller Configuration Tool)-latest version (DAvE_XC878CLM_v2_1.zip)	14 Dec 2009	v2.1	9.4 MB
[illegible] (DAvE_XC866_v2_2.zip)	[illegible]	[illegible]	[illegible]
XC878CLM DIP file for DAvE (Microcontroller Configuration Tool) (XC878CLM.zip)	08 Jul 2008	v1.1	9.1 MB
XC888CLM DIP file for DAvE (Microcontroller Configuration Tool)-latest version (DAvE_XC888CLM_v1_6.zip)	14 Dec 2009	v1.6	7.6 MB
XC886CLM DIP file for DAvE (Microcontroller Configuration Tool)-latest version (DAvE_XC886CLM_v1_8.zip)	14 Dec 2009	v1.8	7.6 MB
XC866 DIP file for DAvE (Microcontroller Configuration Tool) (XC866_v2.0.zip)	08 Feb 2008	v2.0	4.9 MB
XC888CLM DIP file for DAvE (Microcontroller Configuration Tool), V1.3 (XC888CLM_v1.3.zip)	14 Jan 2008	V1.3	7.6 MB
XC866 DIP file for DAvE (Microcontroller Configuration Tool) (XC866_v1.9.zip)	14 Jan 2008	V1.9	4.9 MB
XC886CLM DIP file for DAvE (Microcontroller Configuration Tool), V1.5 (XC886CLM_v1.5.zip)	14 Jan 2008	V1.5	7.5 MB
XC888CLM DIP file for DAvE (Microcontroller Configuration Tool), V1.1 (DAvE_XC888CLM_v1.1.zip)	01 Mar 2007		7.5 MB
XC886CLM DIP file for DAvE (Microcontroller Configuration Tool), V1.3 (XC886CLM_v1.3.zip)	22 Mar 2007		7.5 MB
DAvE XC888 RELEASE NOTES (DAvE_XC888_RELEASE_NOTES.pdf)	24 May 2007		351 KB
XC886CLM DIP file for DAvE (Microcontroller Configuration Tool), V1.1 (XC886CLM_v1.1.zip)	01 Sep 2006		6.4 MB

Unzip the zip-file “XC878CLM_v2_1.zip” and save “ XC878CLM_v2.1.dip “
@ e.g. D:\DAvE\XC878CLM_v2.1.dip.

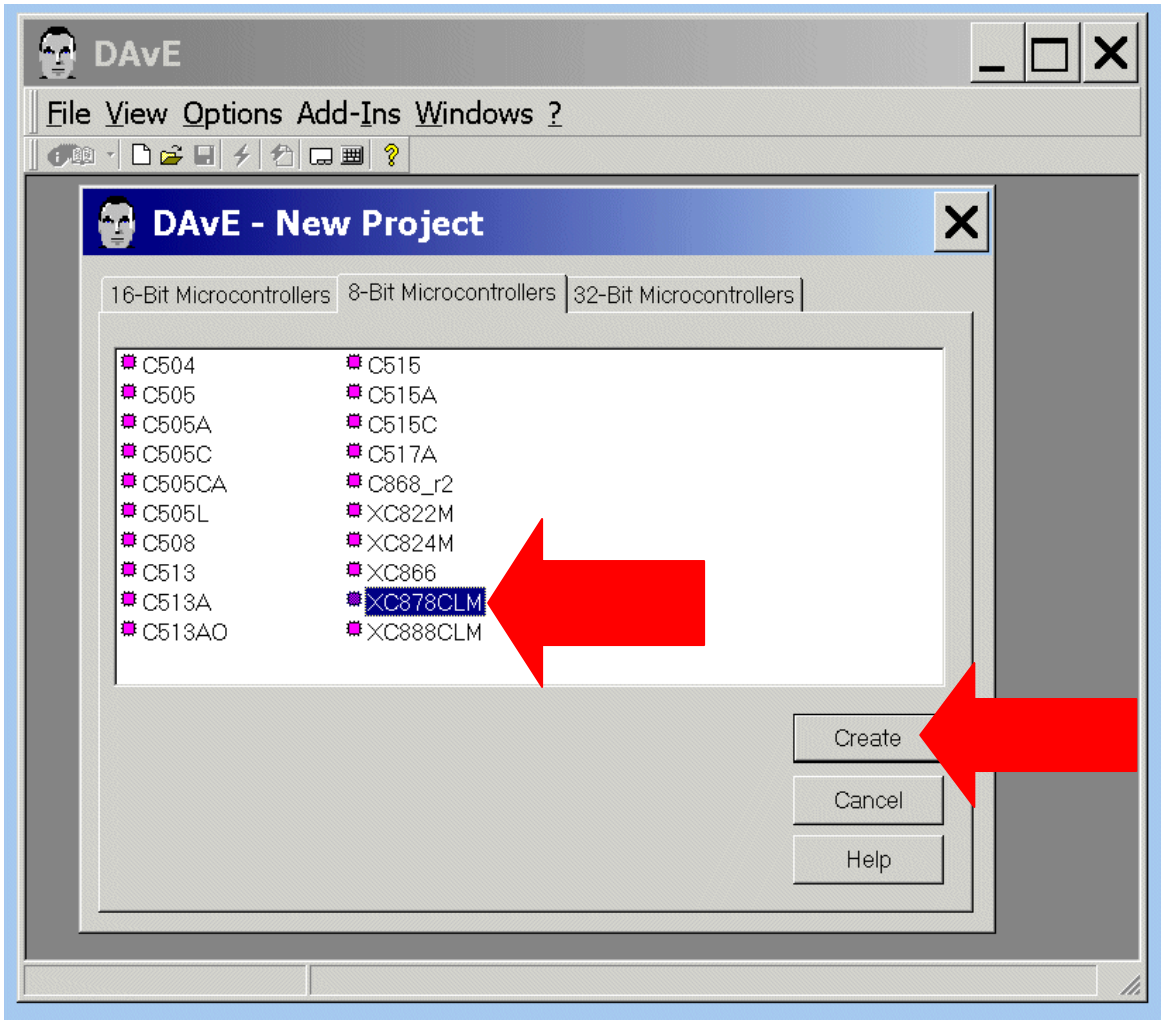
2.) Start DAvE - (click )
3.) View Setup Wizard Default: • Installation Forward> Select: • I want to install products from the DAvE's web site Forward> Select: D:\DAvE Forward> Select: Available Products click ✔ XC878CLM Forward> Install End
4.) DAvE is now ready to generate code for the XC878 microcontroller.

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



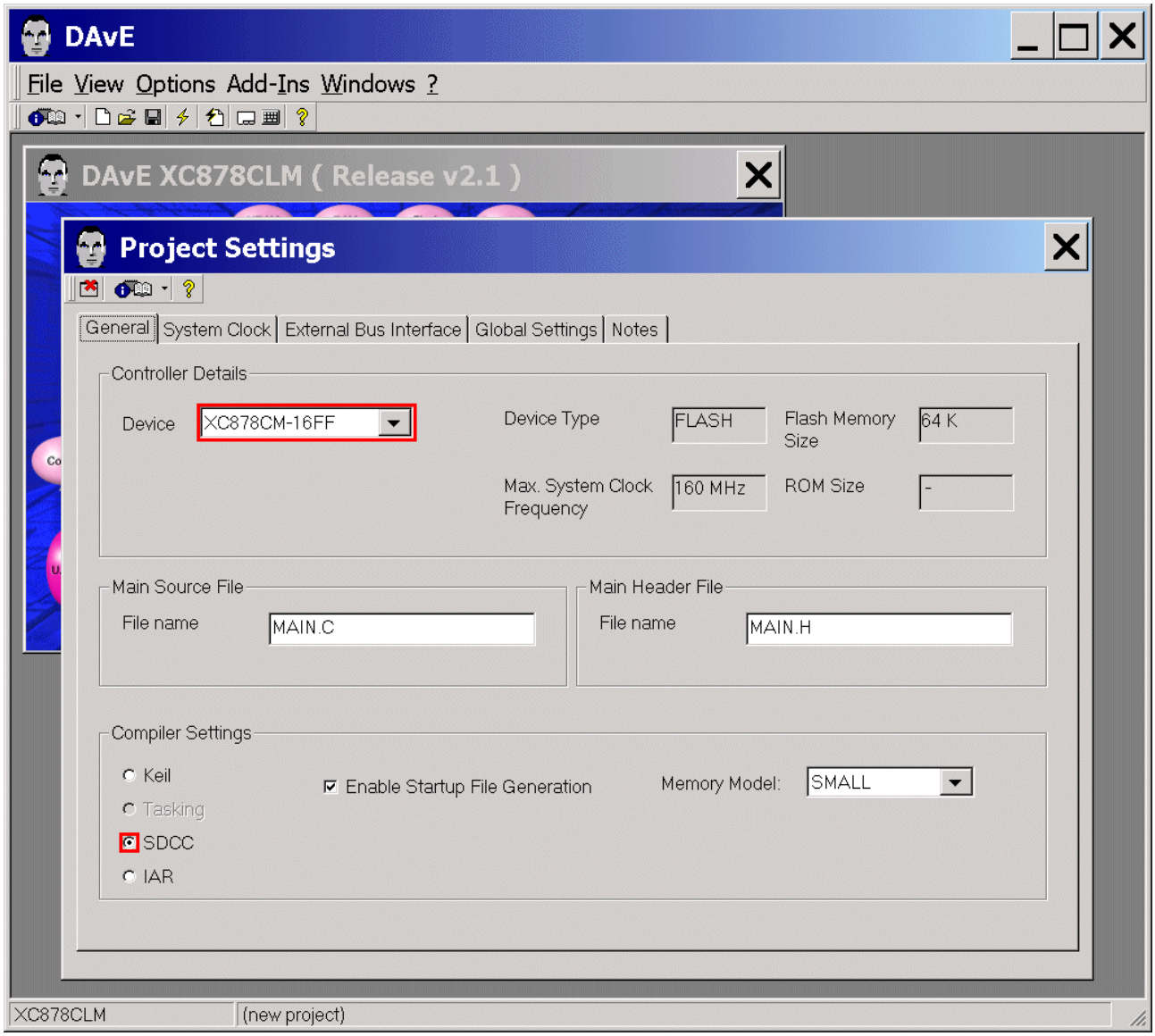
Start the program generator DAvE and select the XC878 microcontroller:

File
New
8-Bit Microcontrollers
select XC878CLM
Create

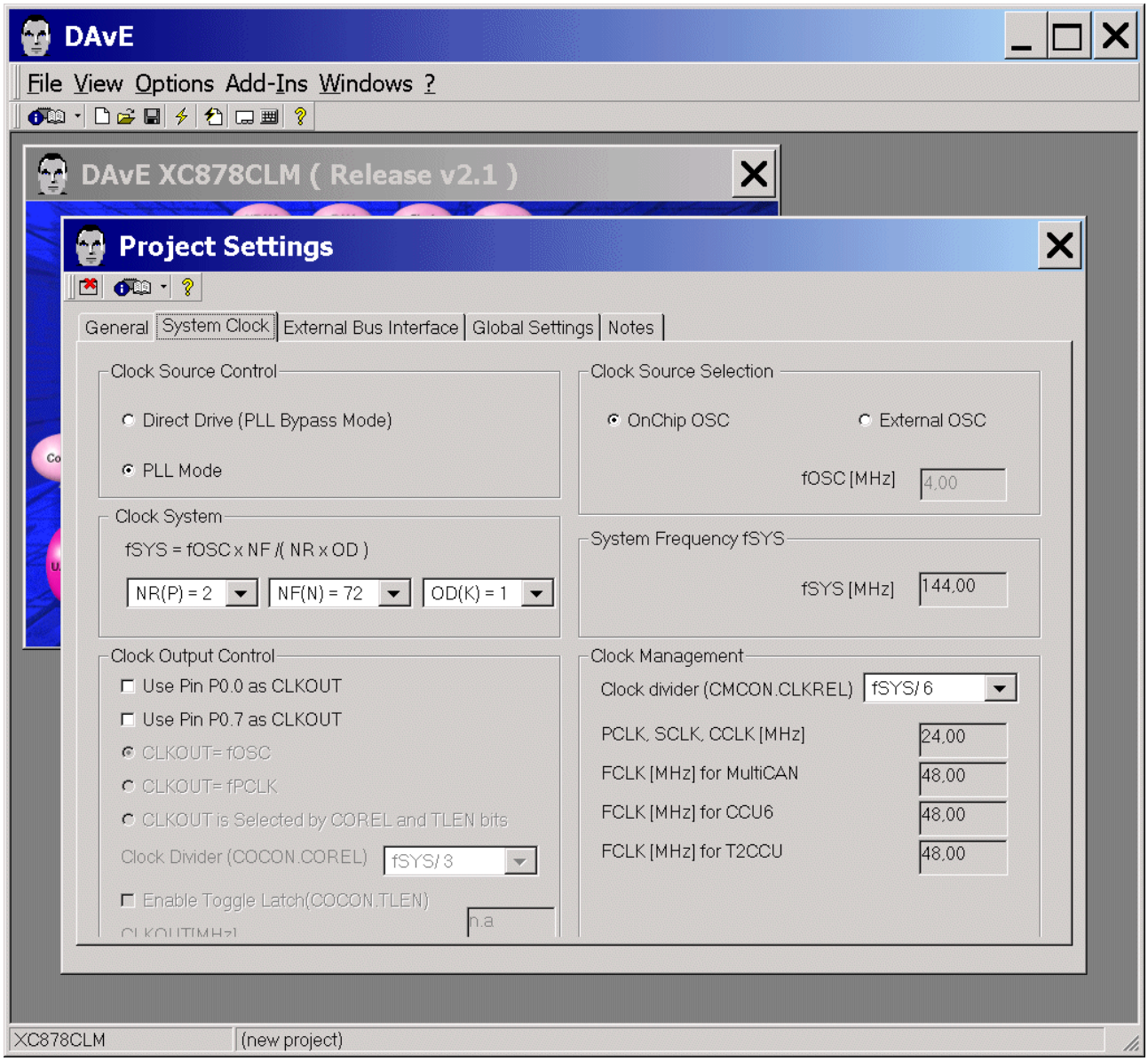


Choose the Project Settings as you can see in the following screenshots:

General: Controller Details: Device: check/select XC878CM-16FF
General: Compiler Settings: For DAvE-Bench check/choose ⦿ SDCC (DAvE Bench)

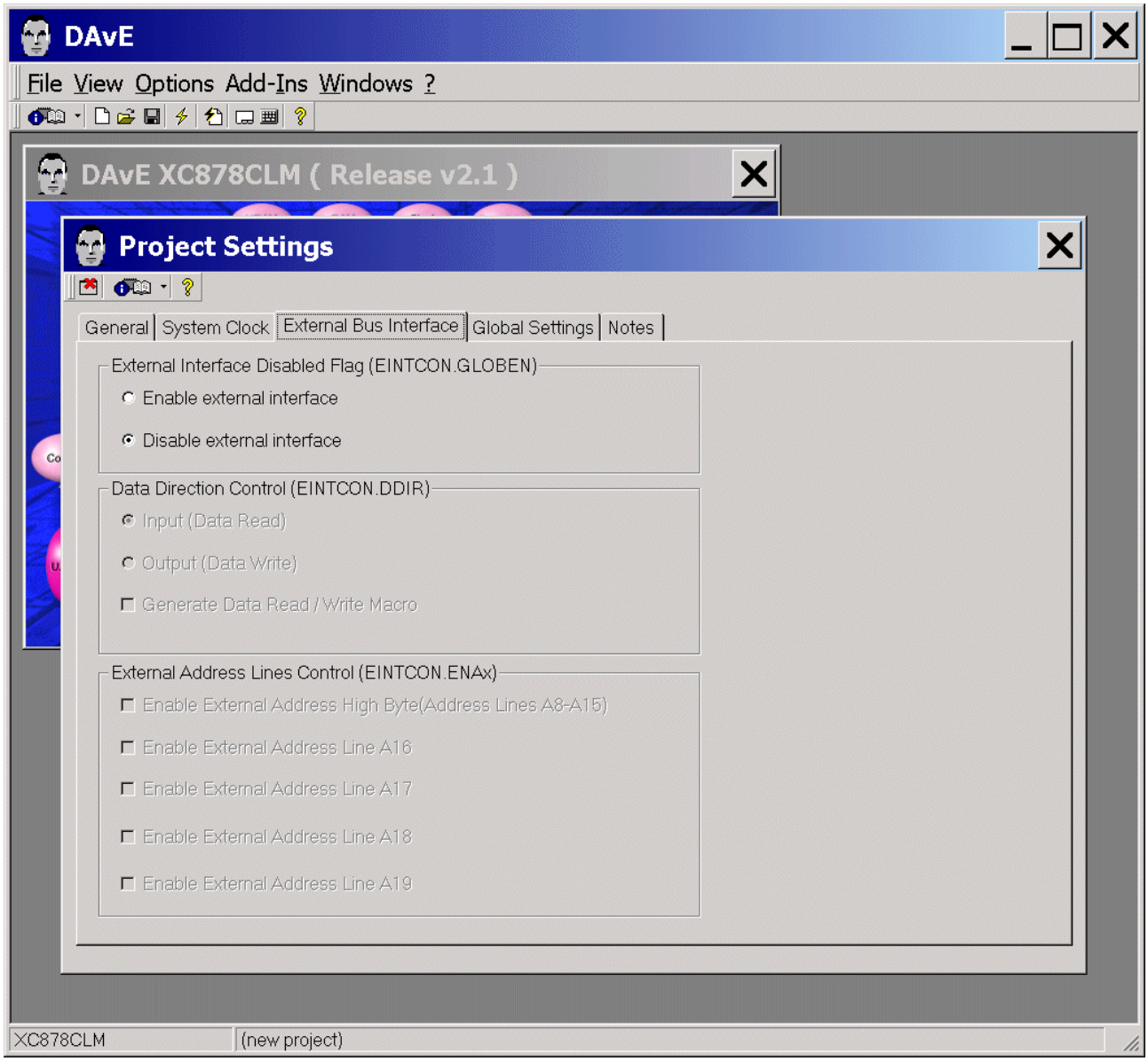


System Clock: (do nothing)

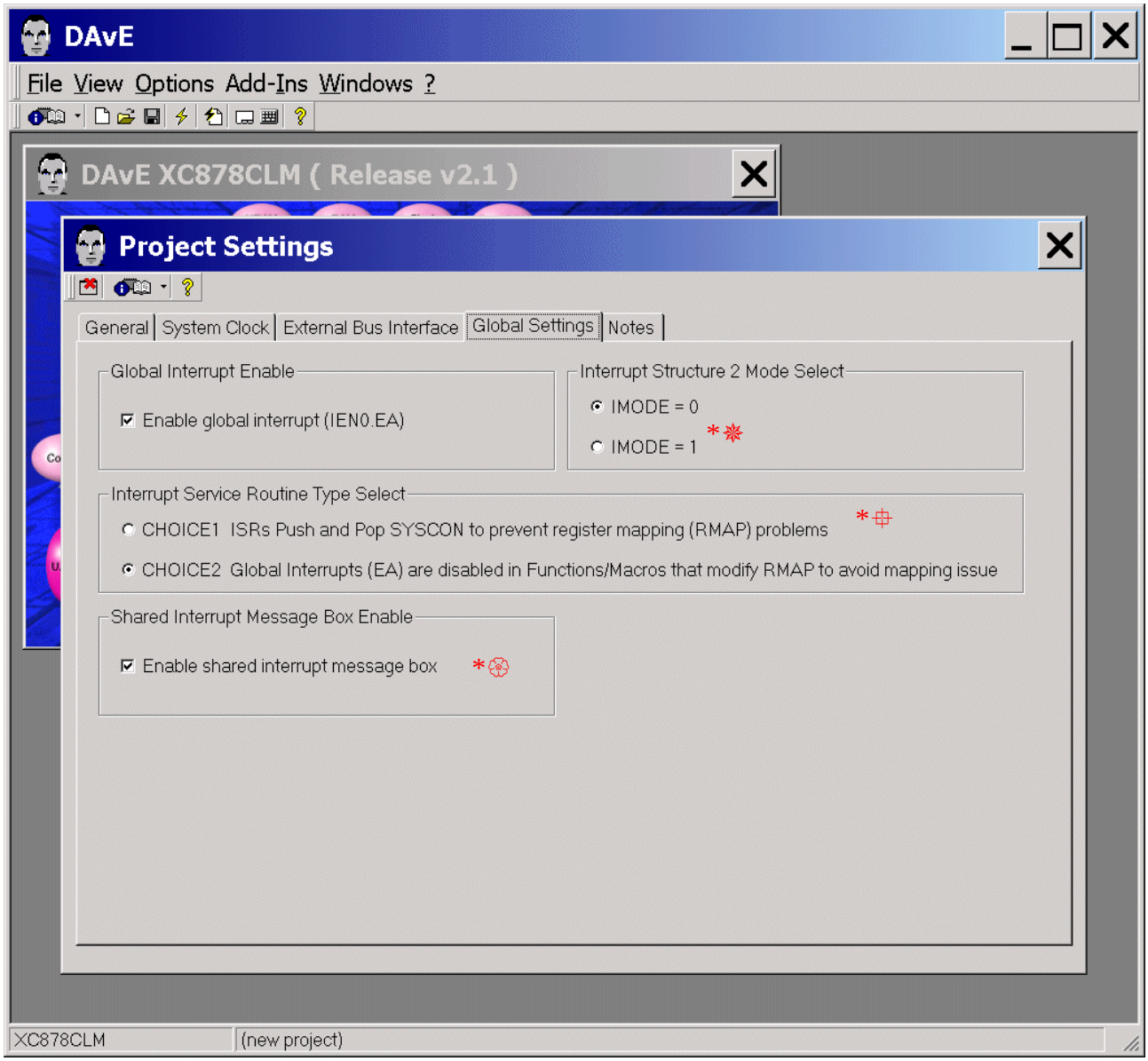


Note:
CPU clock is 24 MHz.

External Bus Interface: (do not change configuration)



Global Settings: (do not change configuration)



Note (Source: DAvE): *⌖ =

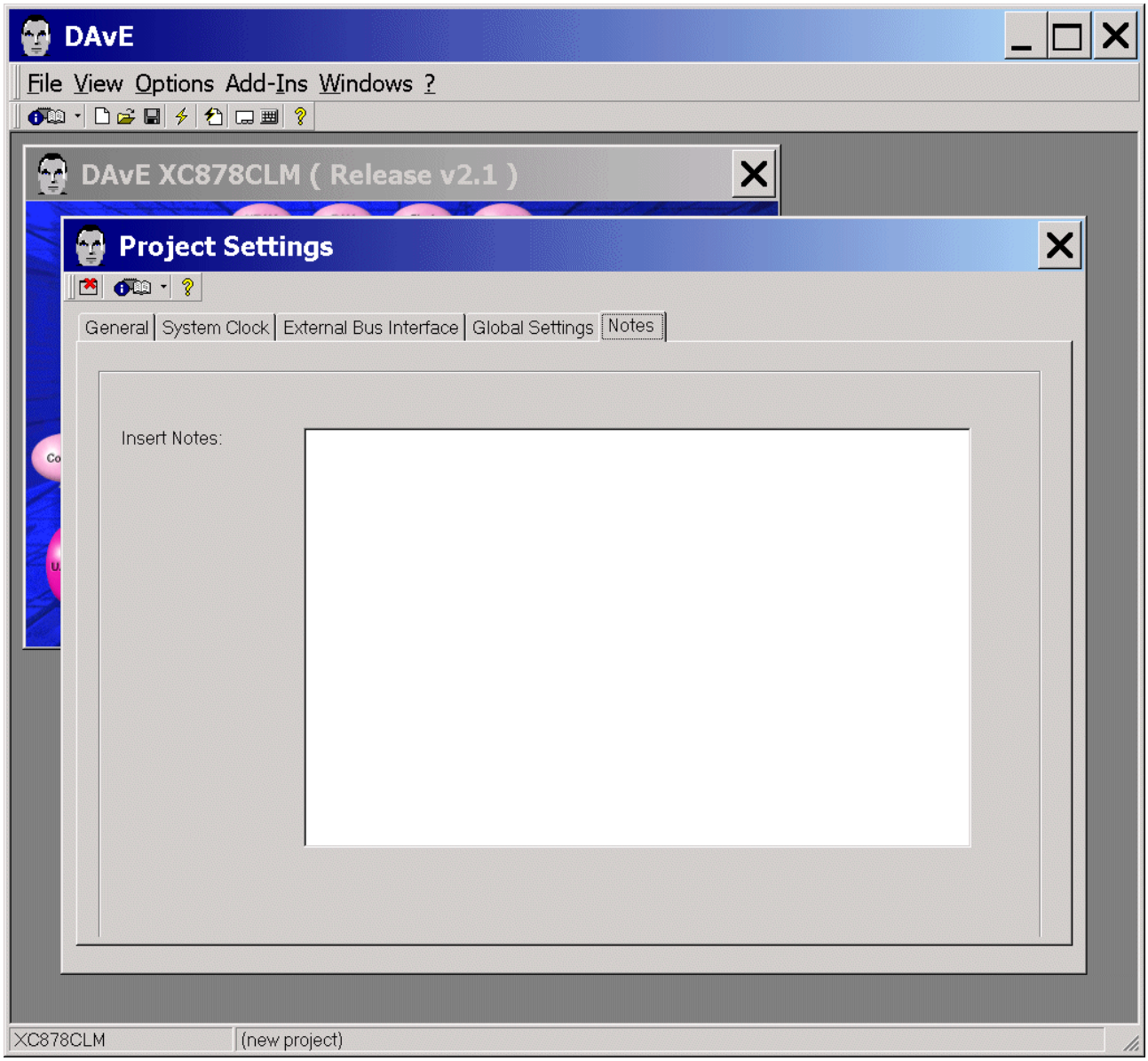
// You have two choices for interrupt type select in Project Settings Page
// under Global Settings Section.
// If you select CHOICE 1 then ISR will be generated with push and pop.
// If you select CHOICE 2 then ISR will be generated without push and pop.
// Default choice is CHOICE 2.
// Current selection is CHOICE 2

Note:

*❋ = There is a slightly different behavior between MODE=0 and MODE=1 in setting/clearing the pending interrupt request bit.

*❀ = If an interrupt node is shared with another interrupt node, the ISR code will be generated in the SHARED_INT.C file.

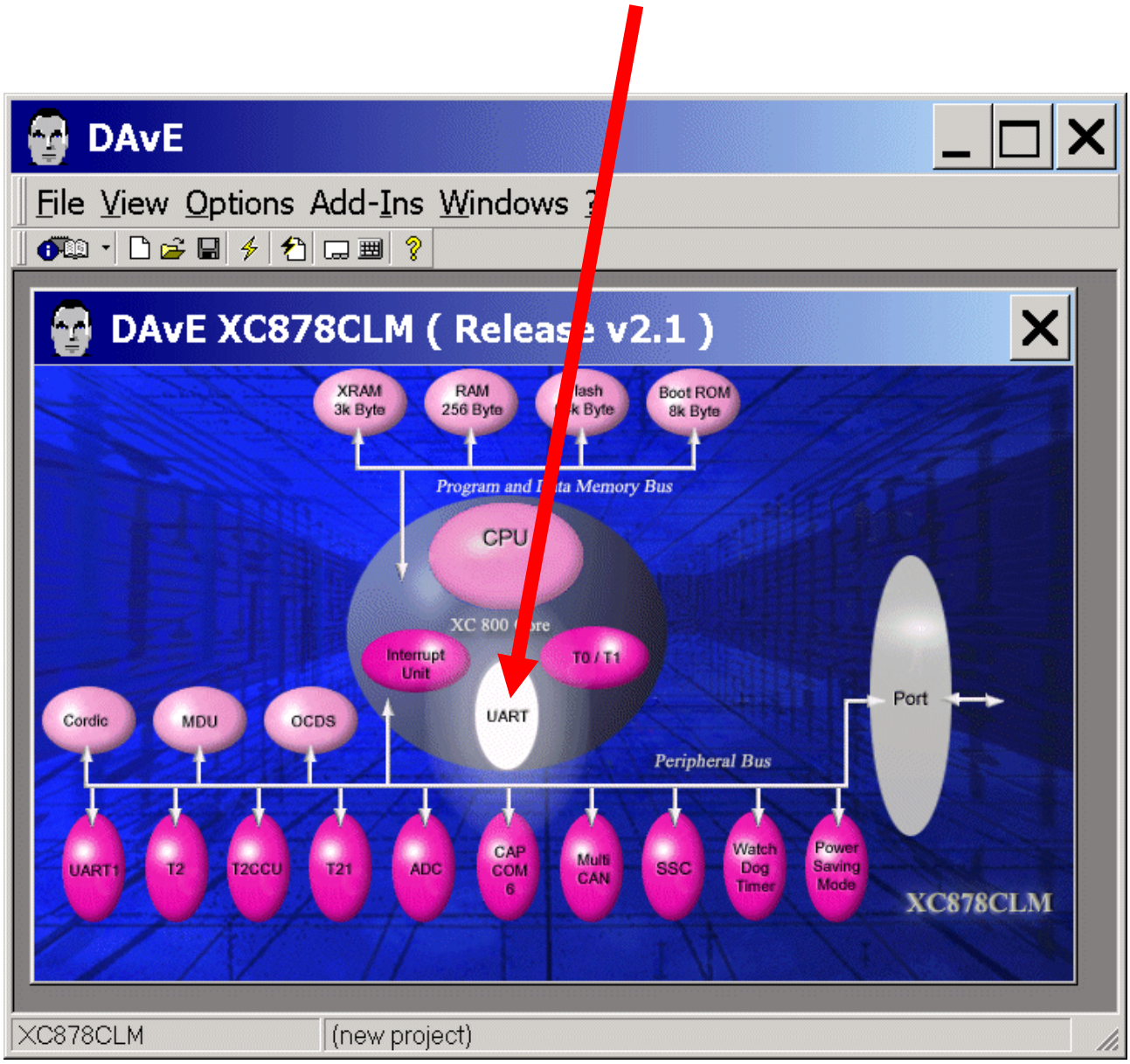
Notes: If you wish, you can insert your comments here.



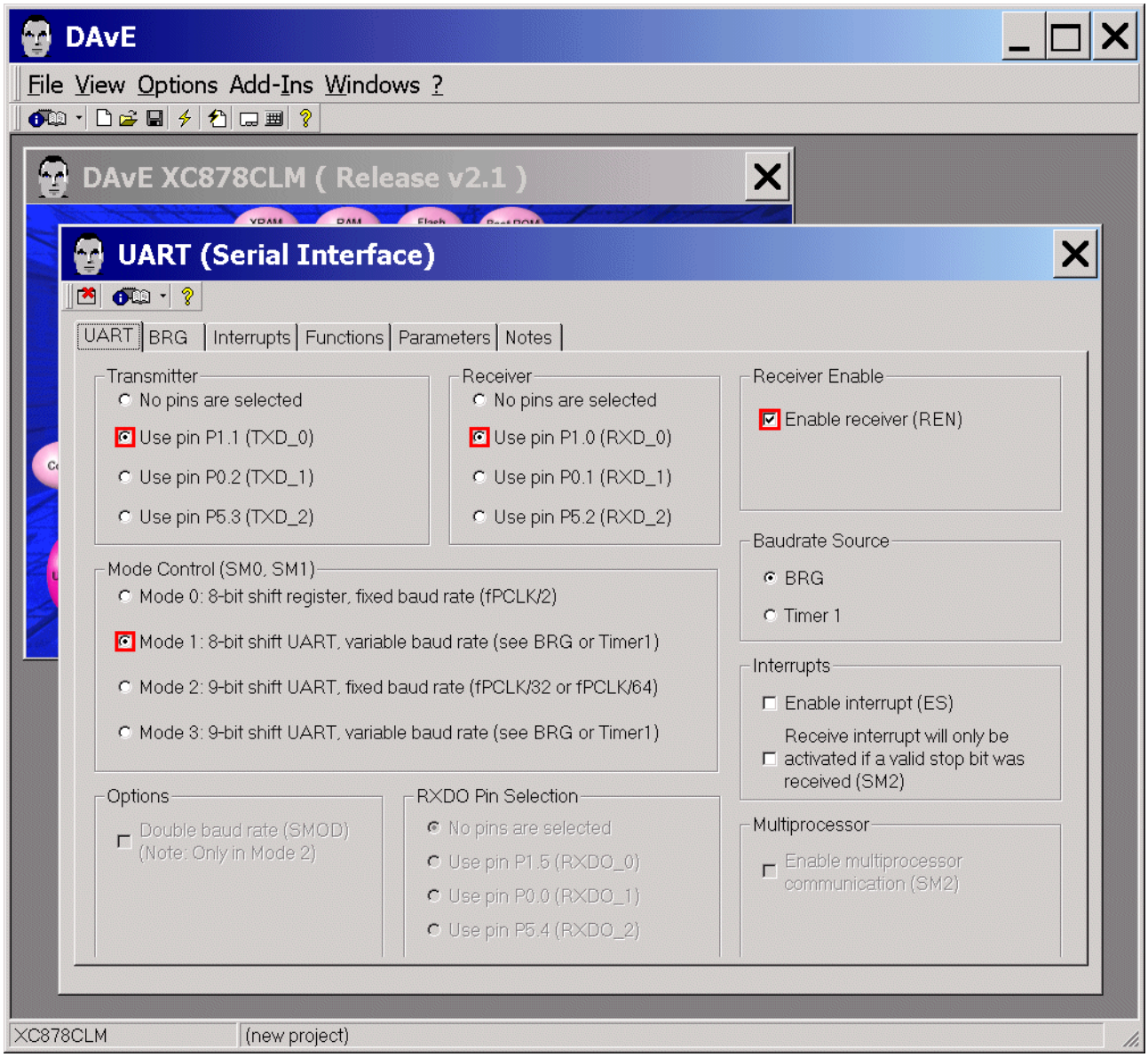
Exit and Save this dialog now by clicking the close button:

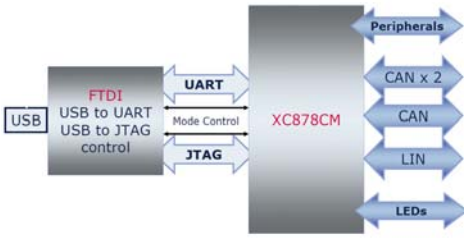
Configuration of the UART:

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



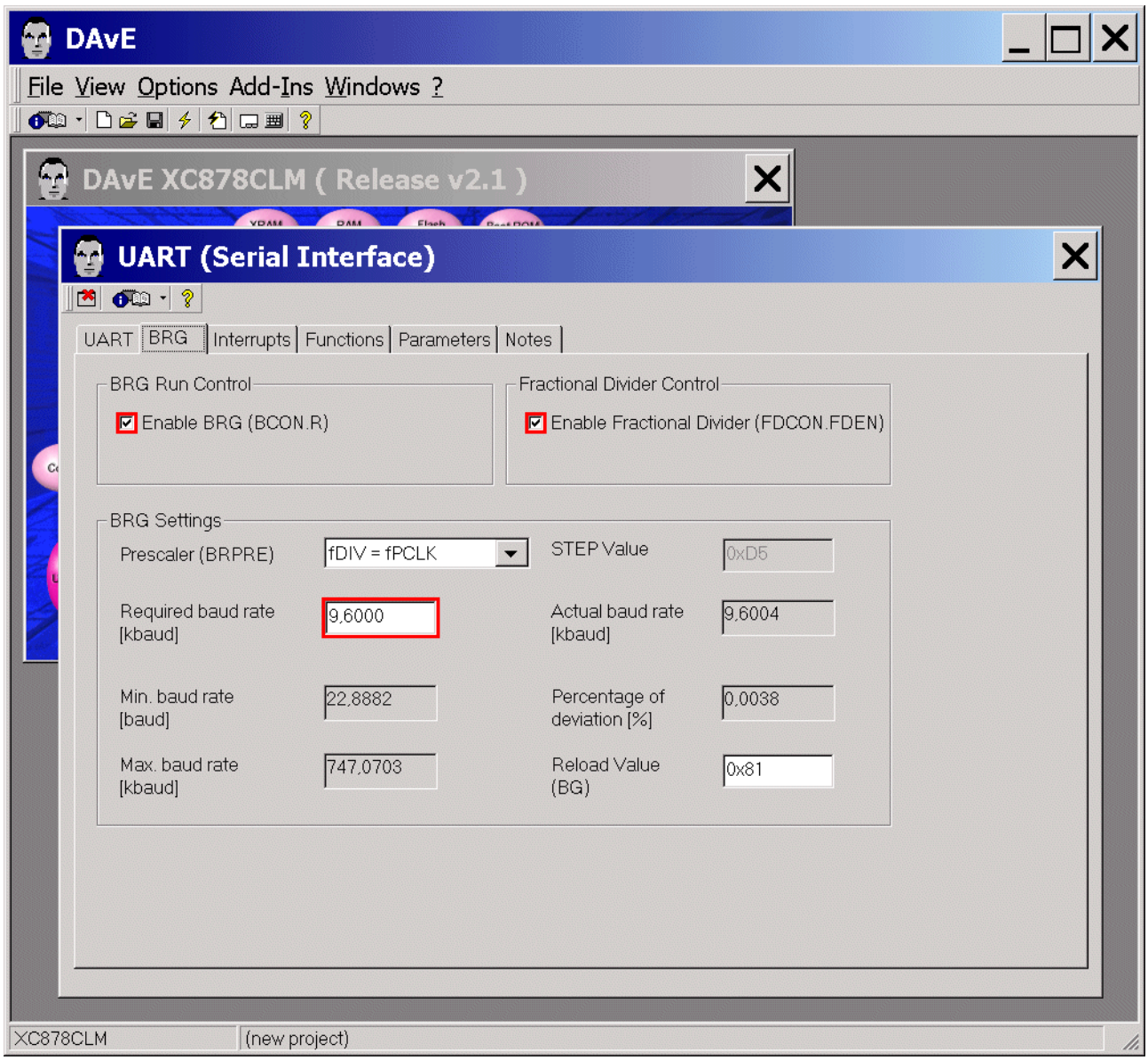
UART: Transmitter: click ⦿ Use pin P1.1 (TXD_0)
UART: Receiver: click ⦿ Use pin P1.0 (RXD_0)
UART: Receiver Enable: tick ✓ Enable receiver (REN)
UART: Mode Control: click ⦿ Mode 1: 8-bit shift UART, variable baud rate (see BRG or Timer1)





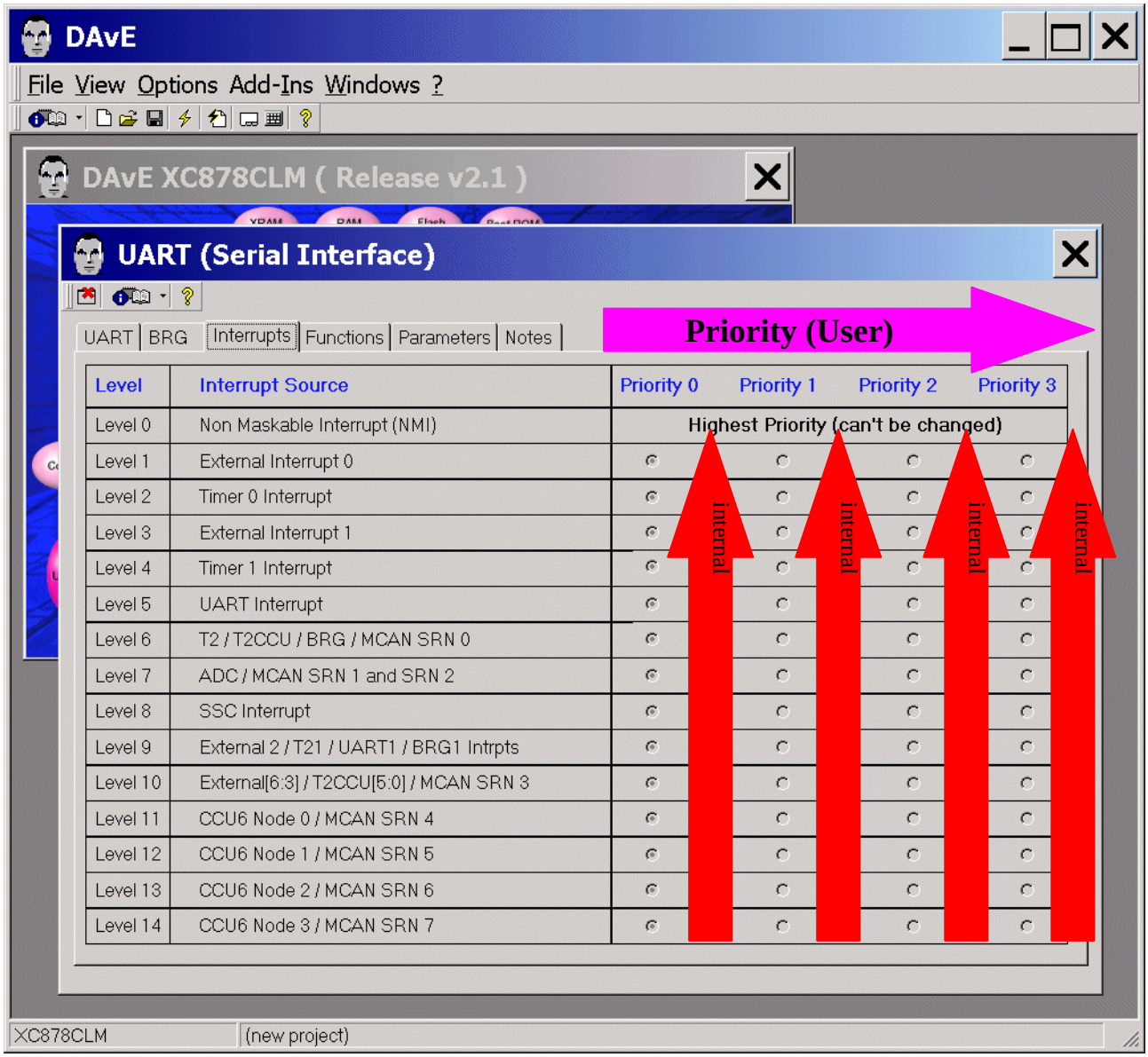
Note:
The RS232 serial interface (UART pins P1.0 and P1.1) is available via the USB port as virtual COM port (e.g. COM12) which converts the TTL-UART-signals to USB-signals.

BRG: BRG Run Control: check/tick ✓ Enable BRG
BRG: Fractional Divider Control: tick ✓ Enable Fractional Divider
BRG: BRG Settings: Required baud rate [kbaud] insert 9,600 <ENTER>



Note:
Validate each alphanumeric entry by pressing ENTER.

Interrupts: (do nothing)



Note:
For the serial communication with a terminal program (e.g. U-SPY) running on your host computer the (printf) / printf_small() / printf_fast_f() - function is used. The “printf” function uses Software-Polling-Mode therefore we do not need to configure any interrupts.

Note:
(printf) / printf_small() / printf_fast_f():

Release Notes

SDCC Compiler Tool chain for XC800

Known Issues (Other than open bugs):

The built-in printf() provided in the package does not work with XC878. Alternatively, other print formats like printf_small(), printf_fast_f() are recommended to be used with XC878.

At the time this document was written, there was no printf (see above) for the XC878 microcontroller. Therefore we are going to use printf_small().

Additional information: printf(), printf_small(), printf_fast_f():

SDCC Compiler User Guide

SDCC 2.9.0
$Date:: 2009-03-13 #$
$Revision: 5413 $

3.17.2 Stdclib functions (puts, printf, strcat etc.)

3.17.2.1 <stdio.h>

getchar(), putchar() As usual on embedded systems you have to provide your own `getchar()` and `putchar()` routines. SDCC does not know whether the system connects to a serial line with or without handshake, LCD, keyboard or other device. And whether a `lf` to `crlf` conversion within `putchar()` is intended. You'll find examples for serial routines f.e. in sdcc/device/lib. For the mcs51 this minimalistic polling `putchar()` routine might be a start:

```
void putchar (char c) {
    while (!TI)     /* assumes UART is initialized */
        ;
    TI = 0;
    SBUF = c;
}
```

printf(), printf_small(), printf_fast_f() (continuation):

printf() The default `printf()` implementation in `printf_large.c` does not support float (except on ds390), only <NO FLOAT> will be printed instead of the value. To enable floating point output, recompile it with the option *-DUSE_FLOATS=1* on the command line. Use *--model-large* for the mcs51 port, since this uses a lot of memory. To enable float support for the pic16 targets, see 4.6.8.

If you're short on code memory you might want to use `printf_small()` *instead* of `printf()`. For the mcs51 there additionally are assembly versions `printf_tiny()` (subset of printf using less than 270 bytes) and `printf_fast()` and `printf_fast_f()` (floating-point aware version of printf_fast) which should fit the requirements of many embedded systems (printf_fast() can be customized by unsetting #defines to *not* support long variables and field widths). Be sure to use only one of these printf options within a project.

Feature matrix of different *printf* options on mcs51.

mcs51	printf	printf USE_FLOATS=1	printf_small	printf_fast	printf_fast_f	printf_tiny
filename	printf_large.c	printf_large.c	printfl.c	printf_fast.c	printf_fast_f.c	printf_tiny.c
"Hello World" size small / large	1.7k / 2.4k	4.3k / 5.6k	1.2k / 1.8k	1.3k / 1.3k	1.9k / 1.9k	0.44k / 0.44k
code size small / large	1.4k / 2.0k	2.8k / 3.7k	0.45k / 0.47k (+ _ltoa)	1.2k / 1.2k	1.6k / 1.6k	0.26k / 0.26k
formats	cdi*o*psux	cdfi*o*psux	cd*o*sx	cdsux	cdfsux	cdsux
long (32 bit) support	x	x	x	x	x	-
byte arguments on stack	b	b	-	-	-	-
float format	-	%f	-	-	%f[9]	-
float formats %e %g	-	-	-	-	-	-
field width	x	x	-	x	x	-
string speed[10], small / large	1.52 / 2.59 ms	1.53 / 2.62 ms	0.92 / 0.93 ms	0.45 / 0.45 ms	0.46 / 0.46 ms	0.45 / 0.45 ms
int speed[11], small / large	3.01 / 3.61 ms	3.01 / 3.61 ms	3.51 / 18.13 ms	0.22 / 0.22 ms	0.23 / 0.23 ms	0.25 / 0.25 ms[12]
long speed[13], small / large	5.37 / 6.31 ms	5.37 / 6.31 ms	8.71 / 40.65 ms	0.40 / 0.40 ms	0.40 / 0.40 ms	-
float speed[14], small / large	-	7.49 / 22.47 ms	-	-	1.04 / 1.04 ms	-

Priority 0	Priority 1	Priority 2	Priority 3

Note: Interrupt Priorities (User):

Note (Source: Application Note AP08053):
There could be six interrupt priorities.
These priorities, with 6 beeing the highest, are as follows:

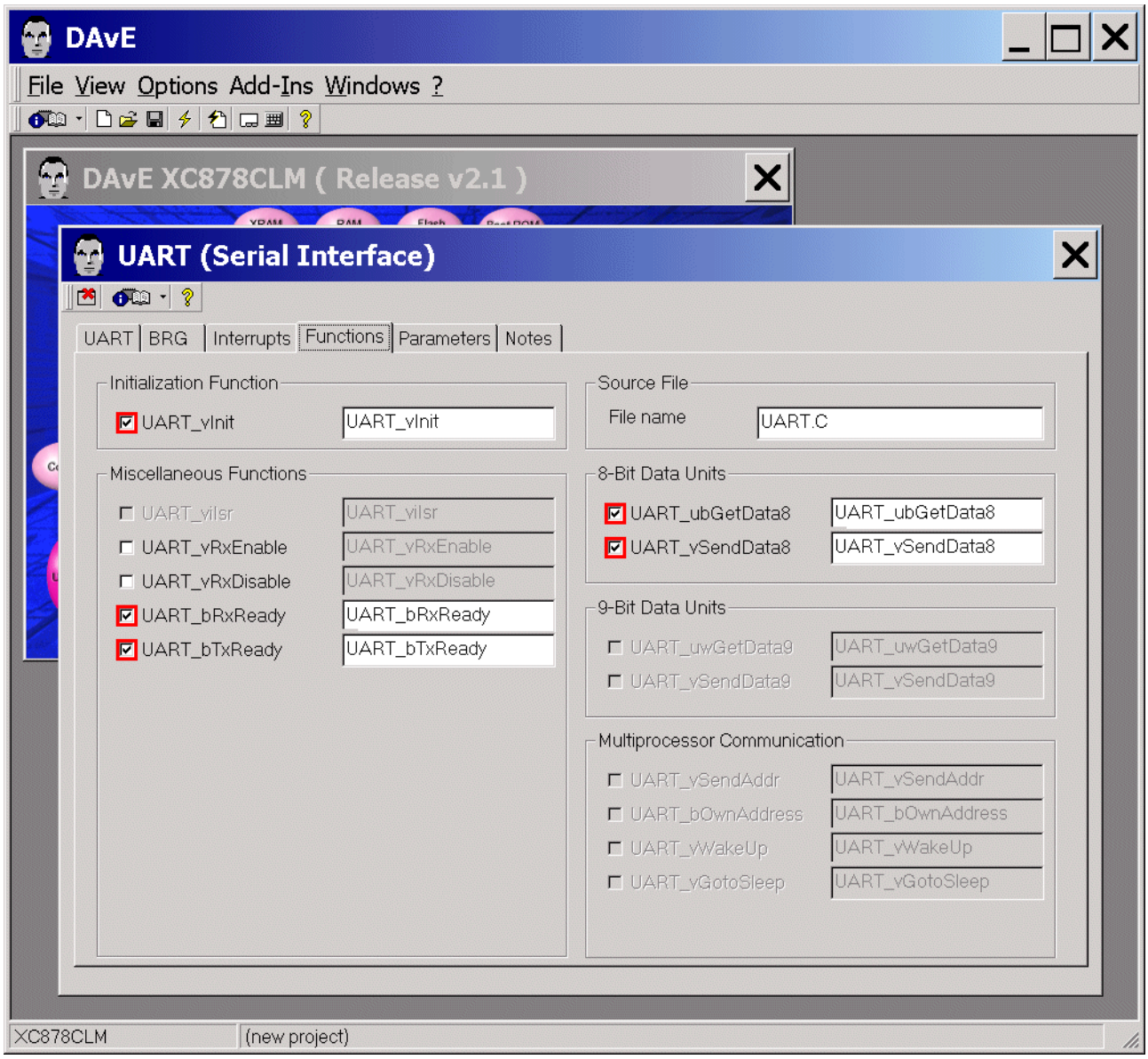
Interrupt Priority:	
6	NMI
5	Interrupt Priority 3
4	Interrupt Priority 2
3	Interrupt Priority 1
2	Interrupt Priority 0
1	Main

Main refers to routines that run prior to any interrupt and can be interrupted by any interrupt.
Each interrupt source can be programmed to any of the four interrupt priorities (0-3).
An interrupt that is currently being serviced can only be interrupted by a higher-priority interrupt, but not by another interrupt of the same or lower priority.
Hence, an interrupt of the highest priority cannot be interrupted by any other interrupt request.
In any case, the NMI always has the highest priority (above level 3) and its priority cannot be programmed.

Level
Level 0
Level 1
Level 2
Level 3
Level 4
Level 5
Level 6
Level 7
Level 8
Level 9
Level 10
Level 11
Level 12
Level 13
Level 14

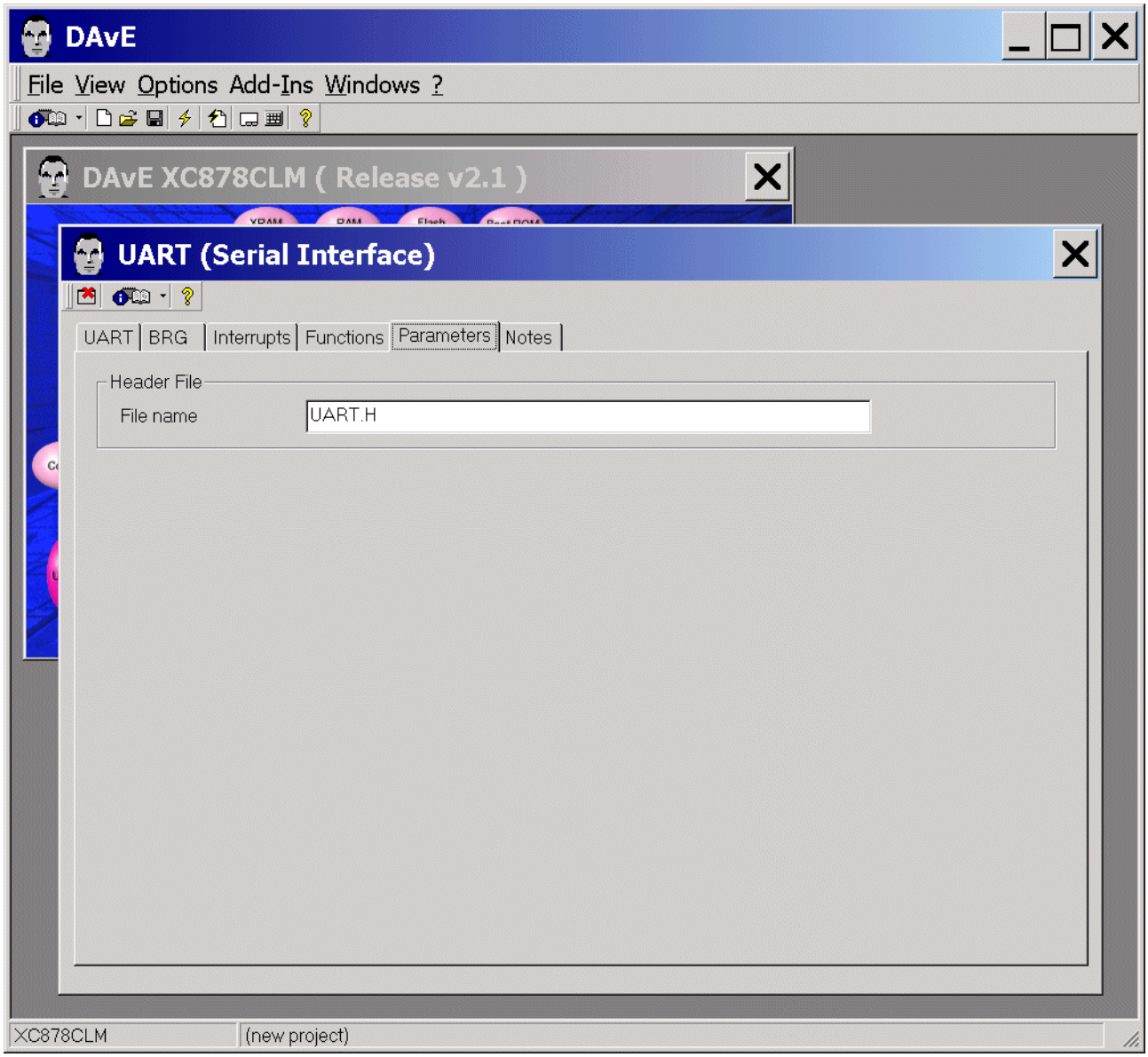
Note: Interrupt Priorities (**internal**, Source: User's Manual):
An interrupt that is currently being serviced can only be interrupted by a higher-priority interrupt, but not by another interrupt of the same or lower priority.
Hence, an interrupt of the highest priority cannot be interrupted by any other interrupt request.
If two or more requests of different priority levels are received simultaneously, the request with the highest priority is serviced first. If requests of the same priority are received simultaneously, an **internal** polling sequence determines which request is serviced first. Thus, within each priority level, there is a second priority structure determined by a polling sequence as shown in the User's Manual and above.

Functions: Initialization Function: tick ✓ UART_vInit
Functions: Miscellaneous Functions: tick ✓ UART_bRxReady
Functions: Miscellaneous Functions: tick ✓ UART_bTxReady
Functions: 8-Bit Data Units: tick ✓ UART_ubGetData8
Functions: 8-Bit Data Units: tick ✓ UART_vSendData8



Note:
You can change function names (e.g. UART_vInit) and file names (e.g. UART.C) anytime.

Parameters: (do nothing)

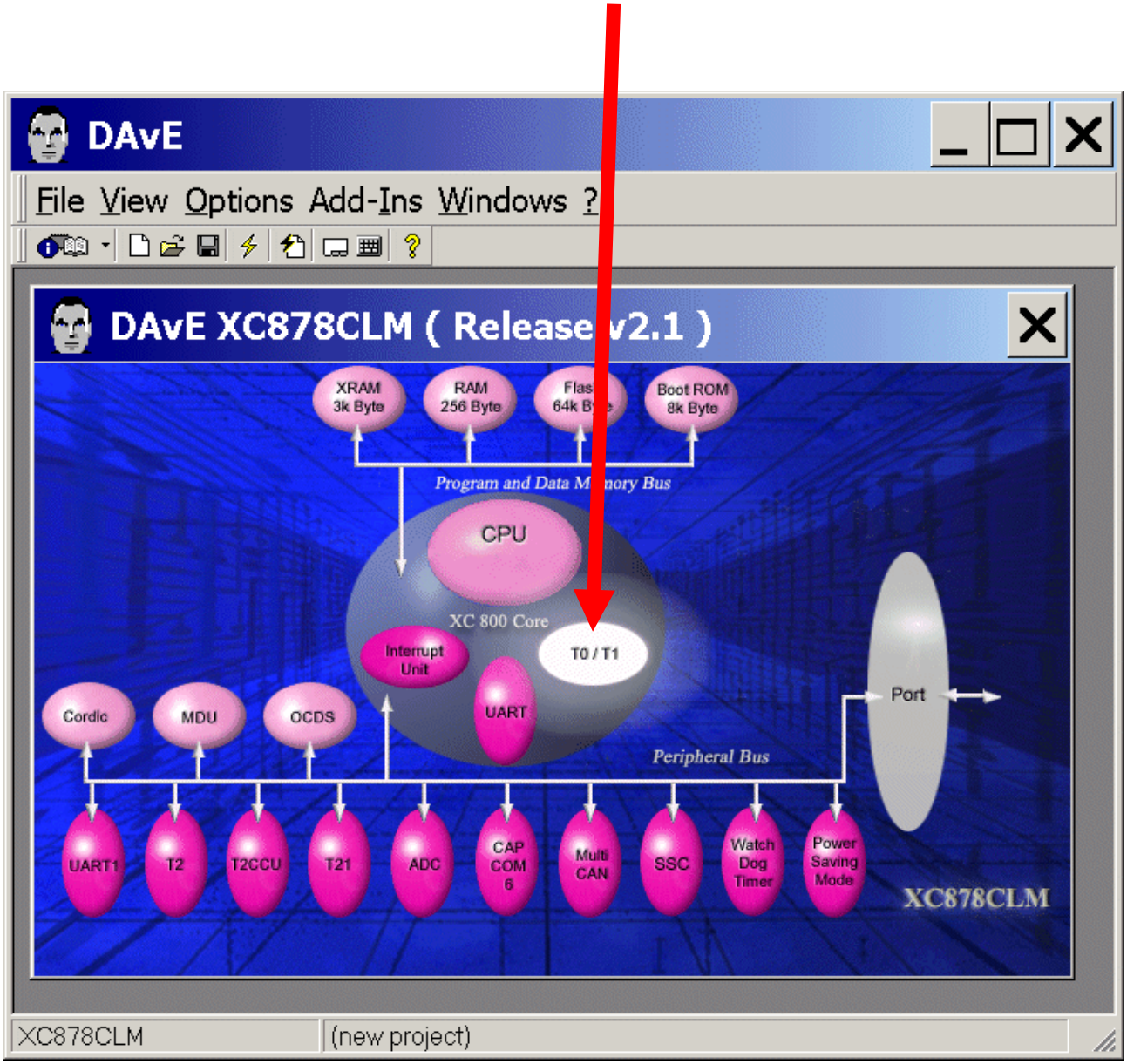


Notes: If you wish, you can insert your comments here.

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

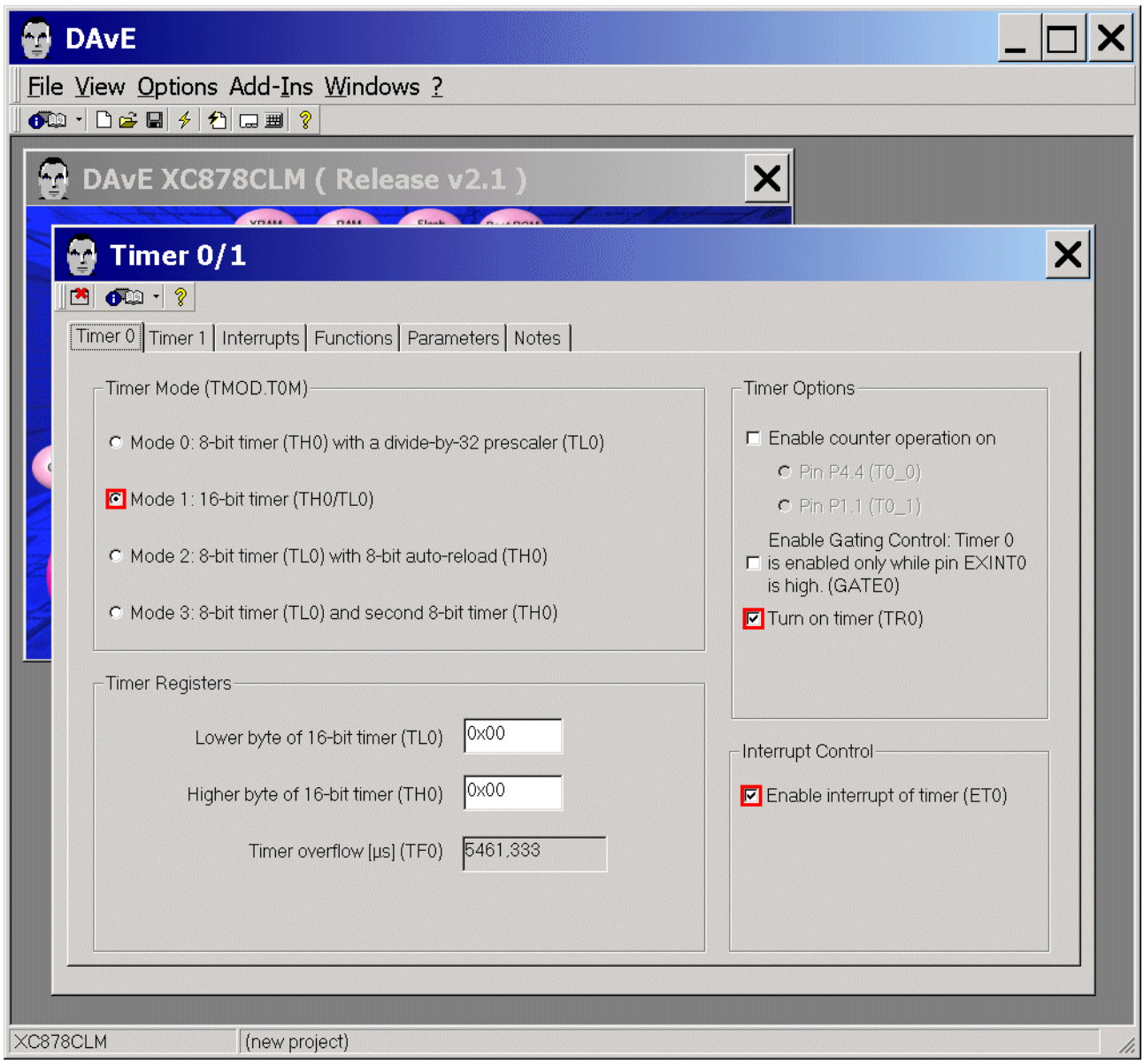
Configure Timer T0:

The configuration window/dialog can be opened by clicking the specific block/module (T0/T1).



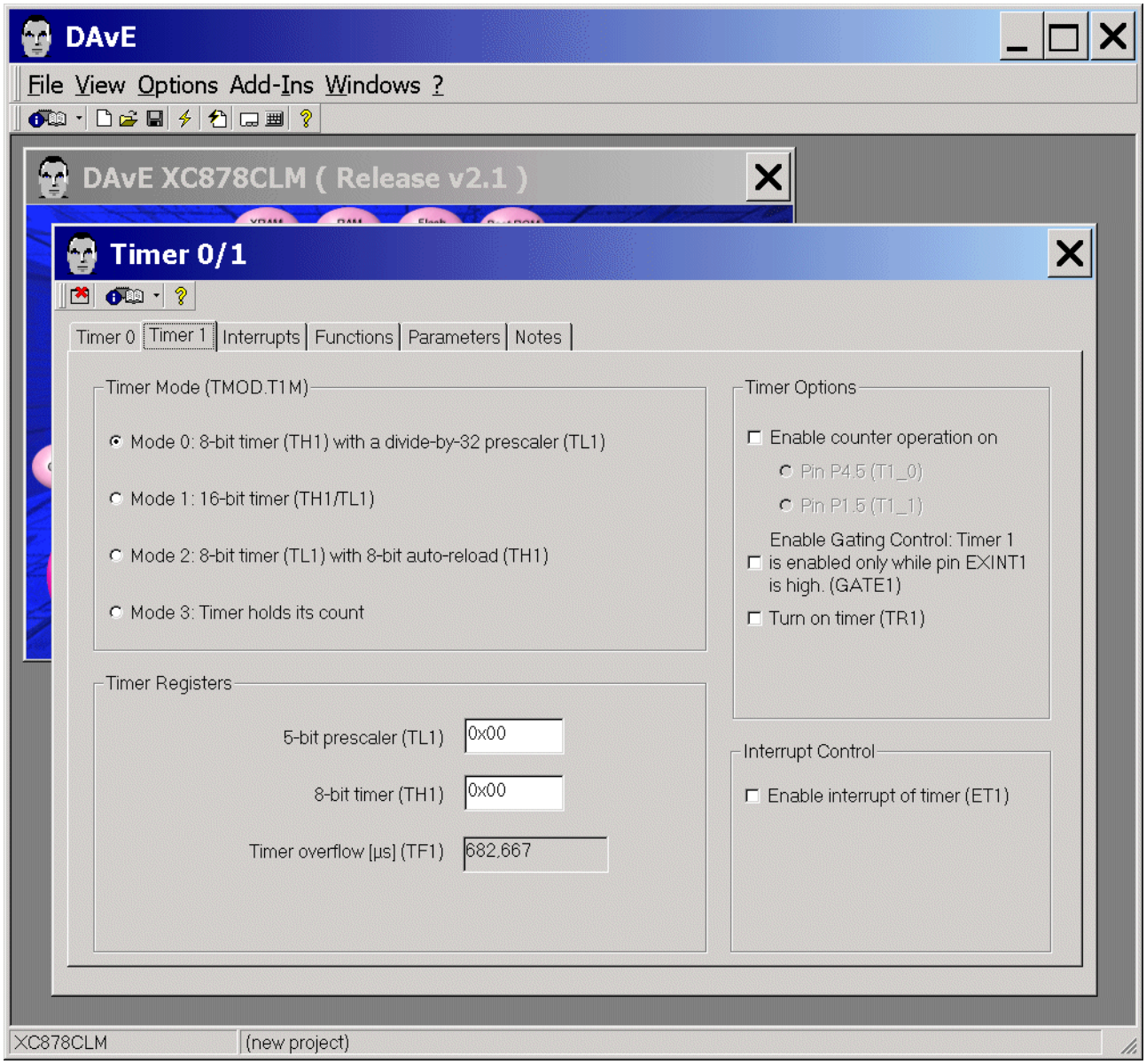
Note:
The LEDs on Port_3 will be blinking (if selected in the main menu)
with a frequency of about 1 second (done in the Timer_0-Interrupt-Service-Routine).
Therefore we have to configure Timer_0.

Timer0: Timer Mode: click ⦿ Mode 1: 16-bit timer
Timer0: Timer Options: tick ✓ Turn on timer (TR0)
Timer0: Interrupt Control: tick ✓ Enable interrupt of timer (ET0)

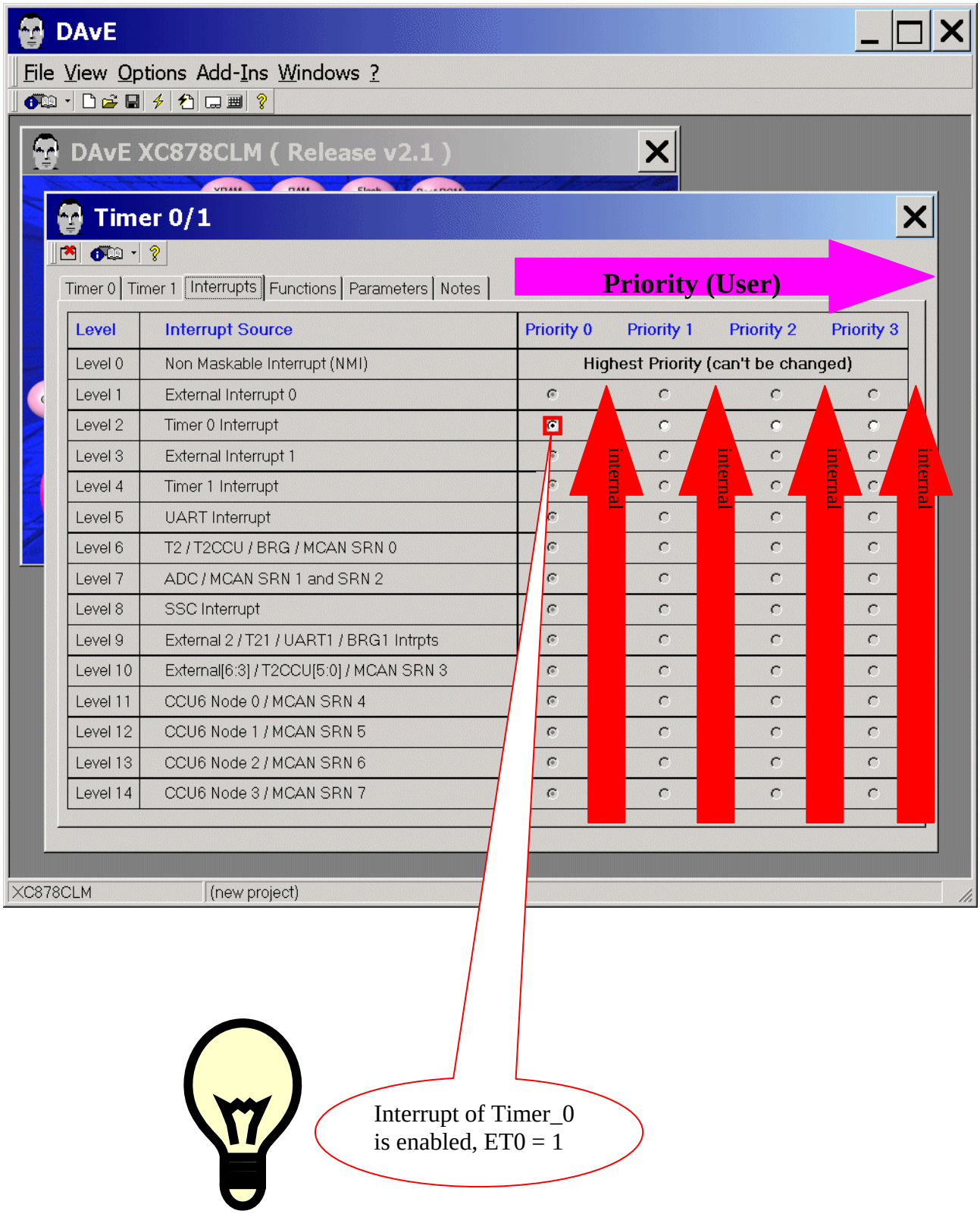


Note:
We need 183 Timer_0 overflows to achieve an approximate 1 second delay.
This will be handled in the Timer_0 interrupt function.
183 * 5461,333 µs = 0,9994 s.

Timer1: do nothing (not used)

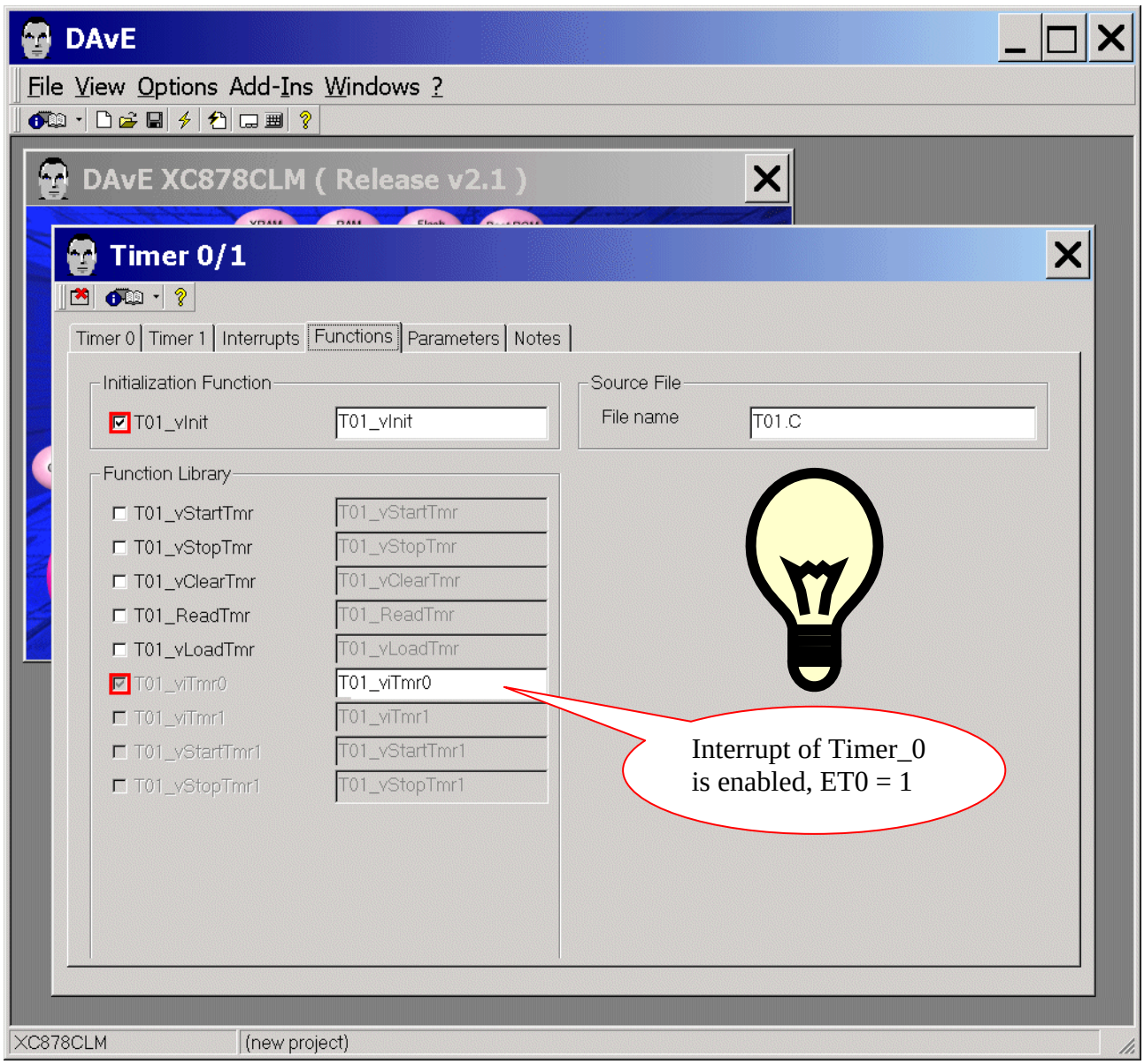


Interrupts: (do nothing)



Interrupt of Timer_0
is enabled, ET0 = 1

Functions: Initialization Function: tick ✓ T01_vInit



Note:
Timer_0 has a dedicated interrupt vector address ($000B_H$), interrupt node and its own interrupt status flag TF0.
The vector is used to service the corresponding interrupt node request – when enabled (ET0=1), which means: the interrupt system will hardware-generate an LCALL to the appropriate service routine at $000B_H$.
TF0 will be automatically cleared by hardware (the core) once its pending interrupt request is serviced.

Additional information: Interrupt Handling (Source: User’s Manual):

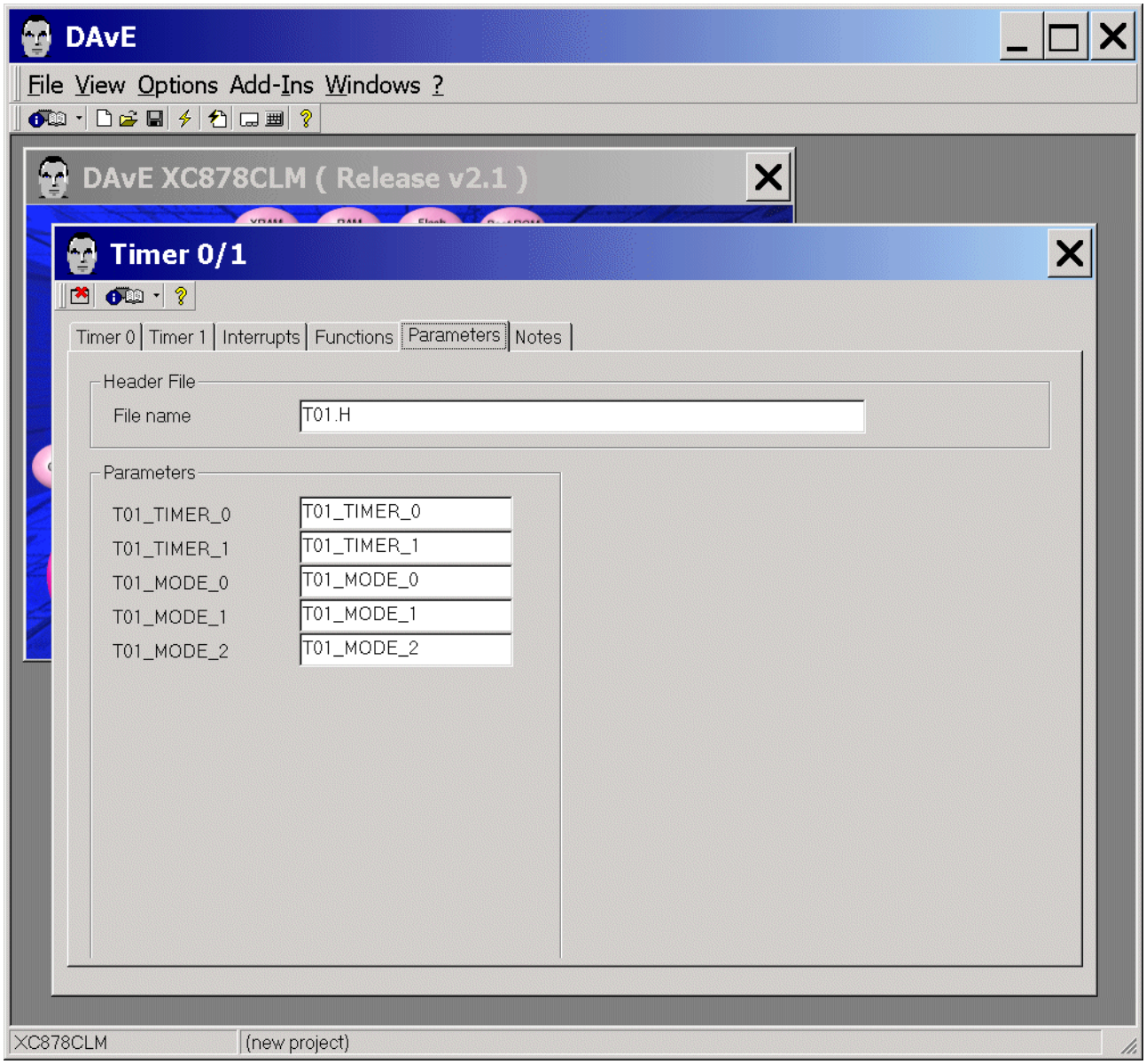
The processor acknowledges an interrupt request by executing a hardware generated LCALL to the appropriate service routine (interrupt vector address).

In some cases, hardware also clears the flag that generated the interrupt, while in other cases, the flag must be cleared by the user’s software (e.g. see DAvE Source Code).

The hardware-generated LCALL pushes the contents of the Program Counter (PC) onto the stack (but it does not save the PSW) and reloads the PC with an address that depends on the source of the interrupt being vectored to (interrupt vector addresses see User’s Manual).

Program execution returns to the next instruction after calling the interrupt when the RETI instruction is encountered. The RETI instruction informs the processor that the interrupt routine is no longer in progress, then pops the two top bytes from the stack and reloads the PC.
Execution of the interrupted program continues from the point where it was stopped.
Note that the RETI instruction is important because it informs the processor that the program has left the current interrupt priority level.
A simple RET instruction would also have returned execution to the interrupted program, but it would have left the interrupt control system on the assumption that an interrupt was still in progress. In this case, no interrupt of the same or lower priority level would be acknowledged.

Parameters: (do nothing)

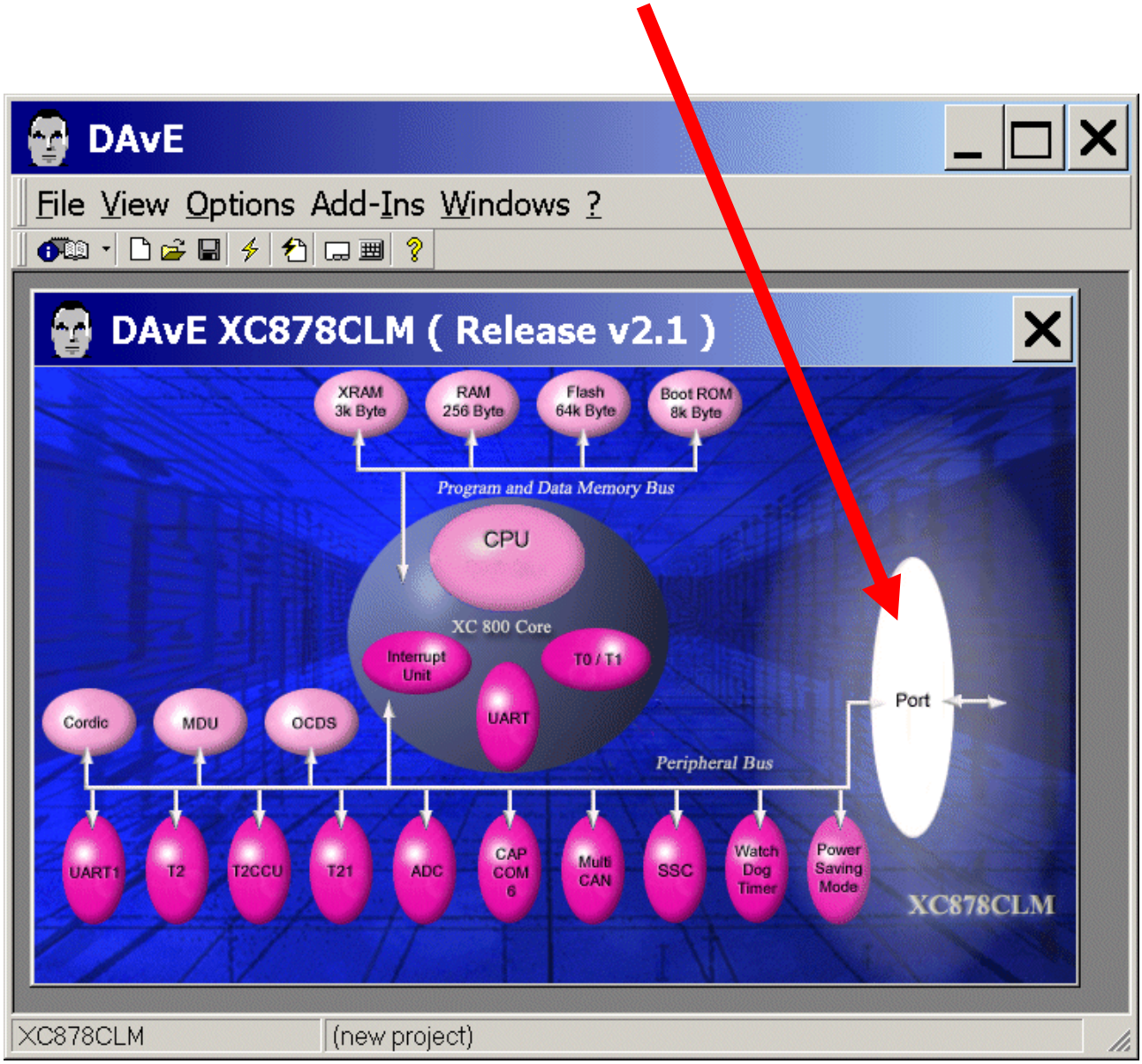


Notes: If you wish, you can insert your comments here.

Exit this dialog now by clicking the close button.

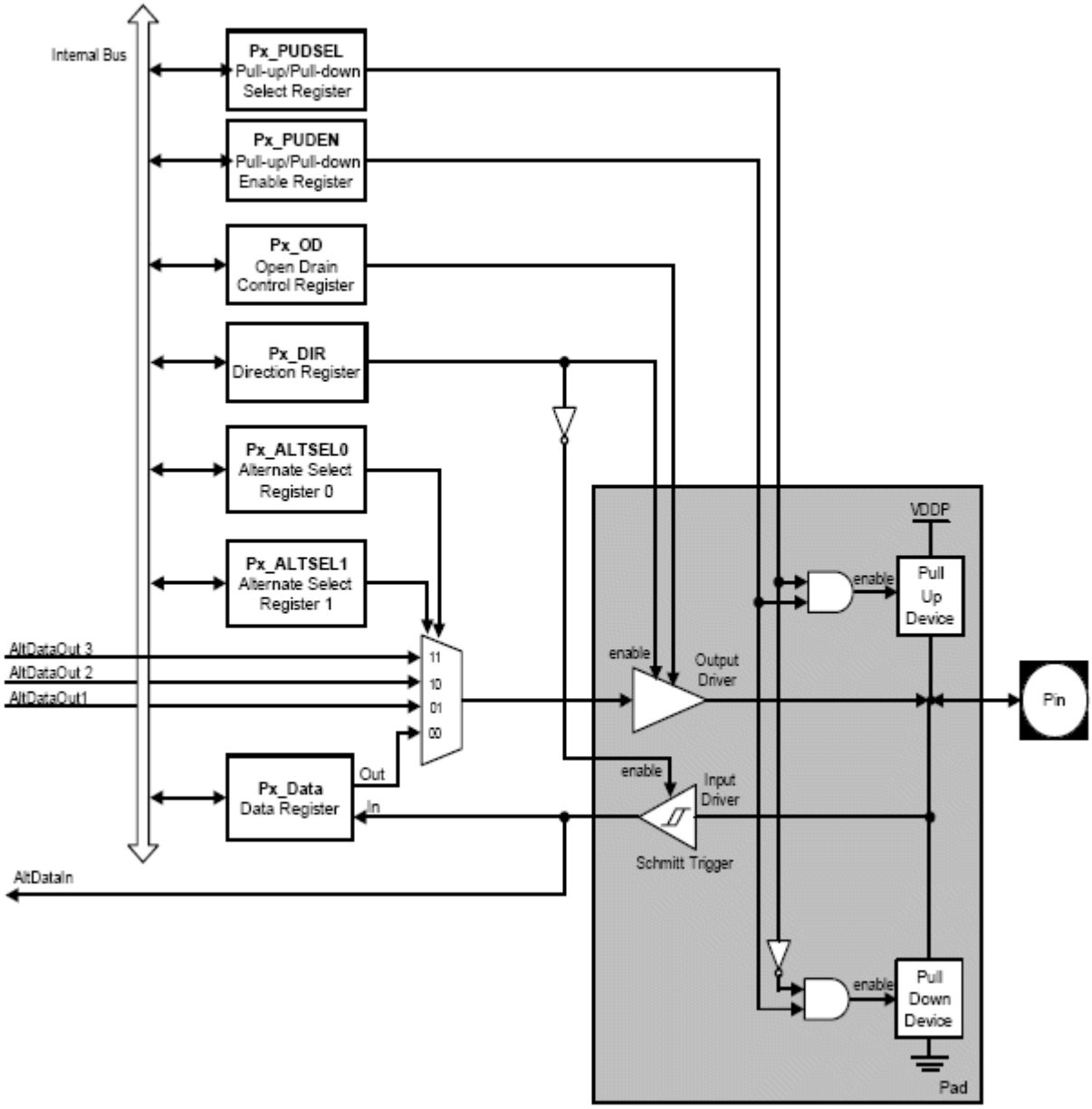
Configure Port 3 to Output:

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

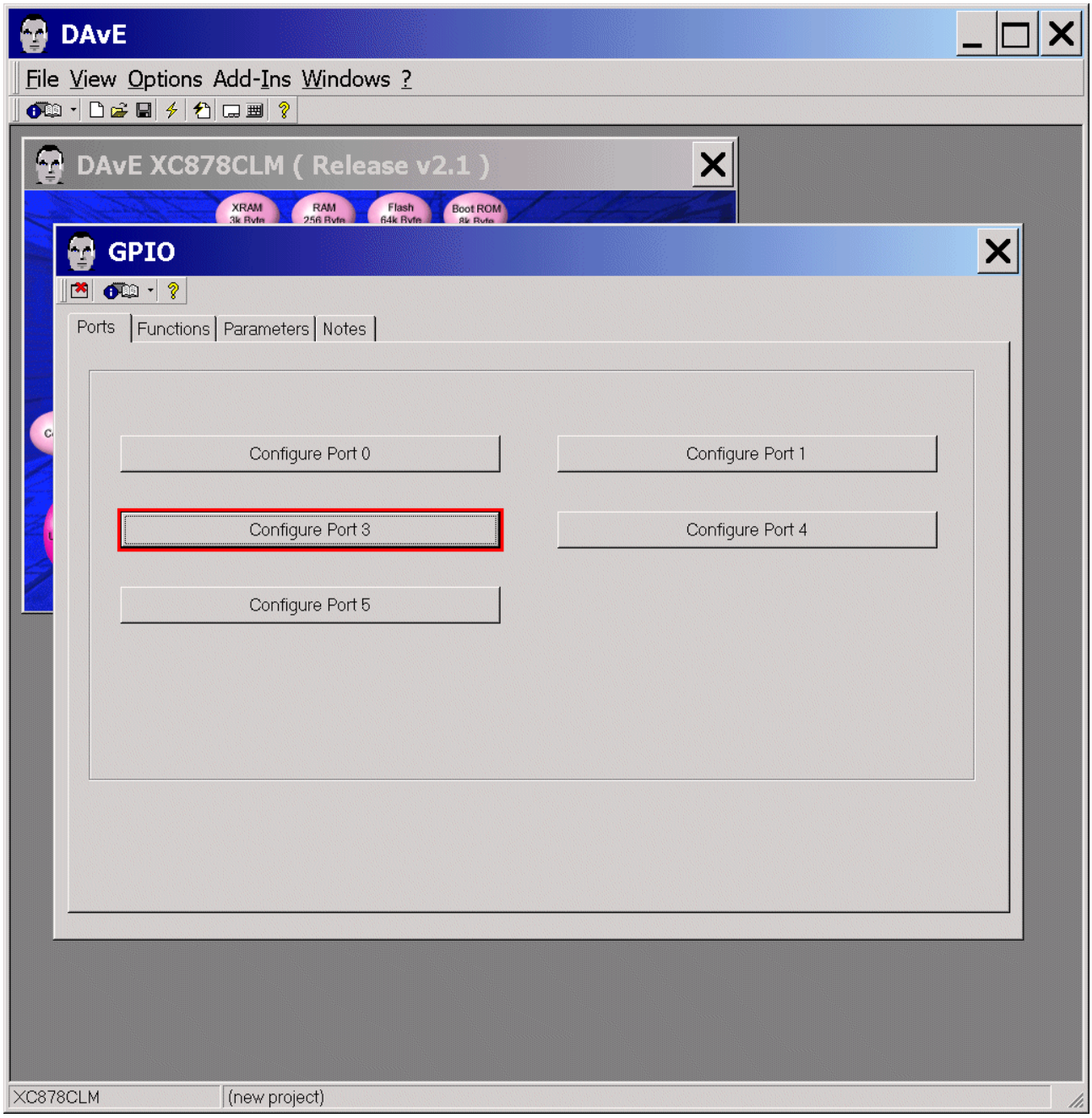


Note:
The User LEDs (red) are connected to Port_3.

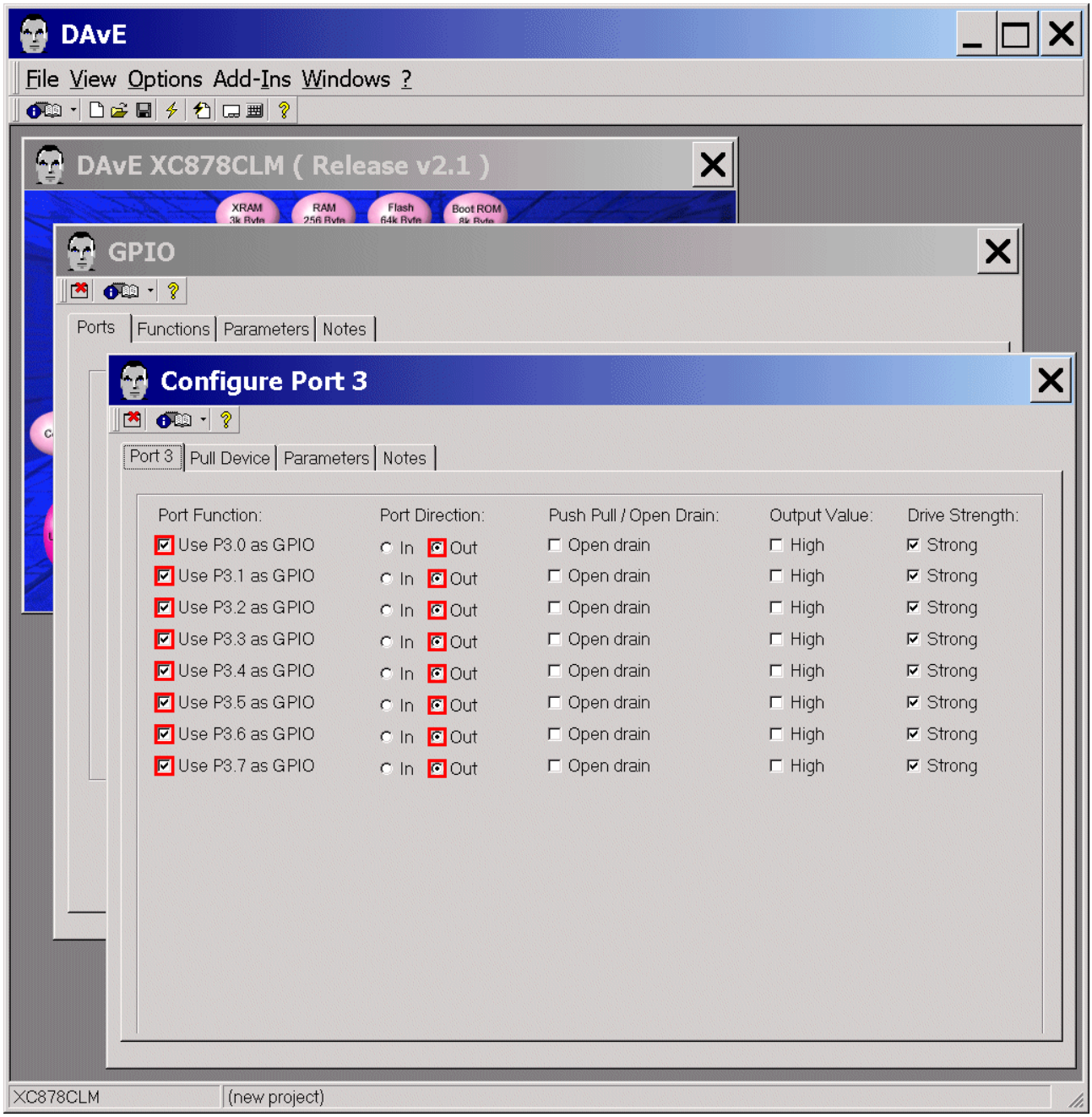
Additional information: Parallel Ports – General Structure (Source: User's Manual):



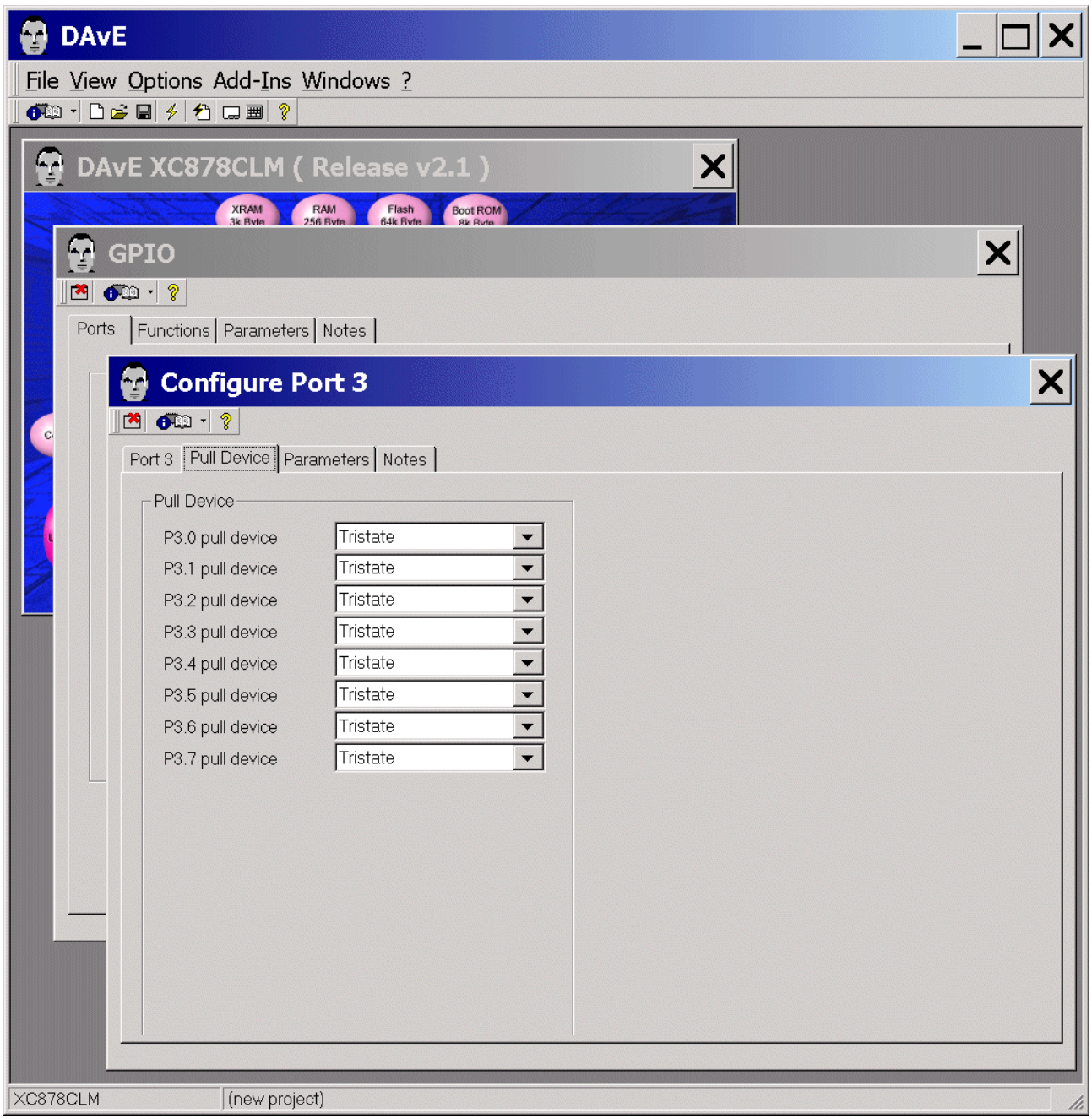
Ports: click “Configure Port 3”



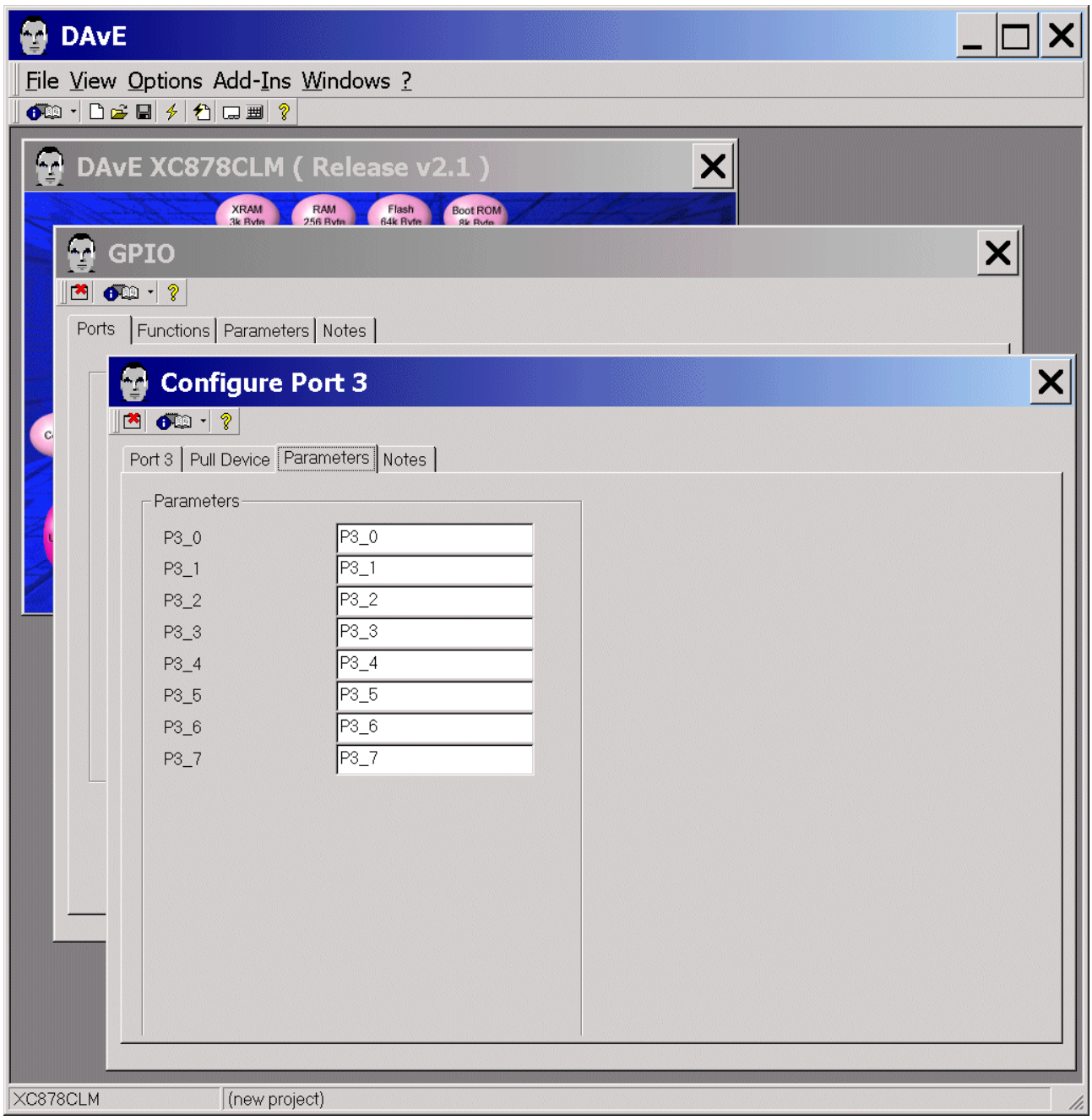
Port 3: Port Function: tick ✓ Use P3.0 as GPIO - Port Direction: click ⦿ Out
Port 3: Port Function: tick ✓ Use P3.1 as GPIO - Port Direction: click ⦿ Out
Port 3: Port Function: tick ✓ Use P3.2 as GPIO - Port Direction: click ⦿ Out
Port 3: Port Function: tick ✓ Use P3.3 as GPIO - Port Direction: click ⦿ Out
Port 3: Port Function: tick ✓ Use P3.4 as GPIO - Port Direction: click ⦿ Out
Port 3: Port Function: tick ✓ Use P3.5 as GPIO - Port Direction: click ⦿ Out
Port 3: Port Function: tick ✓ Use P3.6 as GPIO - Port Direction: click ⦿ Out
Port 3: Port Function: tick ✓ Use P3.7 as GPIO - Port Direction: click ⦿ Out



Pull Device: (do nothing)



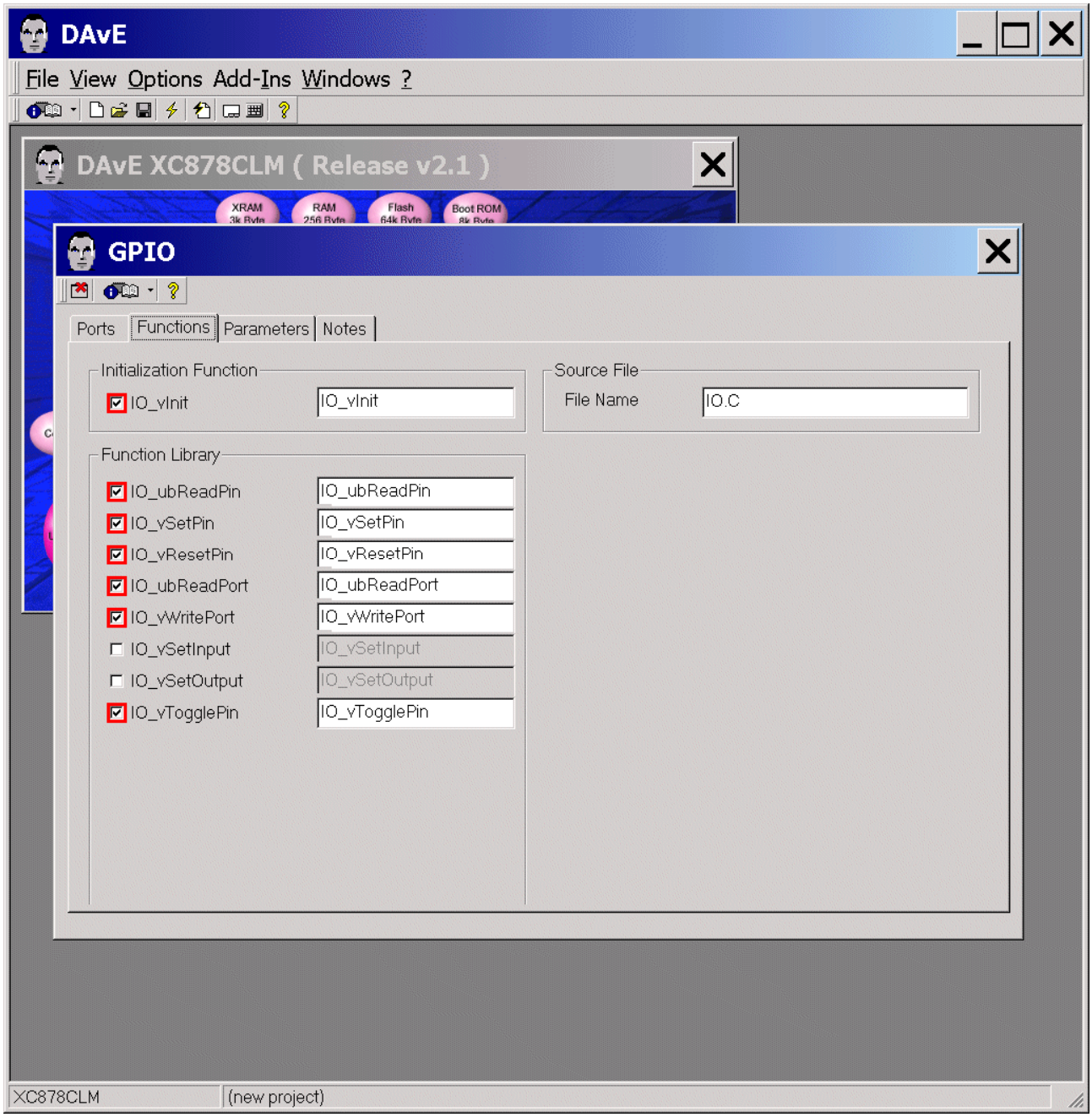
Parameters: (do nothing)



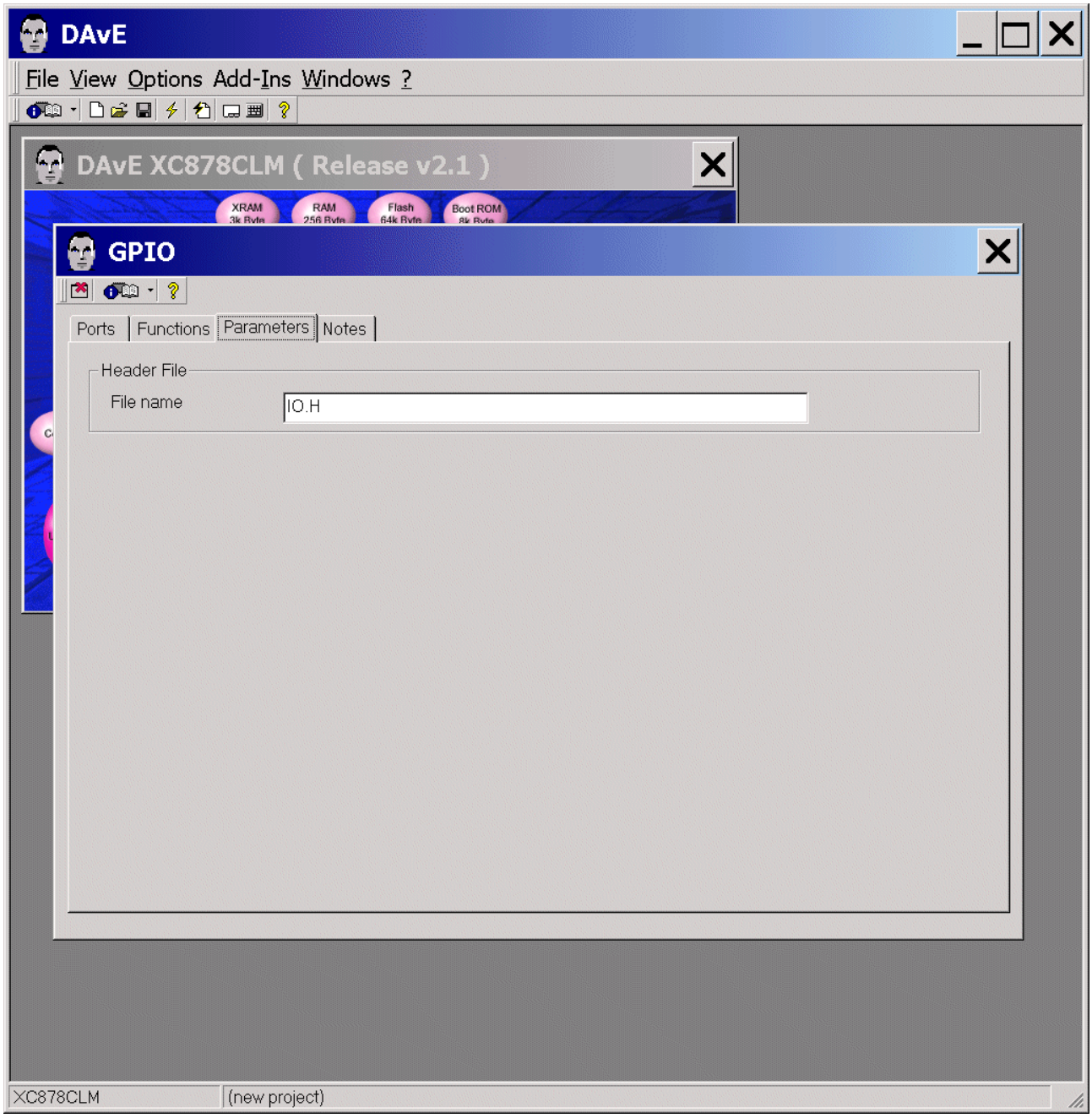
Notes: If you wish, you can insert your comments here.

Exit this dialog now by clicking the close button.

Functions: Initialization Functions: tick ✓ IO_vInit
Functions: Function Library: tick ✓ IO_ubReadPin
Functions: Function Library: tick ✓ IO_vSetPin
Functions: Function Library: tick ✓ IO_vResetPin
Functions: Function Library: tick ✓ IO_ubReadPort
Functions: Function Library: tick ✓ IO_vWritePort
Functions: Function Library: tick ✓ IO_vTogglePin



Parameters: (do nothing)



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

Exit this dialog now by clicking the close button.

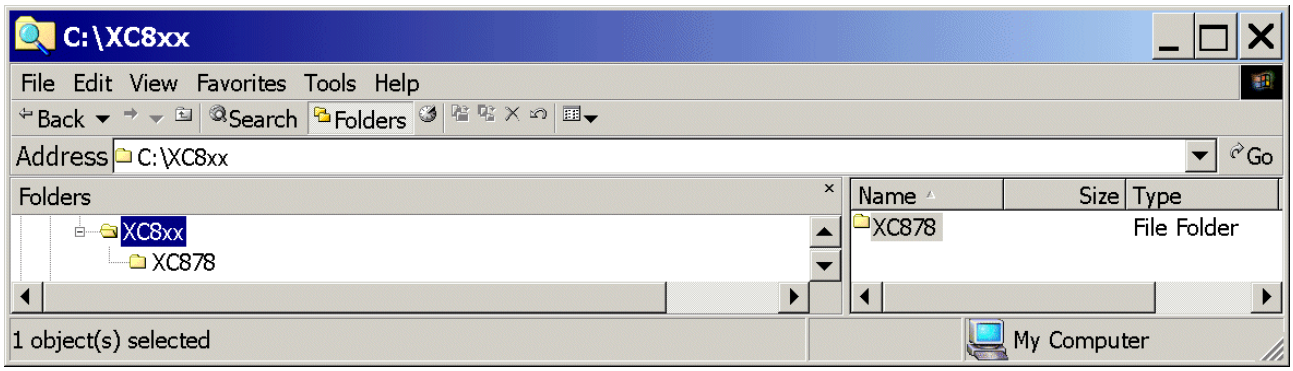
Note:
Before we save the DAvE Project we are going to create a suitable directory structure with Windows File Explorer (see next page!).

Create a suitable directory structure for DAvE-Bench (Eclipse IDE, SDCC compiler):

Start Windows File Explorer

Create directory/folder C:**XC8xx**

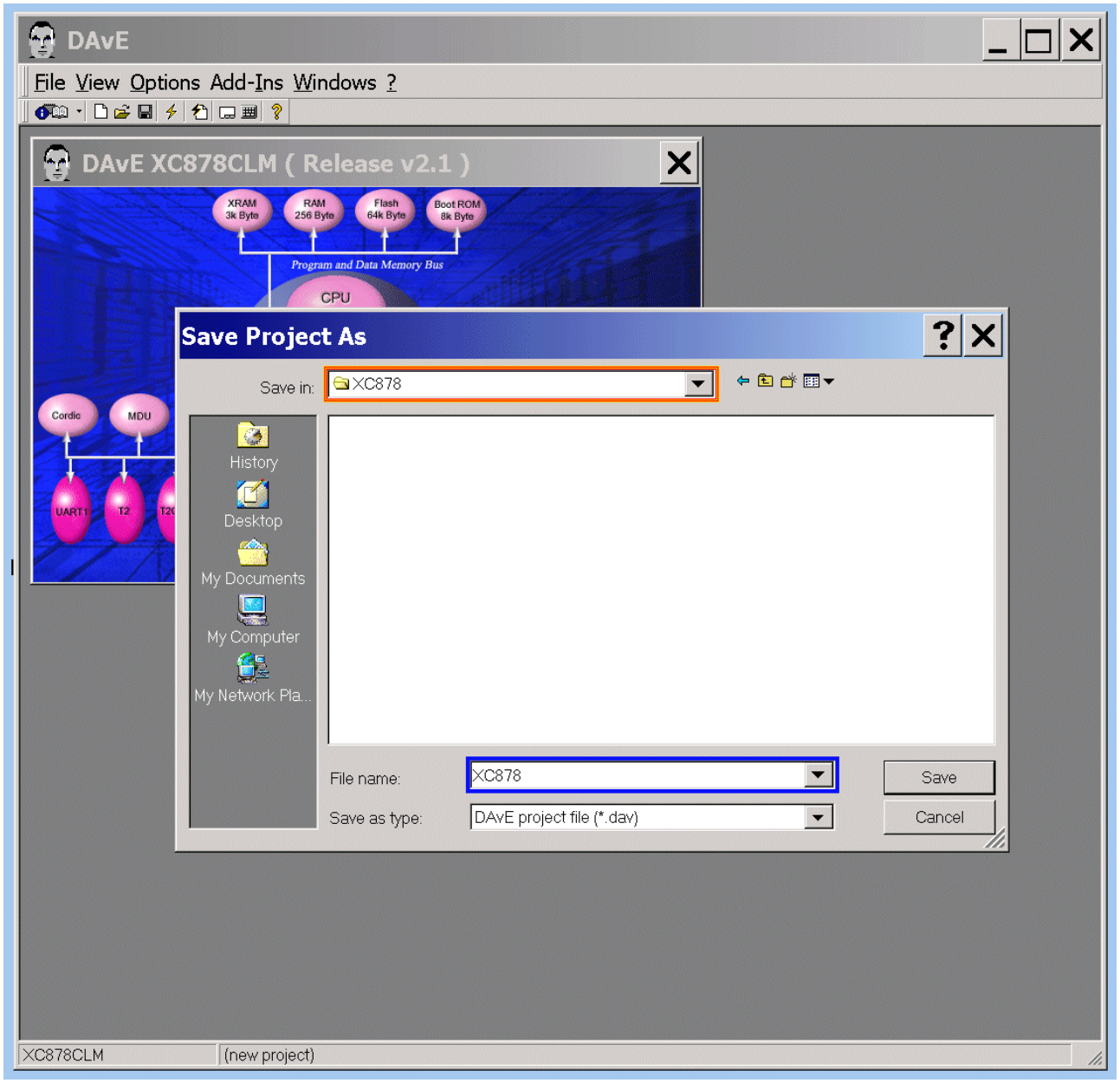
Create directory/folder C:**XC8xx****XC878**



Save the project:

File
Save

Save project: Save in select C:\XC8xx\XC878
File name: insert XC878

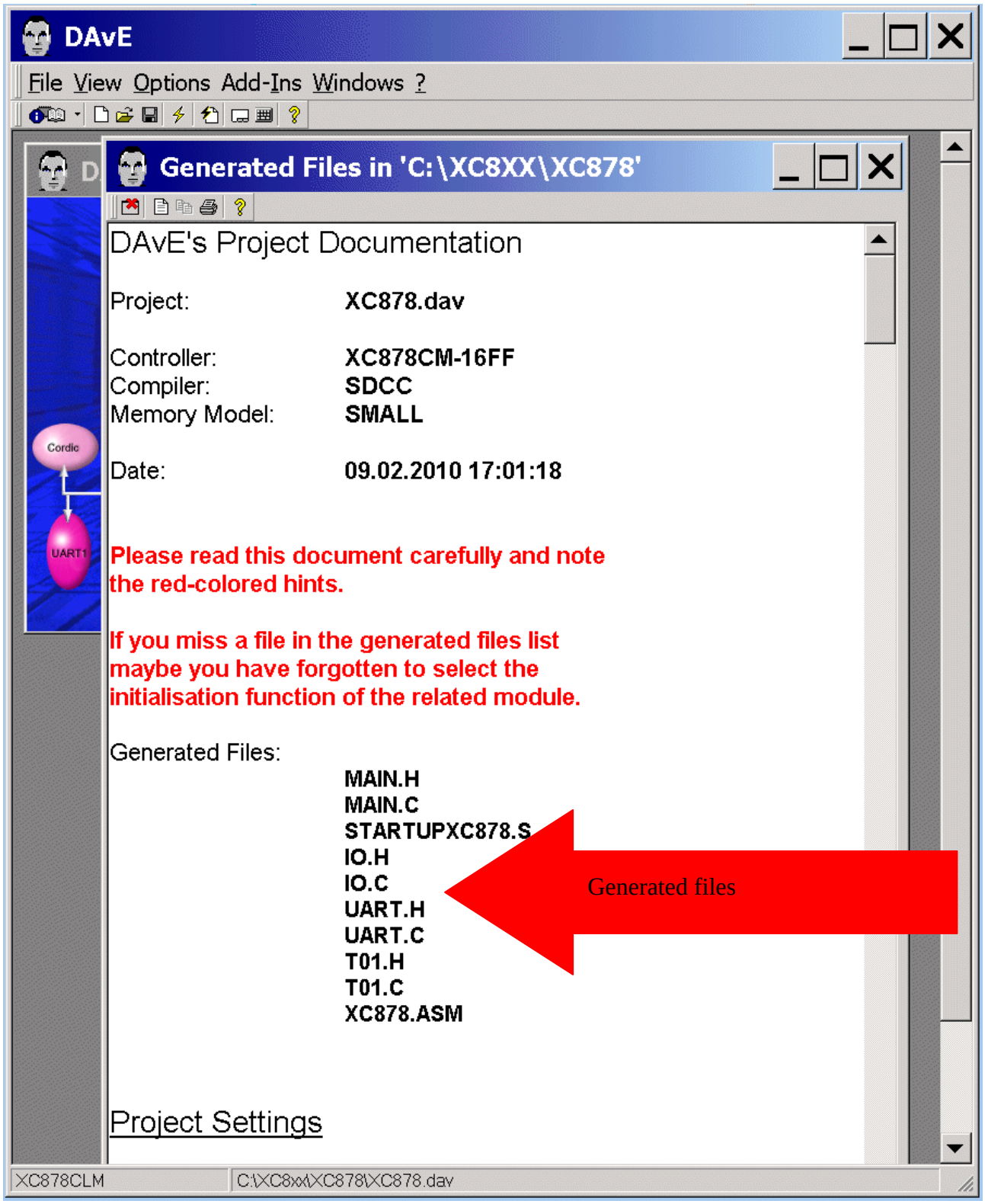


Save

Generate Code:

File Generate Code	or click 

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



File - Exit

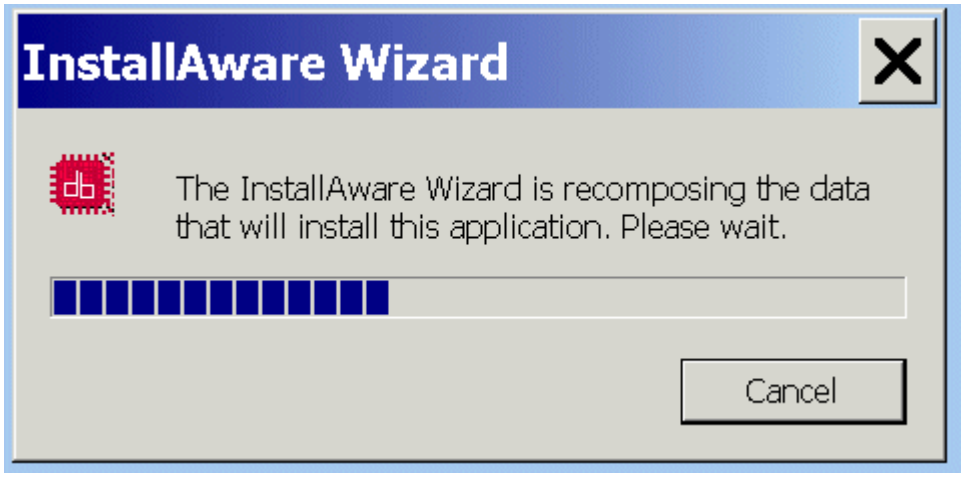
Save changes?

click Yes

4.) Using DAvE Bench:

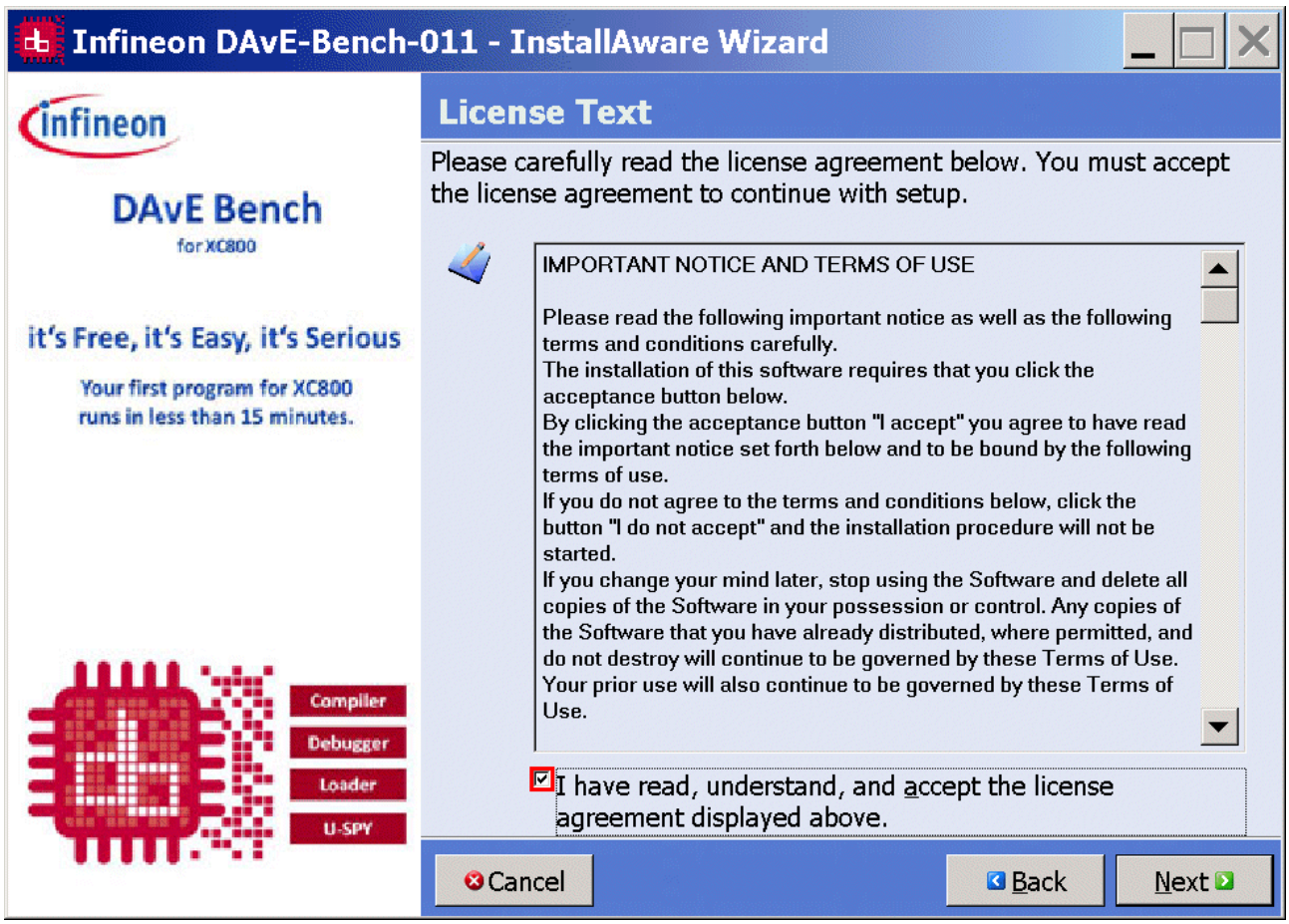
Download and install DAvE Bench (http://www.infineon.com/DAvE-Bench):

Double-click setup.exe:



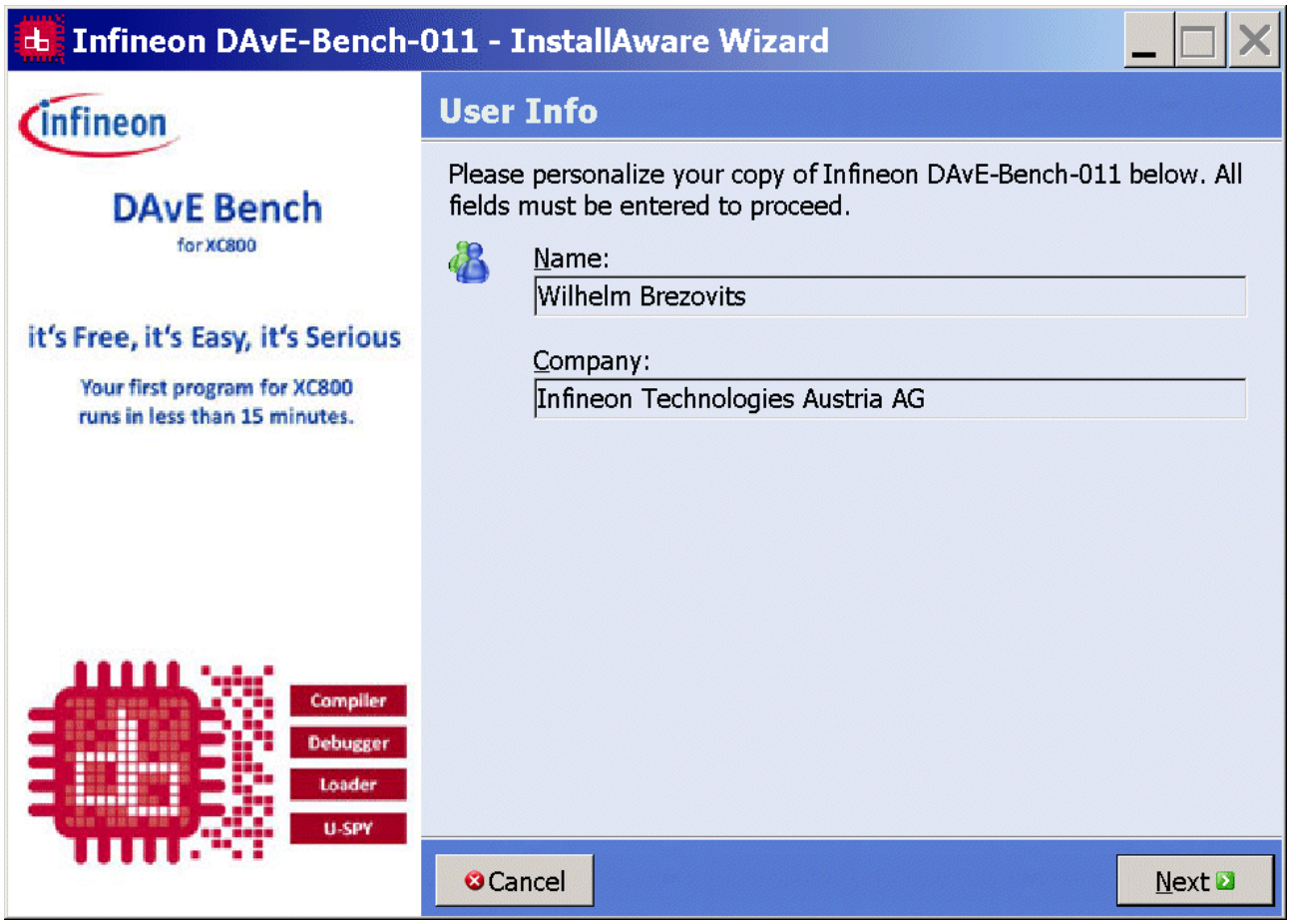

Infineon DAvE-Bench-100 - InstallAware Wizard
Welcome
DAvE Bench
for XC800
it's Free, it's Easy, it's Powerful
Your first program for XC800
runs in less than 15 minutes.
Compiler
Debugger
Loader
U-SPY
Welcome to the InstallAware Wizard for Infineon DAvE-Bench-100 Setup. This wizard will install release 01.00.0000 on your computer.
You may navigate through the pages in this wizard by clicking the Back and Next buttons below. Click Cancel at anytime to exit the
Cancel
Next

Click 
Next



Tick ☑ I have read, understand…

Click 

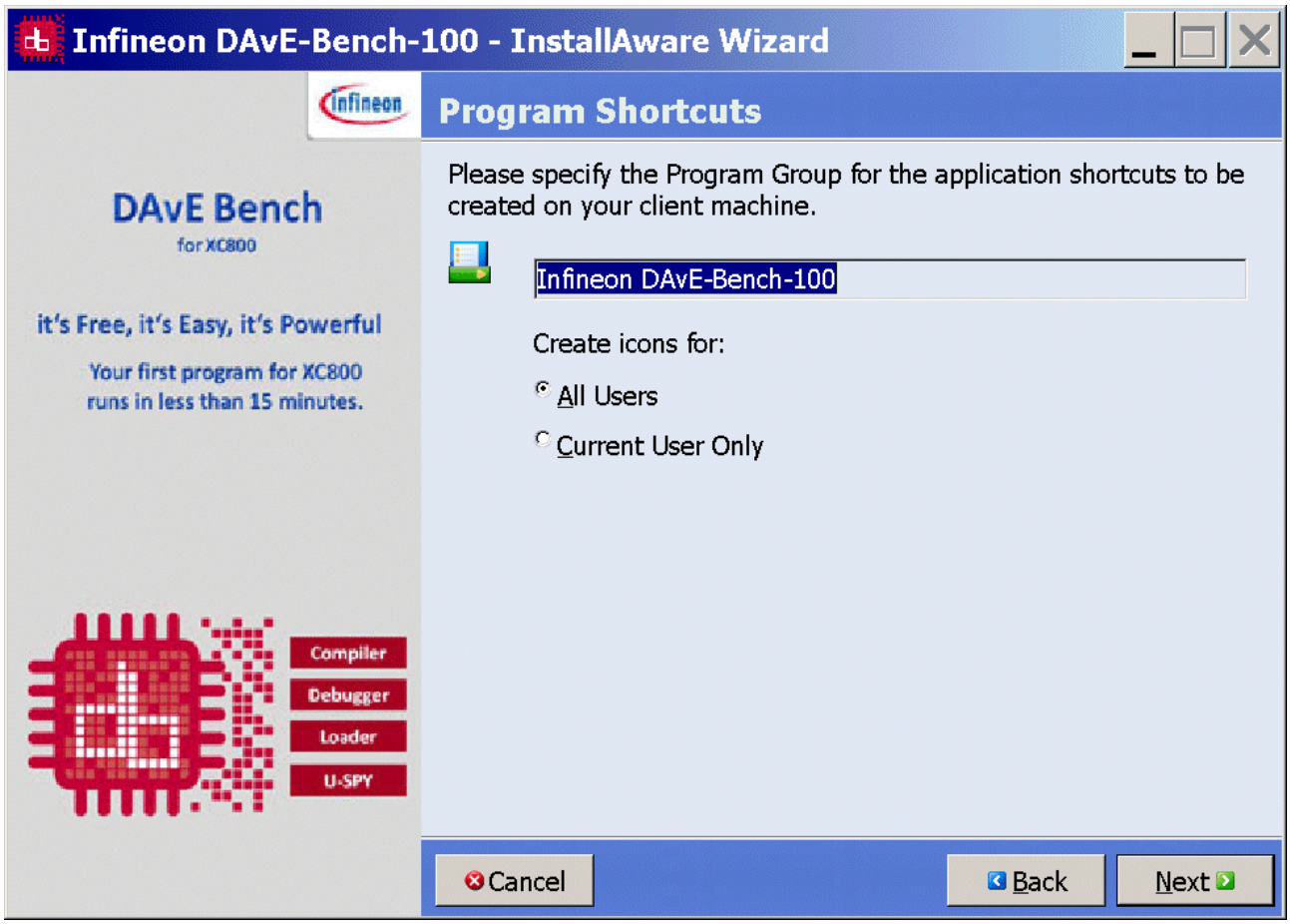


Insert Name: Your name
Insert Company: Your company

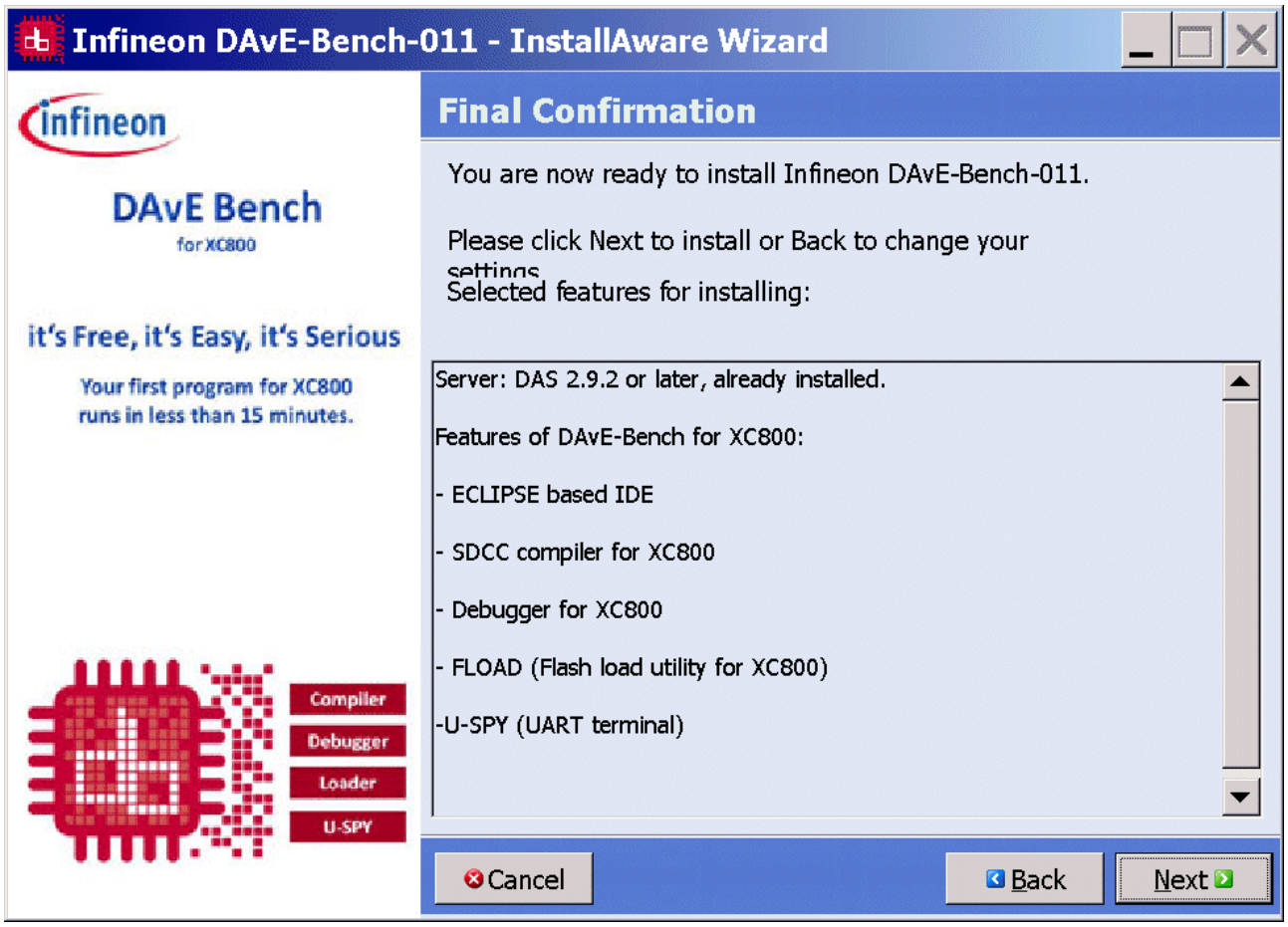
Click 

Select a suitable directory and

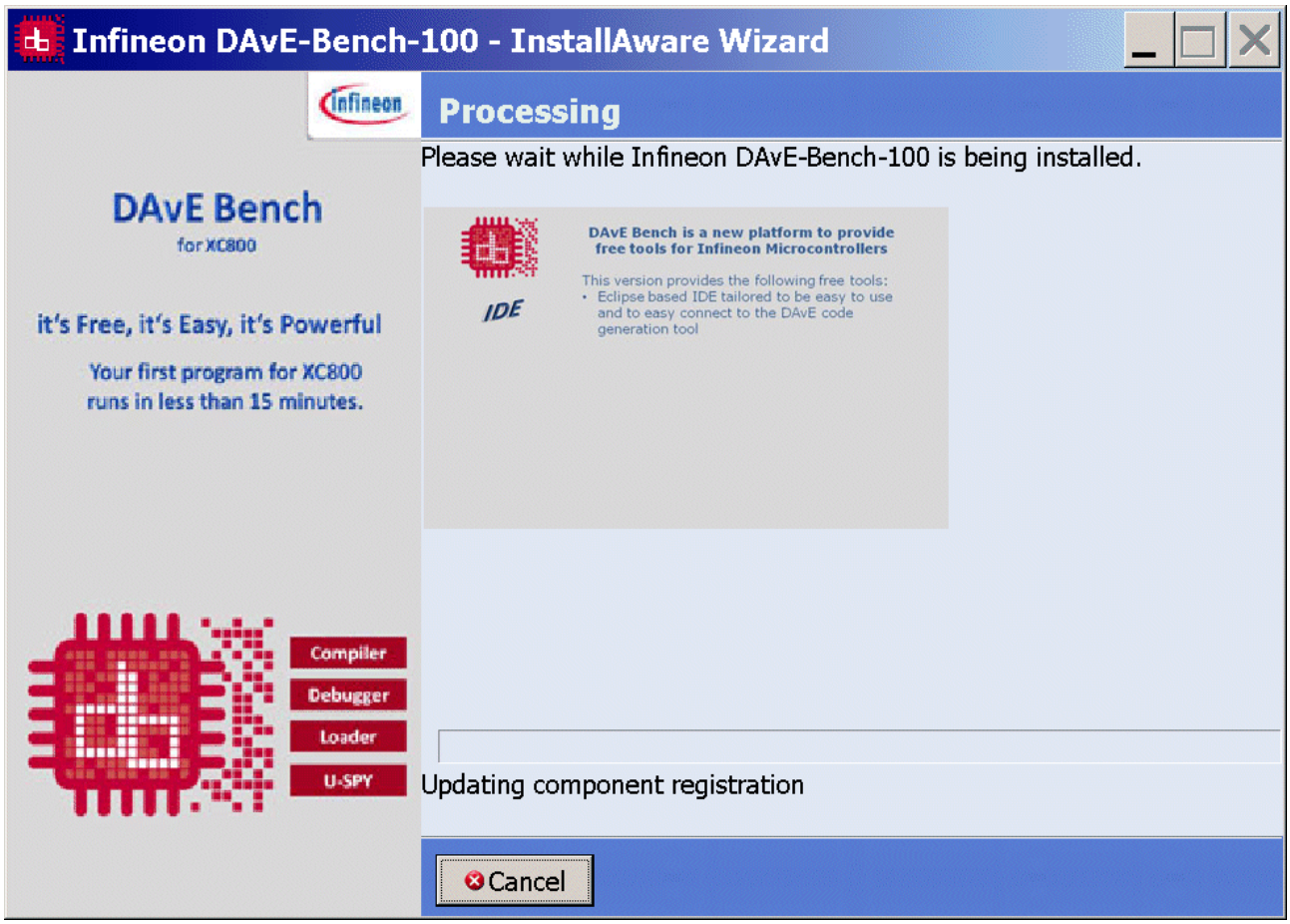
Click 

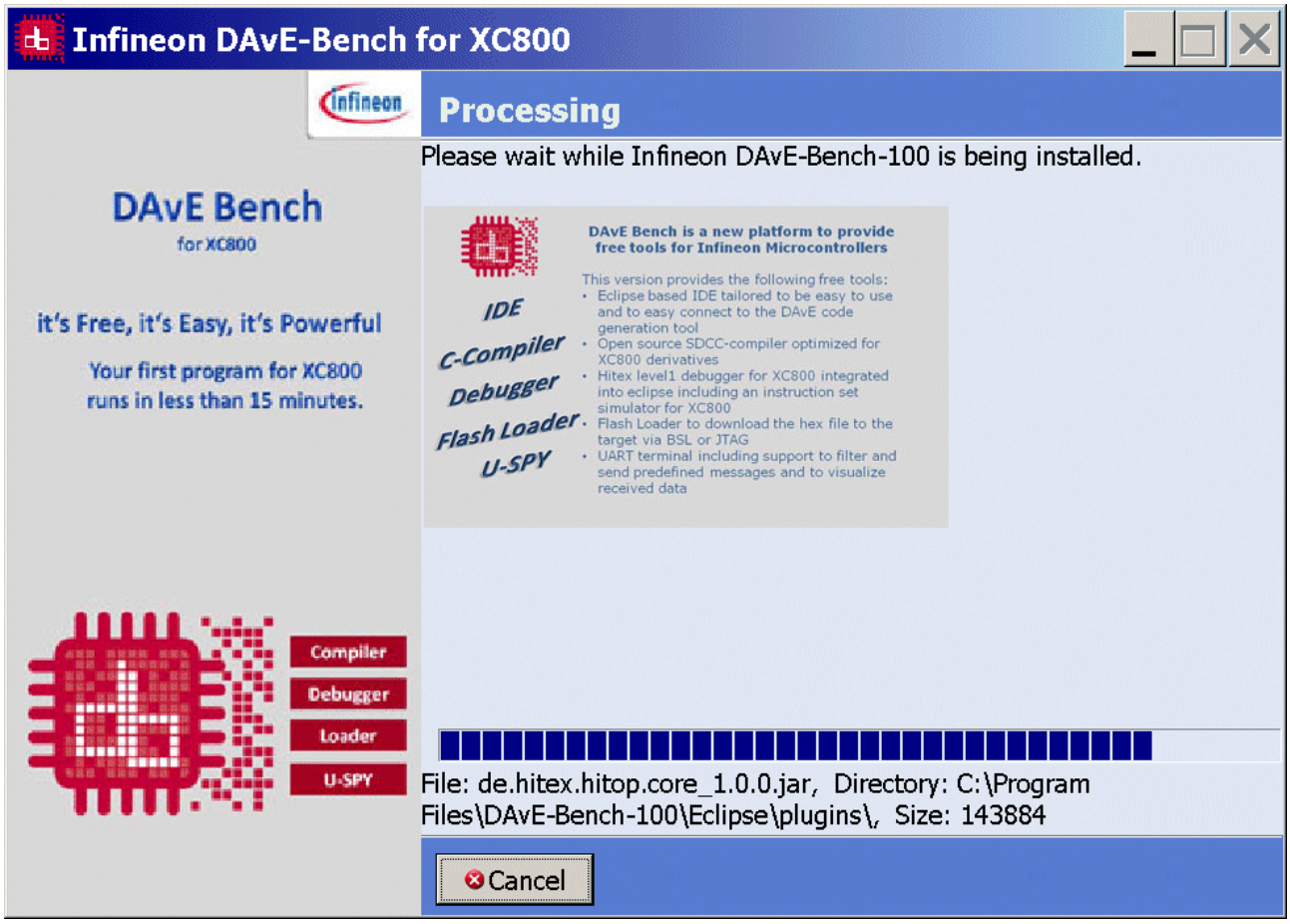

Infineon DAvE-Bench-100 - InstallAware Wizard
Program Shortcuts
DAvE Bench
for XC800
it's Free, it's Easy, it's Powerful
Your first program for XC800
runs in less than 15 minutes.
Compiler
Debugger
Loader
U-SPY
Please specify the Program Group for the application shortcuts to be created on your client machine.
Infineon DAvE-Bench-100
Create icons for:
All Users
Current User Only
Cancel
Back
Next

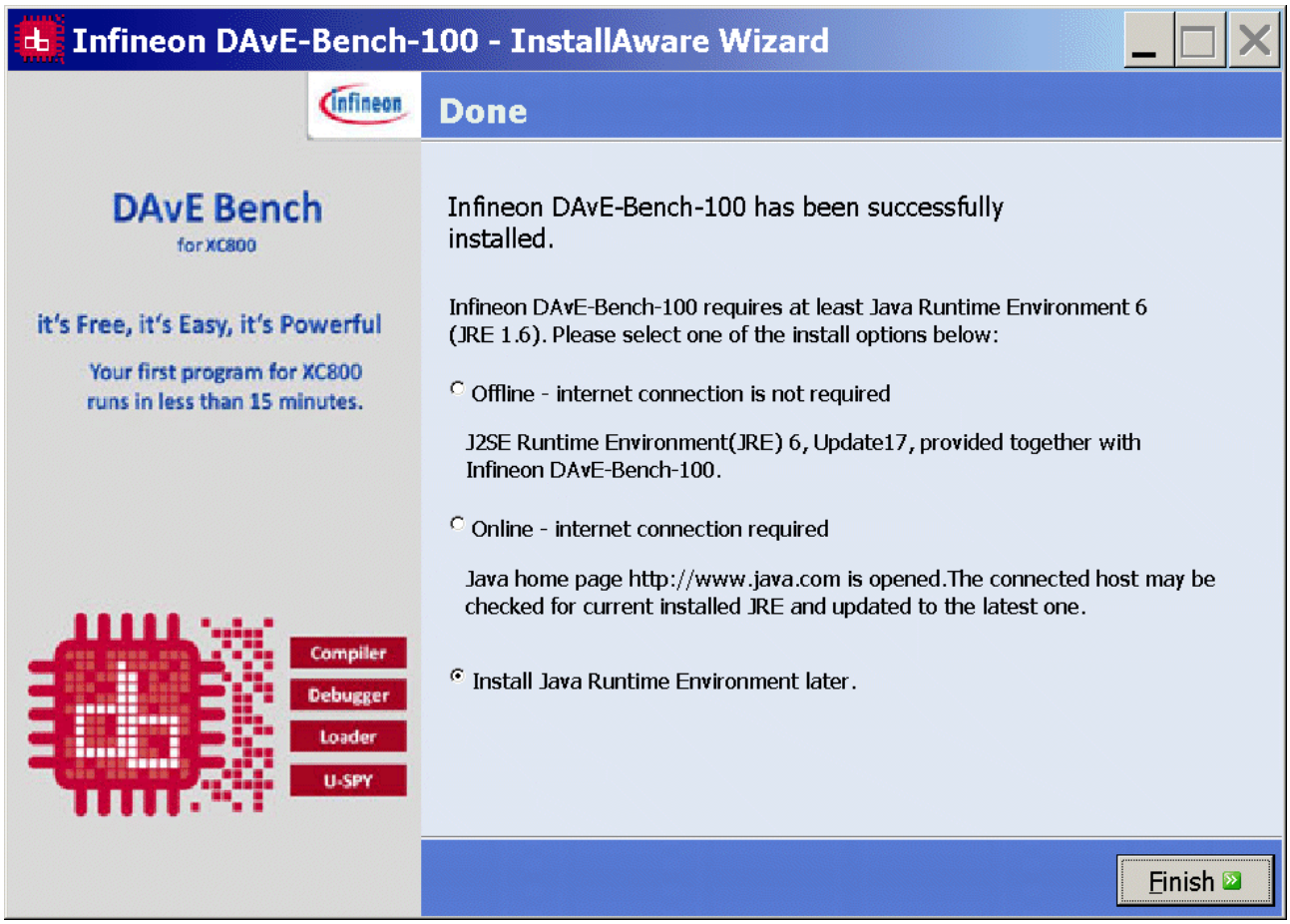
Click 
Next


Infineon DAvE-Bench-011 - InstallAware Wizard
Final Confirmation
DAvE Bench
for XC800
it's Free, it's Easy, it's Serious
Your first program for XC800
runs in less than 15 minutes.
Compiler
Debugger
Loader
U-SPY
You are now ready to install Infineon DAvE-Bench-011.
Please click Next to install or Back to change your settings.
Selected features for installing:
Server: DAS 2.9.2 or later, already installed.
Features of DAvE-Bench for XC800:
- ECLIPSE based IDE
- SDCC compiler for XC800
- Debugger for XC800
- FLOAD (Flash load utility for XC800)
-U-SPY (UART terminal)
Cancel
Back
Next

Click 
Next

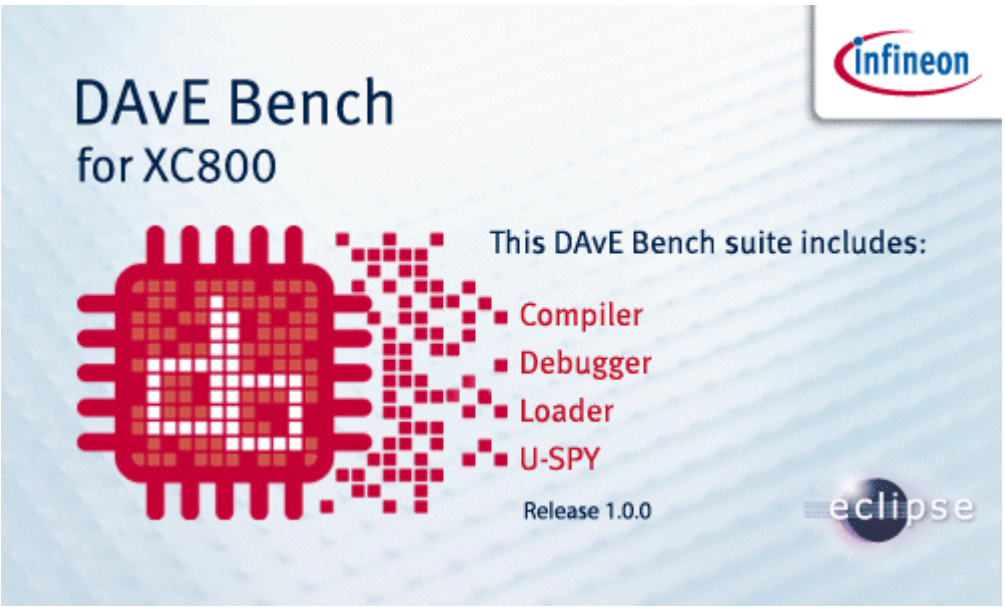

Infineon DAvE-Bench-100 - InstallAware Wizard
Processing
Please wait while Infineon DAvE-Bench-100 is being installed.
DAvE Bench
for XC800
it's Free, it's Easy, it's Powerful
Your first program for XC800
runs in less than 15 minutes.
DAvE Bench is a new platform to provide free tools for Infineon Microcontrollers
This version provides the following free tools:
• Eclipse based IDE tailored to be easy to use and to easy connect to the DAvE code generation tool
IDE
Compiler
Debugger
Loader
U-SPY
Updating component registration
Cancel

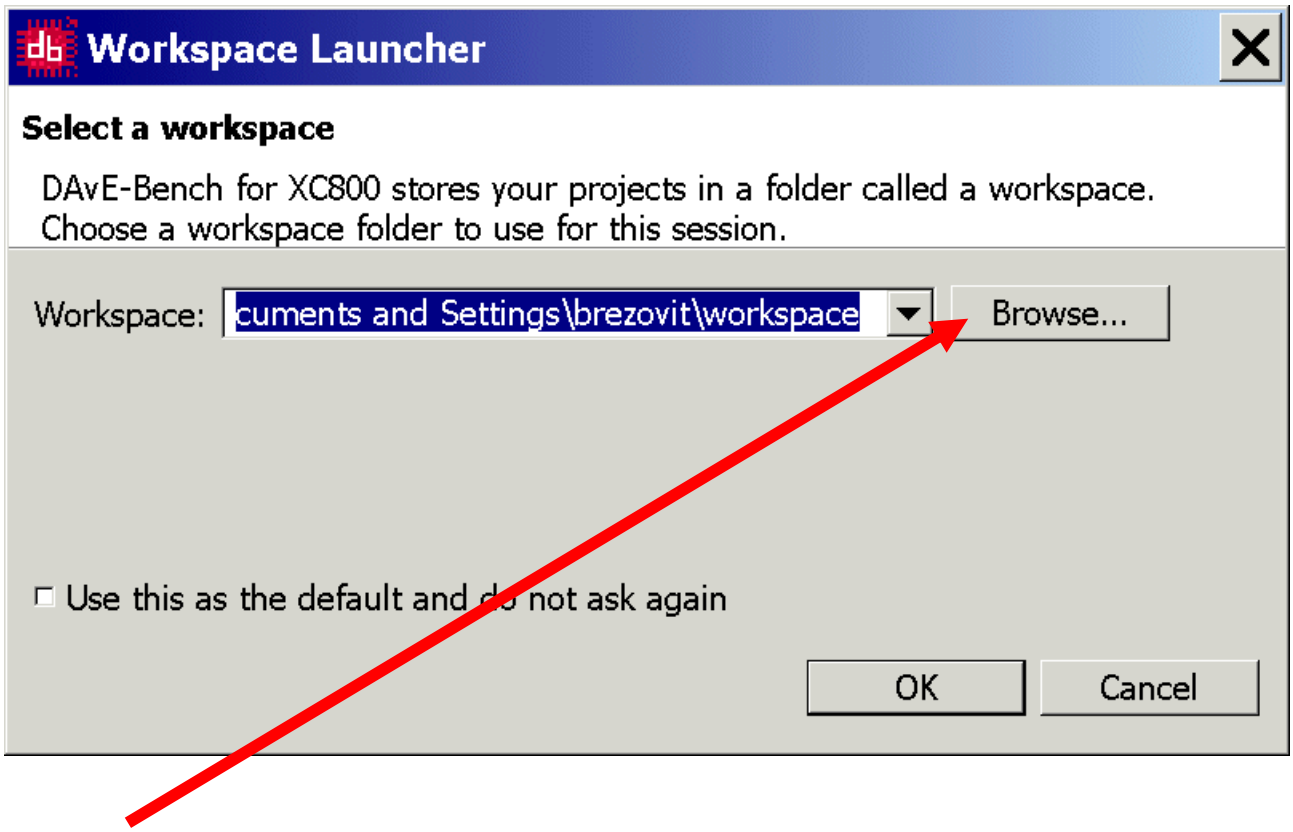

Infineon DAvE-Bench for XC800
Processing
Please wait while Infineon DAvE-Bench-100 is being installed.
DAvE Bench
for XC800
it's Free, it's Easy, it's Powerful
Your first program for XC800
runs in less than 15 minutes.
DAvE Bench is a new platform to provide free tools for Infineon Microcontrollers
This version provides the following free tools:
IDE
C-Compiler
Debugger
Flash Loader
U-SPY
Eclipse based IDE tailored to be easy to use and to easy connect to the DAvE code generation tool
Open source SDCC-compiler optimized for XC800 derivatives
Hitex level1 debugger for XC800 integrated into eclipse including an instruction set simulator for XC800
Flash Loader to download the hex file to the target via BSL or JTAG
UART terminal including support to filter and send predefined messages and to visualize received data
Compiler
Debugger
Loader
U-SPY
File: de.hitex.hitop.core_1.0.0.jar, Directory: C:\Program Files\DAvE-Bench-100\Eclipse\plugins\, Size: 143884
Cancel


Infineon DAvE-Bench-100 - InstallAware Wizard
Done
DAvE Bench
for XC800
it's Free, it's Easy, it's Powerful
Your first program for XC800
runs in less than 15 minutes.
Compiler
Debugger
Loader
U-SPY
Infineon DAvE-Bench-100 has been successfully installed.
Infineon DAvE-Bench-100 requires at least Java Runtime Environment 6 (JRE 1.6). Please select one of the install options below:
Offline - internet connection is not required
J2SE Runtime Environment(JRE) 6, Update17, provided together with Infineon DAvE-Bench-100.
Online - internet connection required
Java home page http://www.java.com is opened.The connected host may be checked for current installed JRE and updated to the latest one.
Install Java Runtime Environment later.
Finish

Click 
Finish

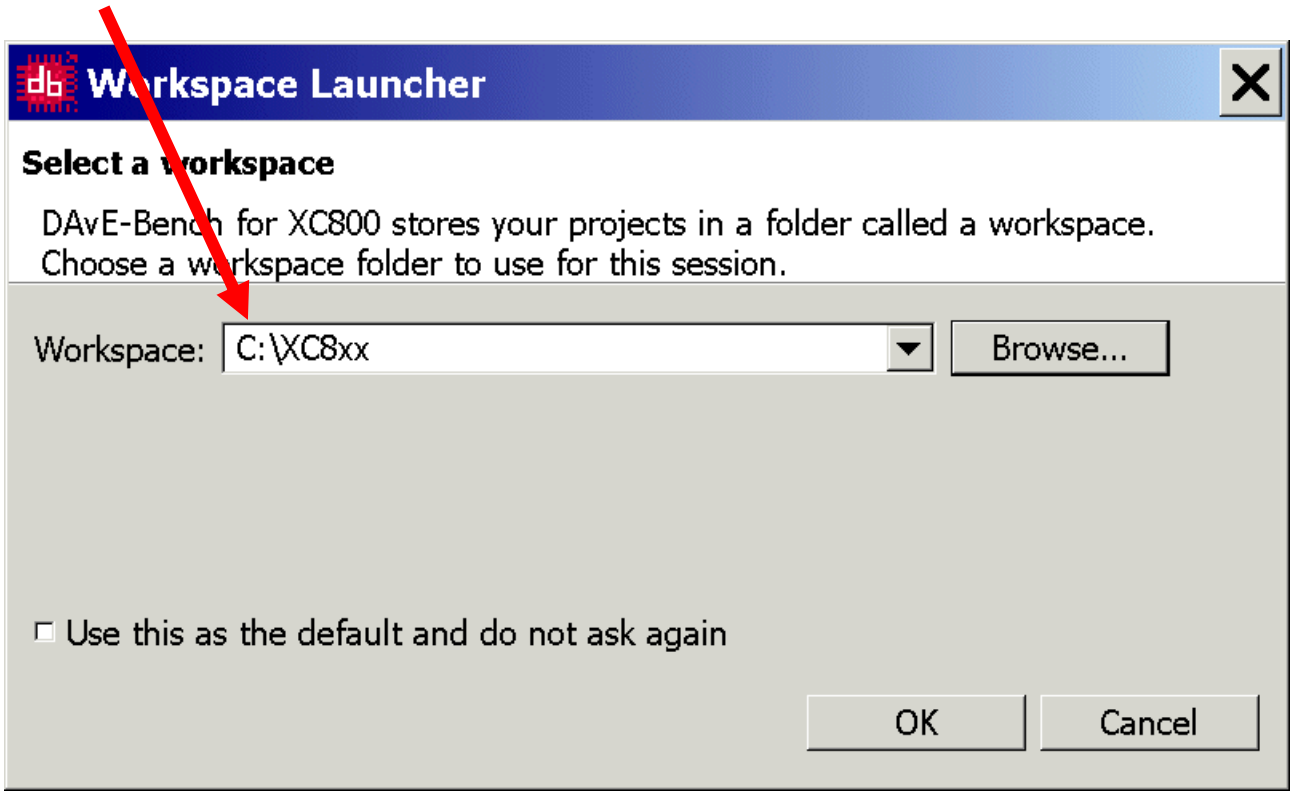
Start/Launch DAvE Bench and open/add the DAvE Project:



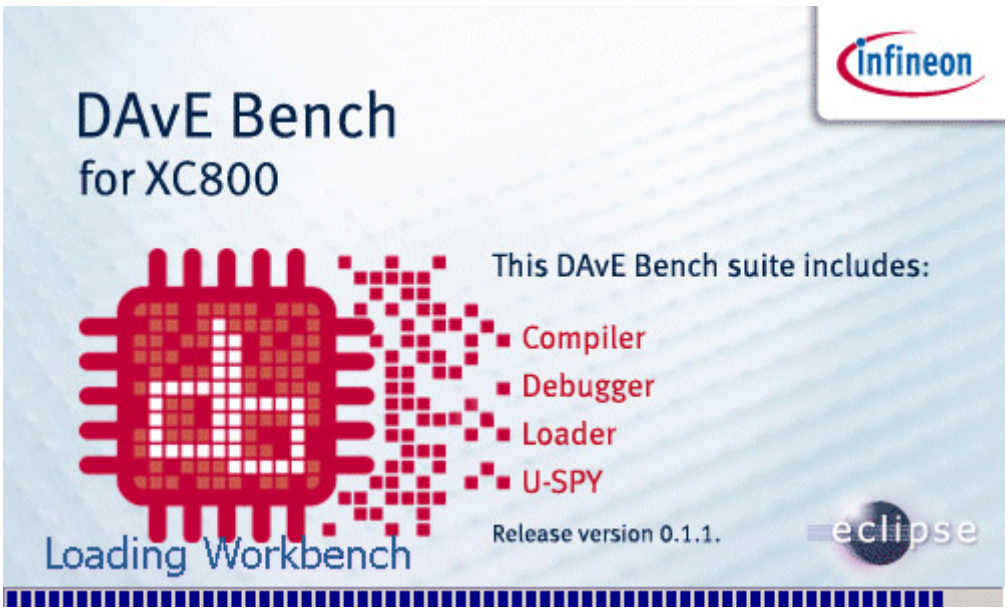


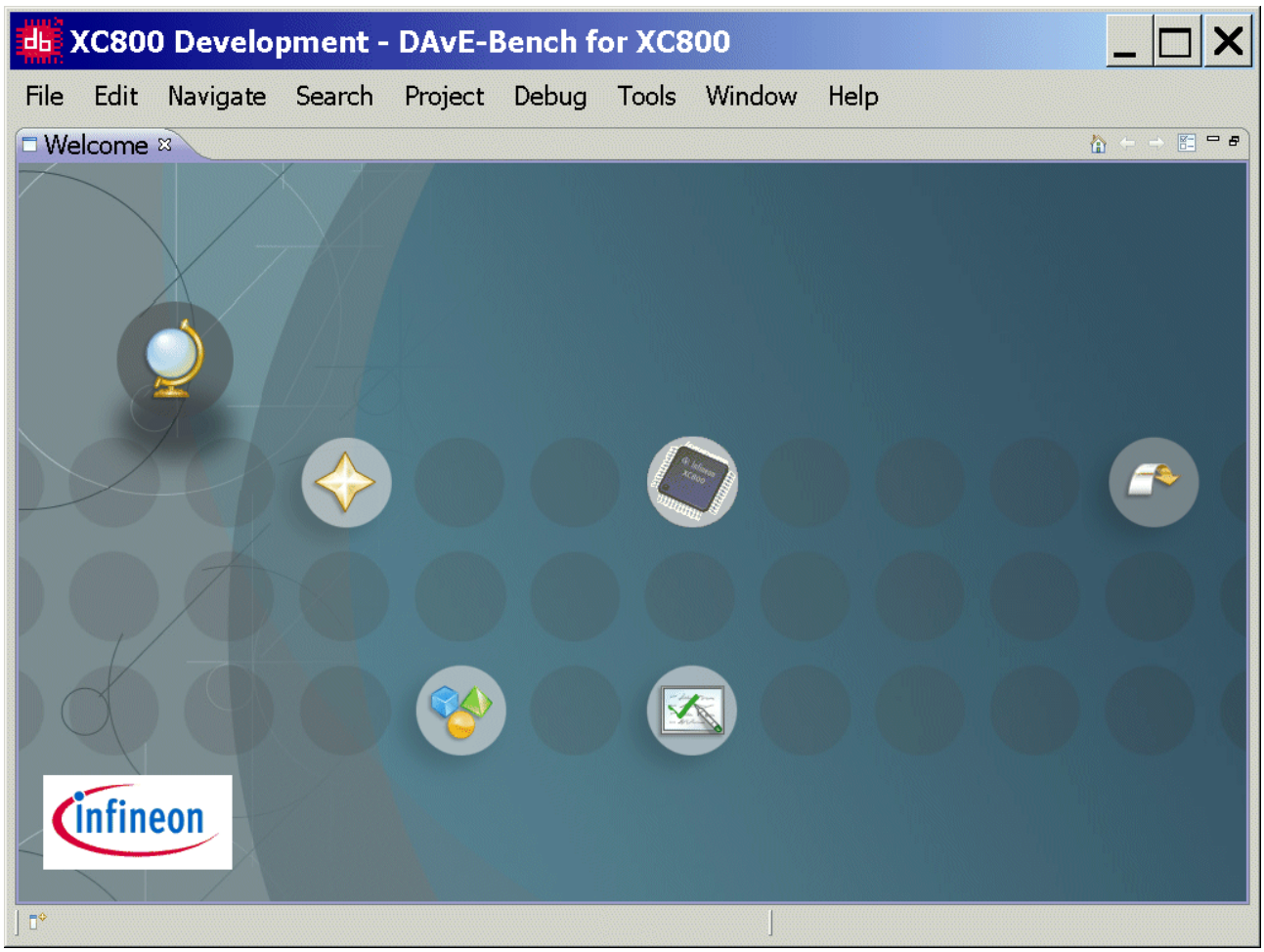
Click Browse…

and select C:**XC8xx**:

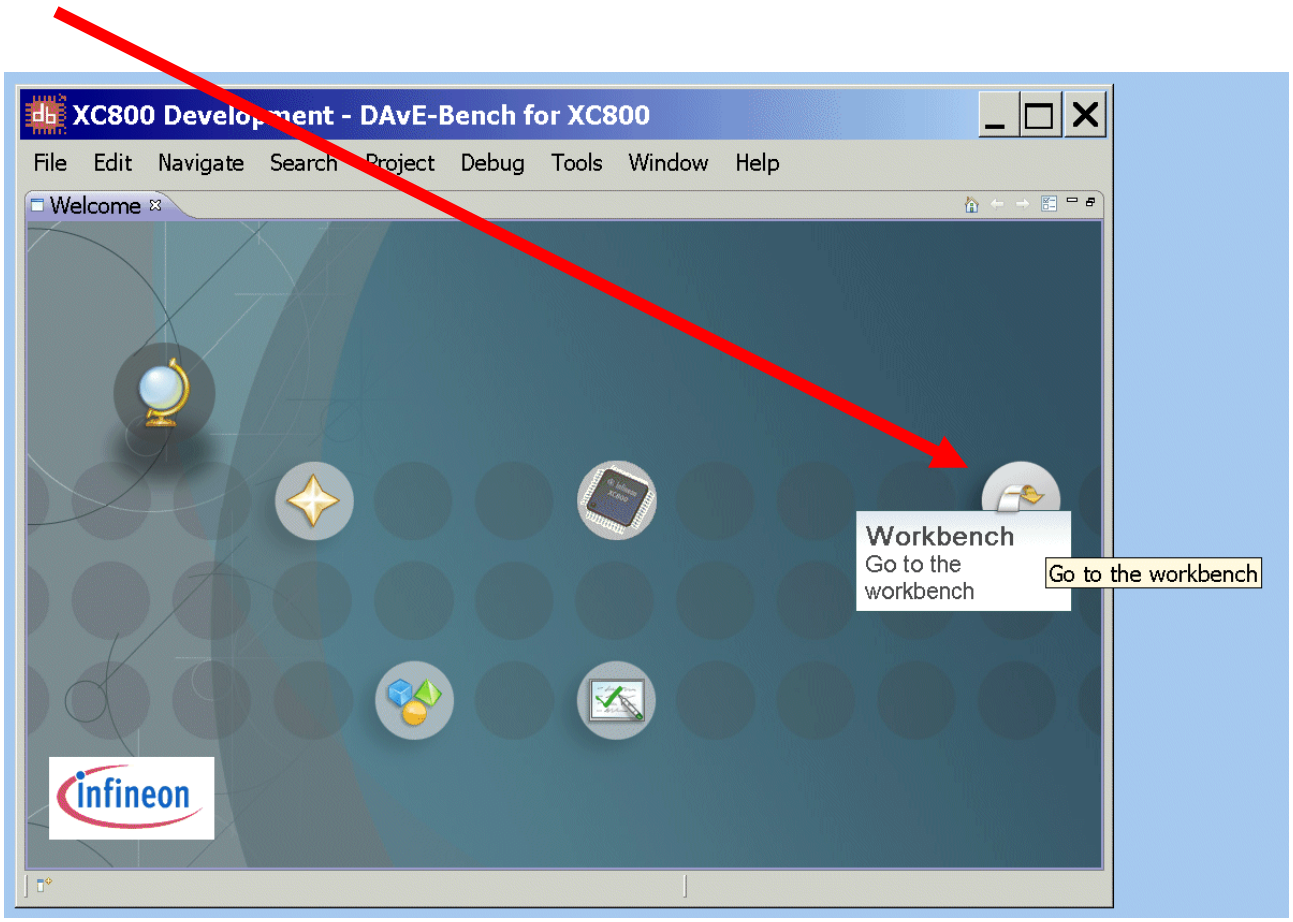


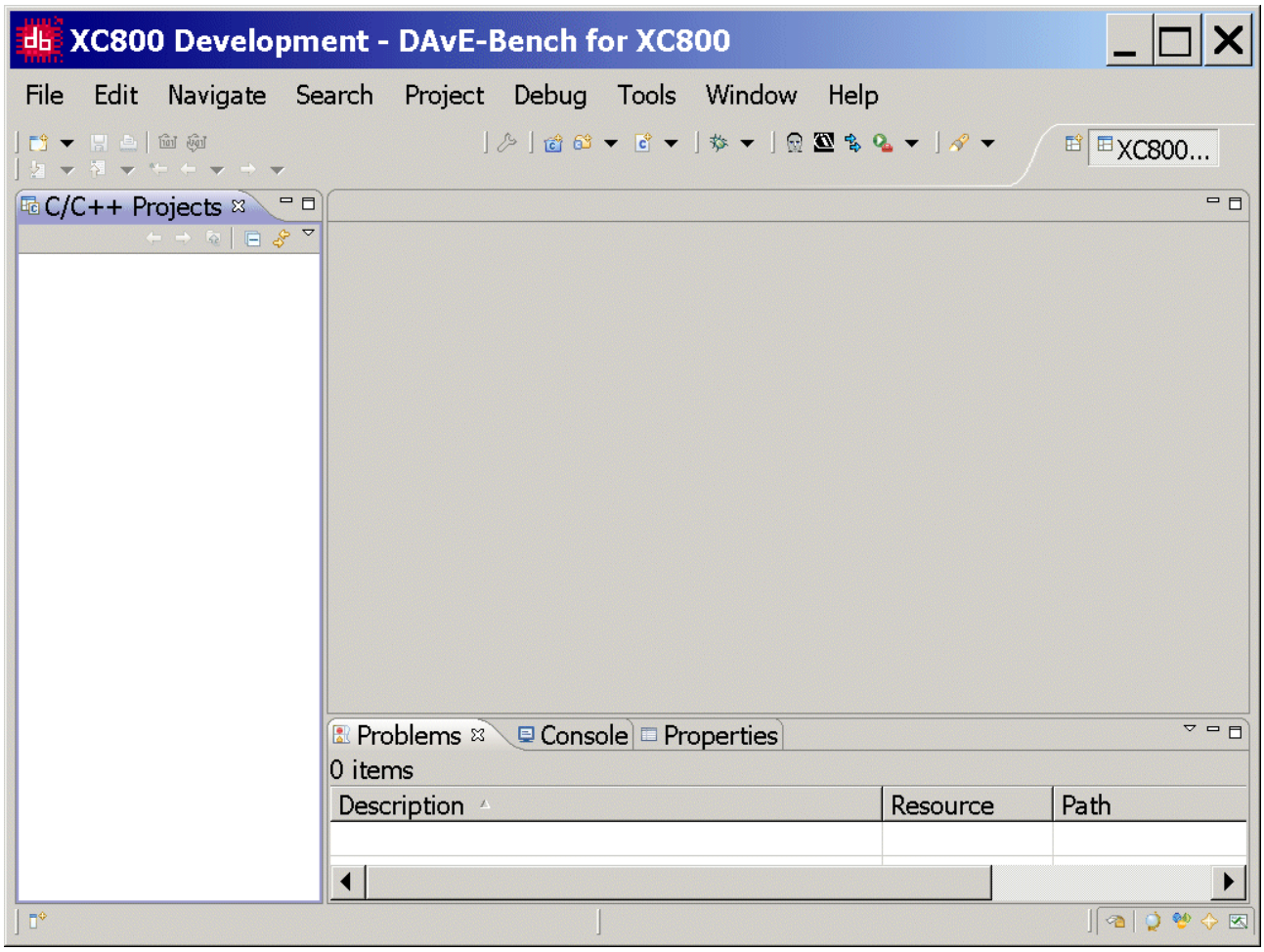
OK - OK


DAvE Bench
for XC800
This DAvE Bench suite includes:
Compiler
Debugger
Loader
U-SPY
Release version 0.1.1.
Loading Workbench
eclipse

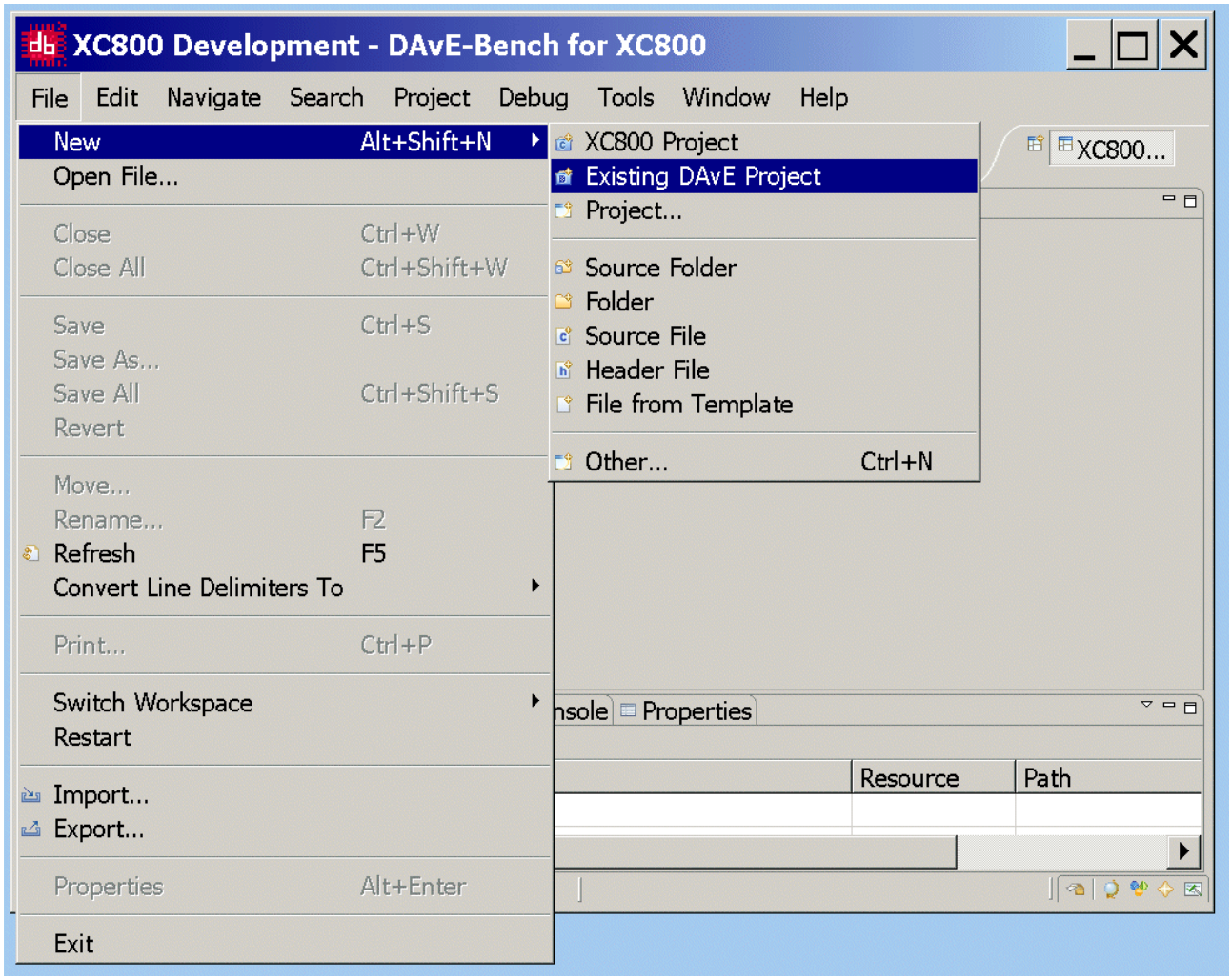

XC800 Development - DAvE-Bench for XC800
File Edit Navigate Search Project Debug Tools Window Help
Welcome

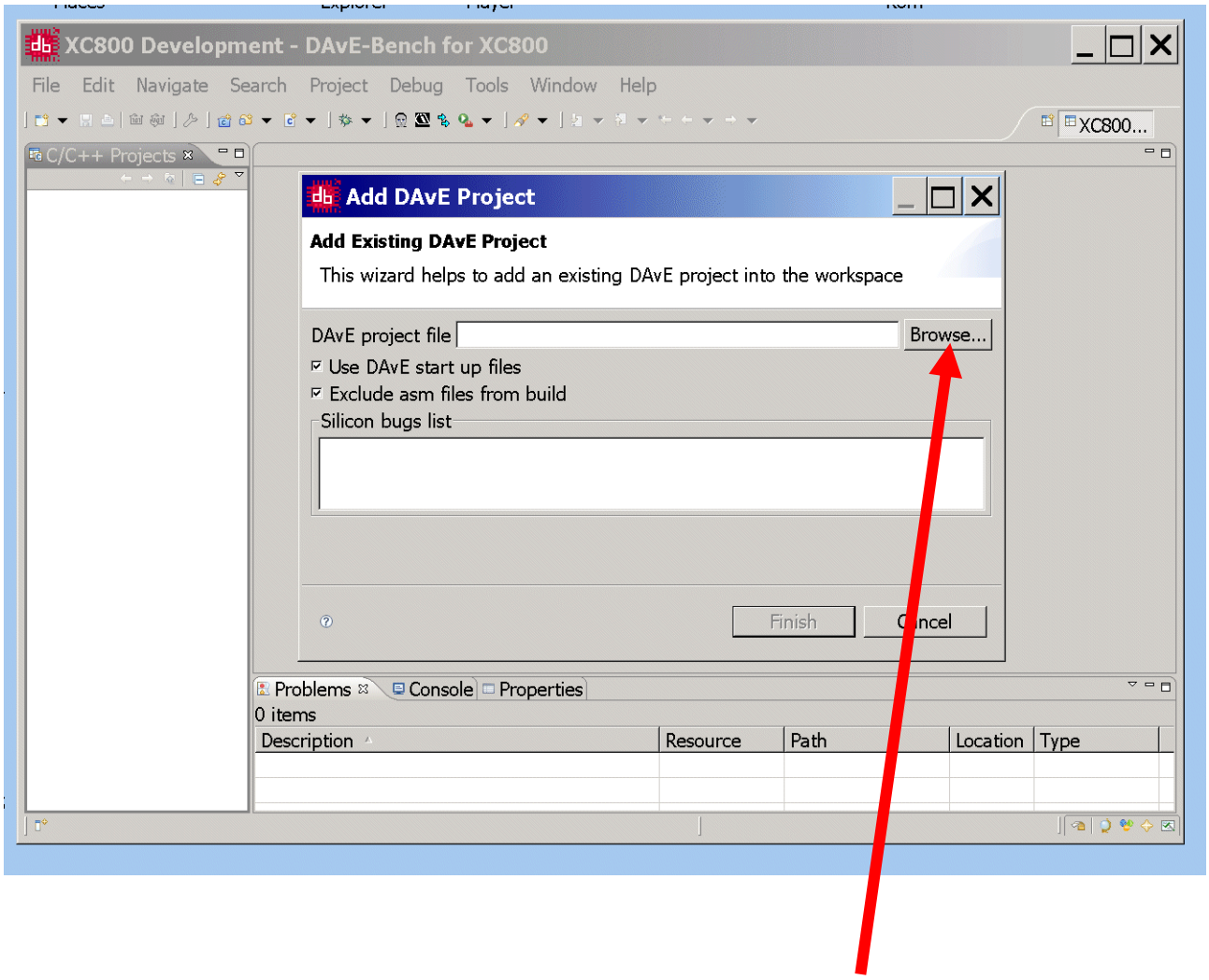
Click “Go to the workbench”:




XC800 Development - DAvE-Bench for XC800
File
Edit
Navigate
Search
Project
Debug
Tools
Window
Help
XC800...
C/C++ Projects
Problems
Console
Properties
0 items
Description
Resource
Path

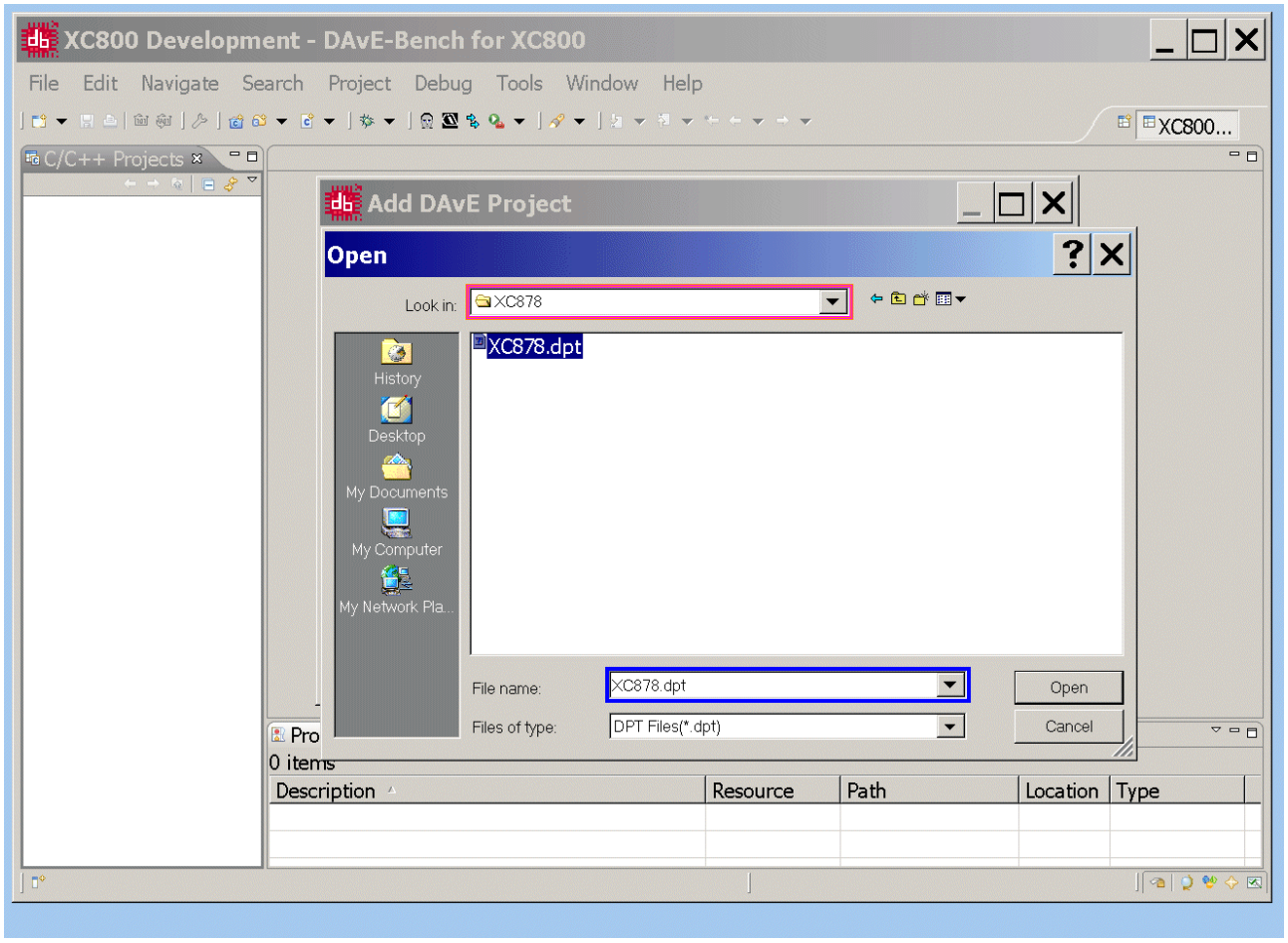
File – New – Existing DAvE Project



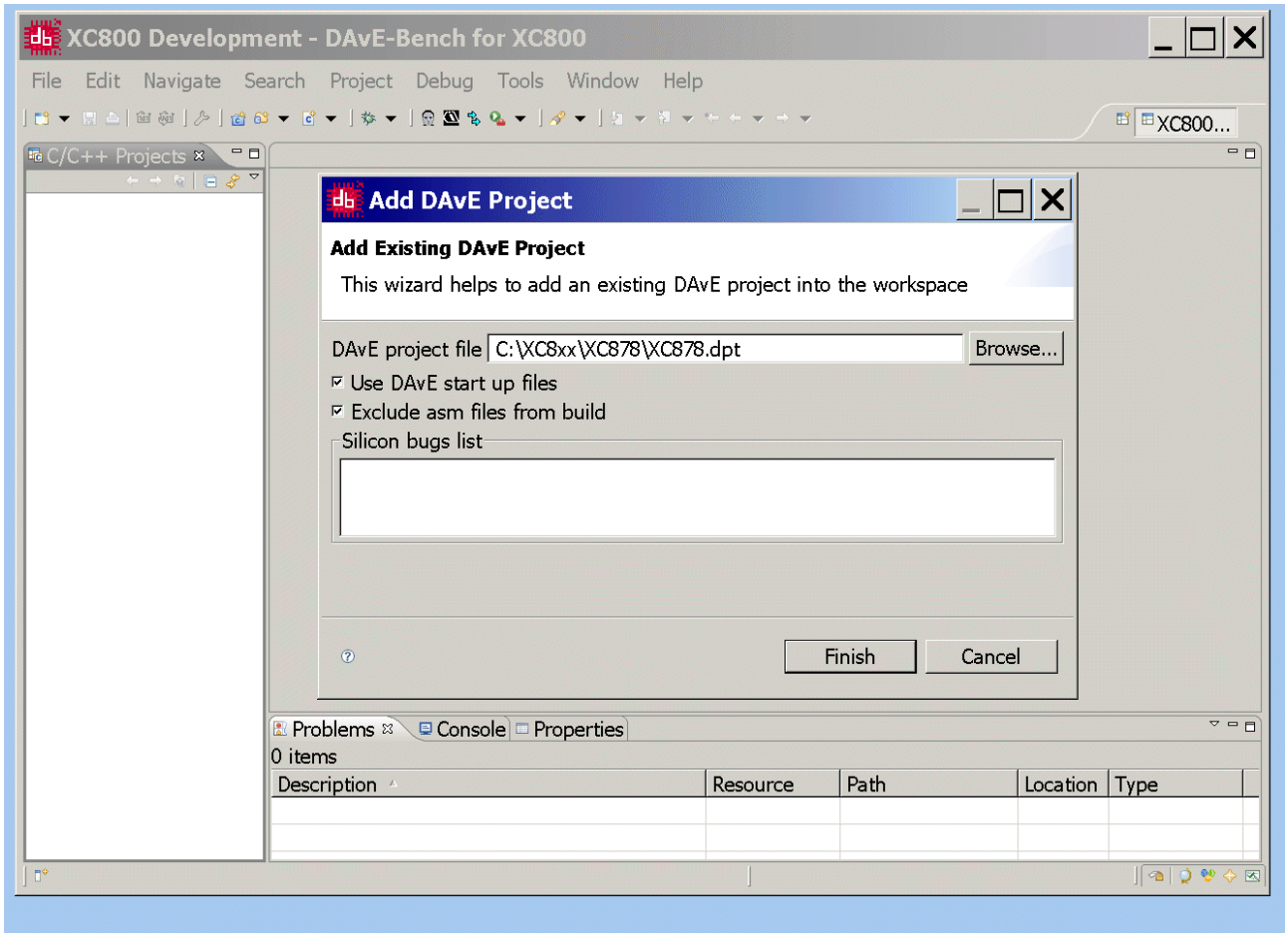


Add DAvE Project: Add Existing DAvE Project: DAvE project file: click Browse…

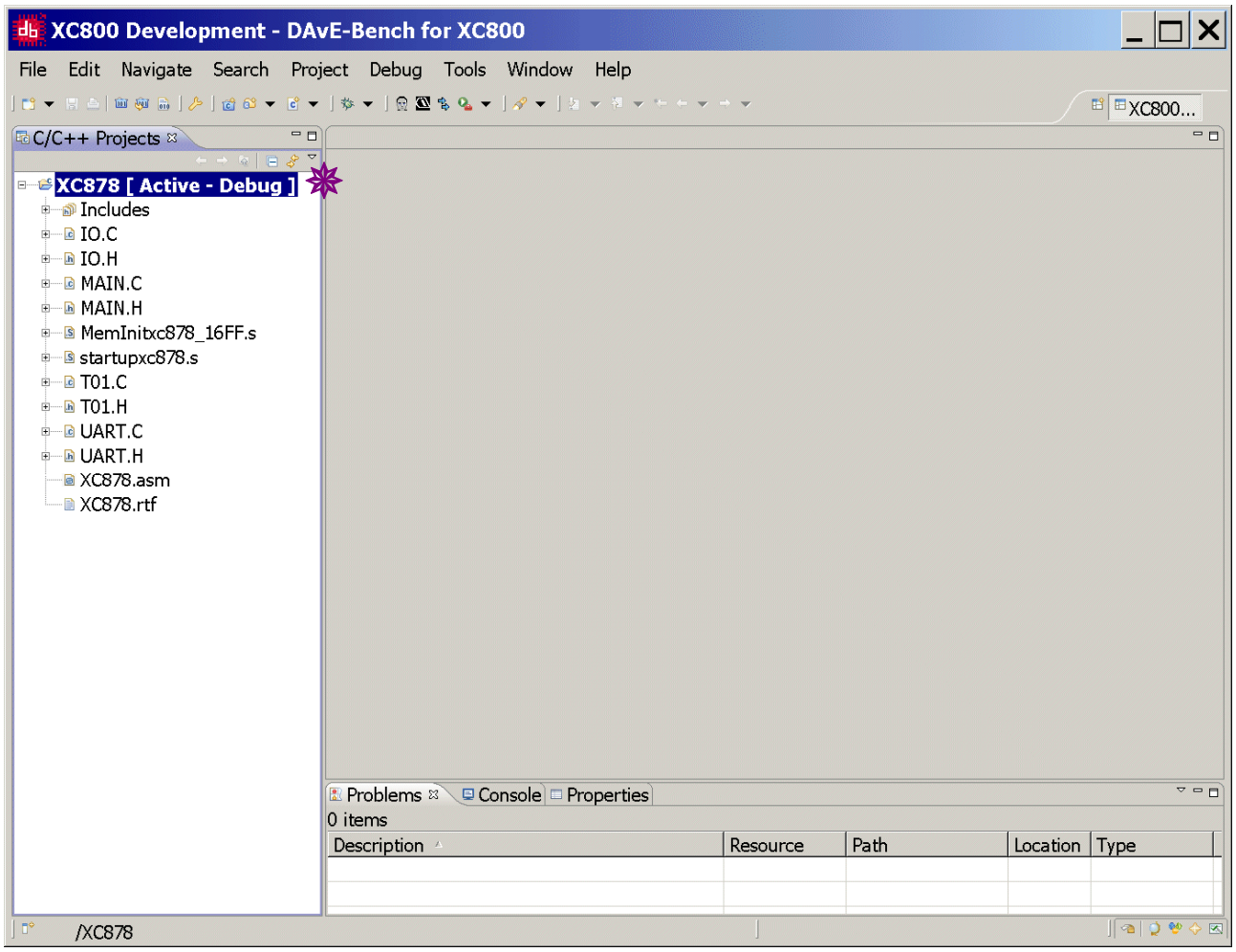
Open: Look in: select C:**XC8xx****XC878**
Open: File name: select **XC878.dpt**



Open


XC800 Development - DAvE-Bench for XC800
File Edit Navigate Search Project Debug Tools Window Help
XC800...
C/C++ Projects
Add DAvE Project
Add Existing DAvE Project
This wizard helps to add an existing DAvE project into the workspace
DAvE project file
C:\XC8xx\XC878\XC878.dpt
Browse...
Use DAvE start up files
Exclude asm files from build
Silicon bugs list
Finish
Cancel
Problems
Console
Properties
0 items
Description
Resource
Path
Location
Type

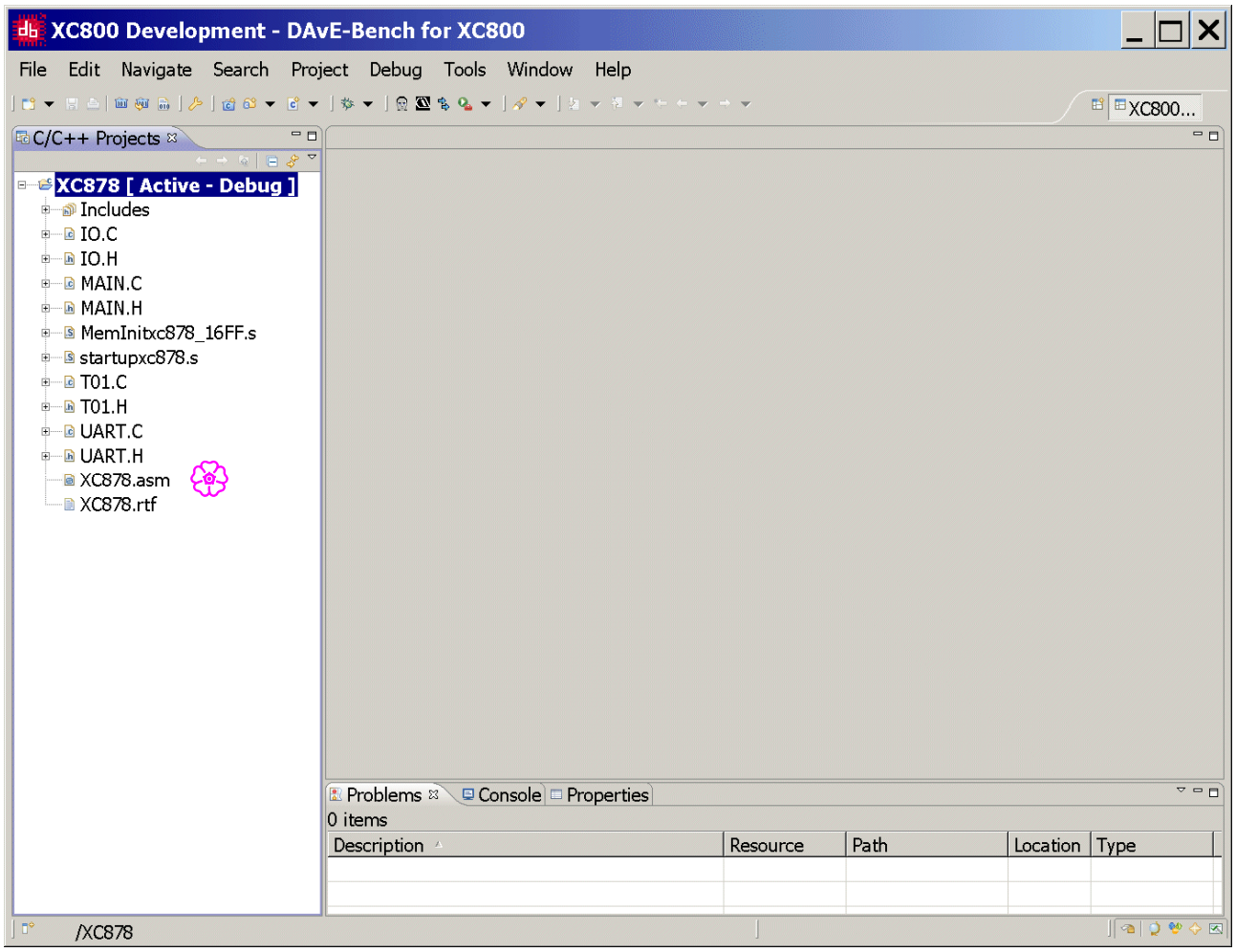
Finish



Note:

Check that the active project is

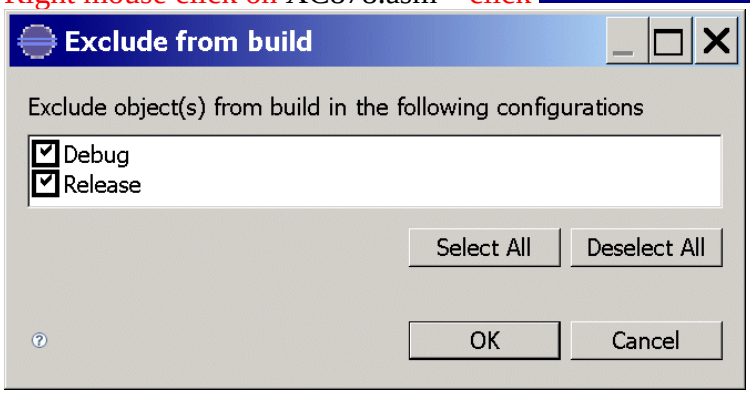
XC888 [Active - Debug] and NOT **XC888 [Active - Release]**.



Note:

Check that XC878.asm is NOT part of the build process:

Right mouse click on XC878.asm – click Exclude from build...



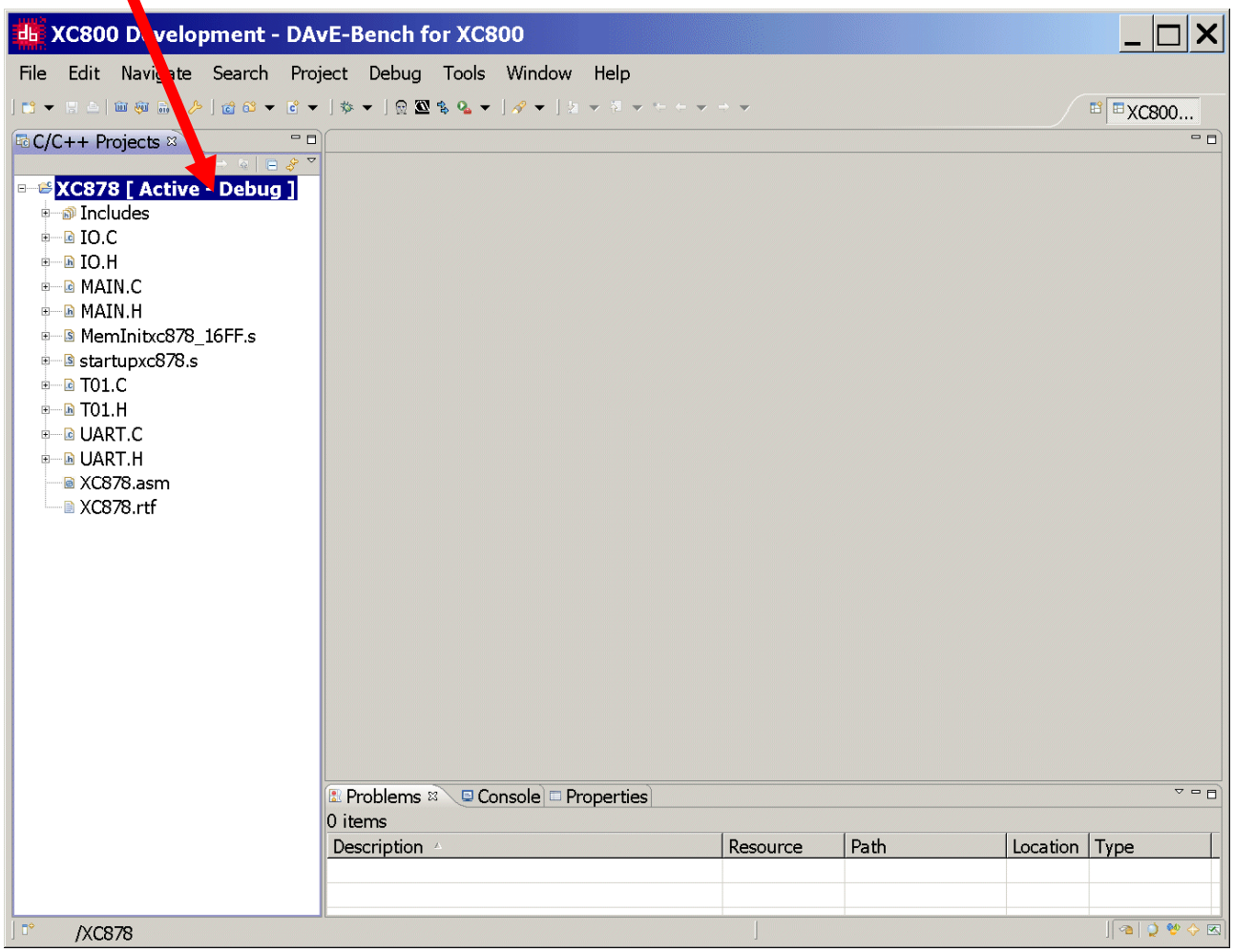
OK

Note:
The DAvE created files can be directly accessed in the IDE (DAvE Bench) or in DAvE.
The user has to make sure that no conflicts or data losses happen.
This can be avoided by saving the files before switching from the IDE to DAvE or vice versa.

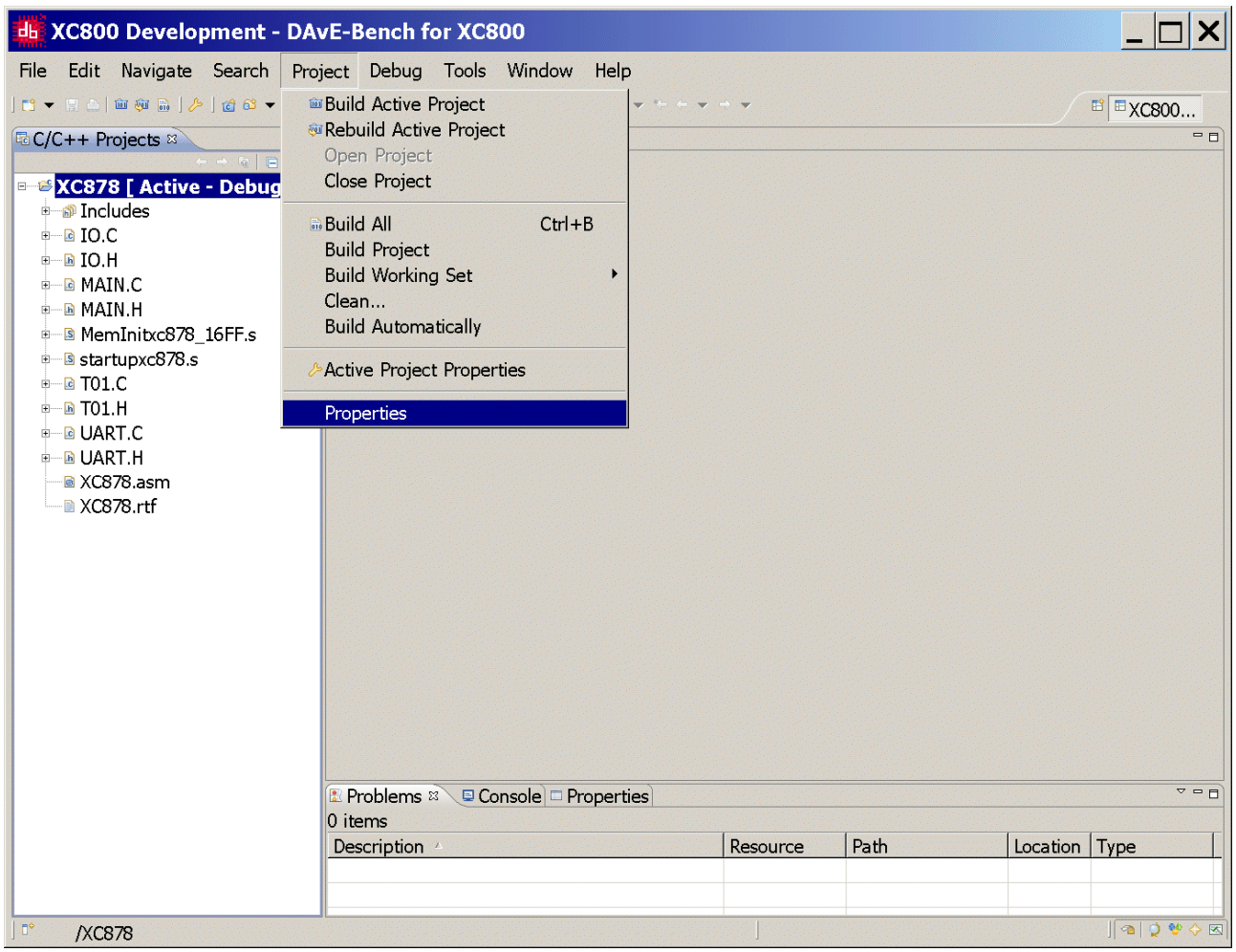
Note:
The following pages/screenshots exist for documentation purposes only.
There is nothing to do!
If you are in a hurry, we suggest you jump to page 138 (Insert your application specific program) and continue working there.

Configure Compiler, Assembler, Linker, Locater, Hex-Converter, Build – Control, Debugger and Utilities:

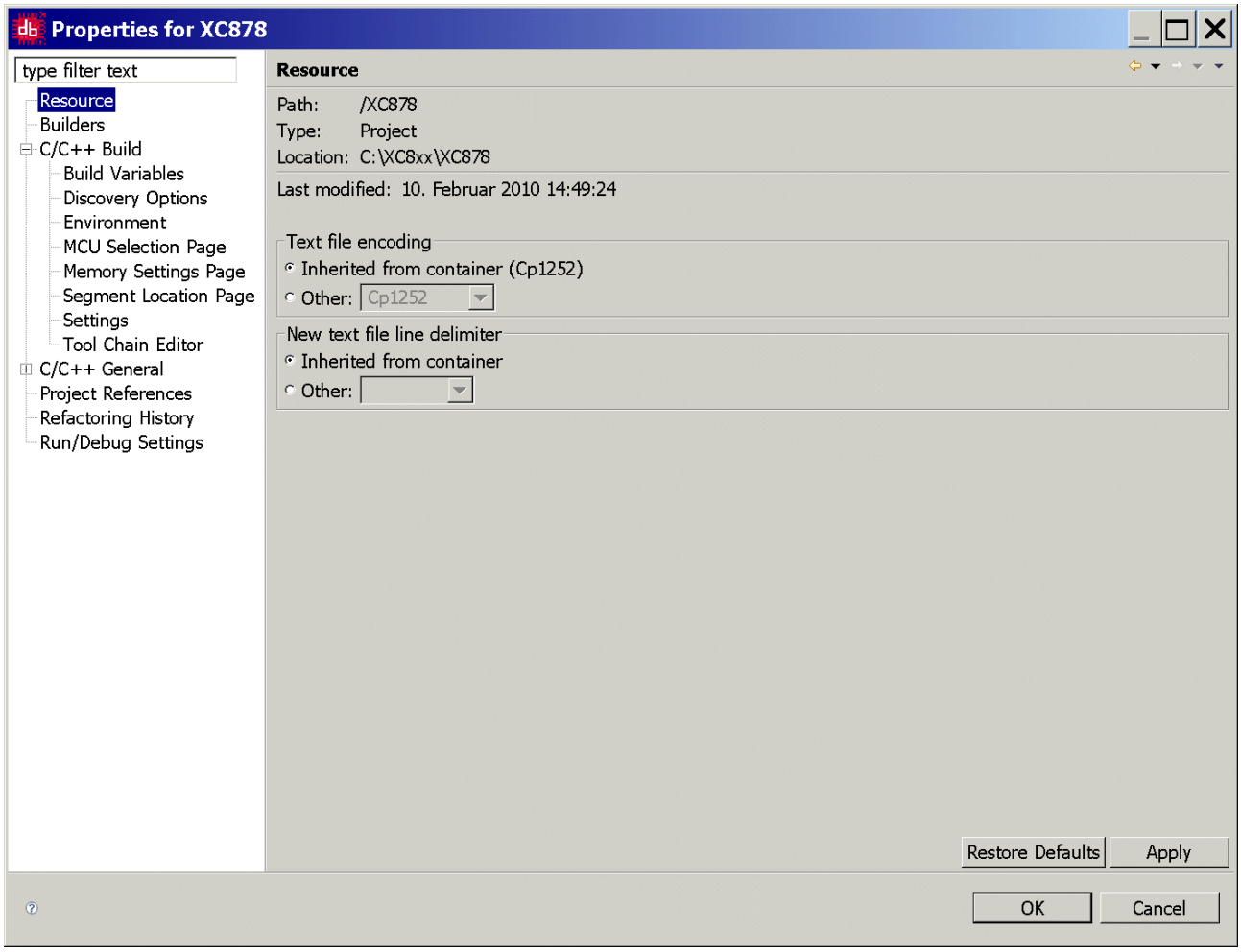
Click/Mark (left mouse click on) **XC888 [Active - Debug]**



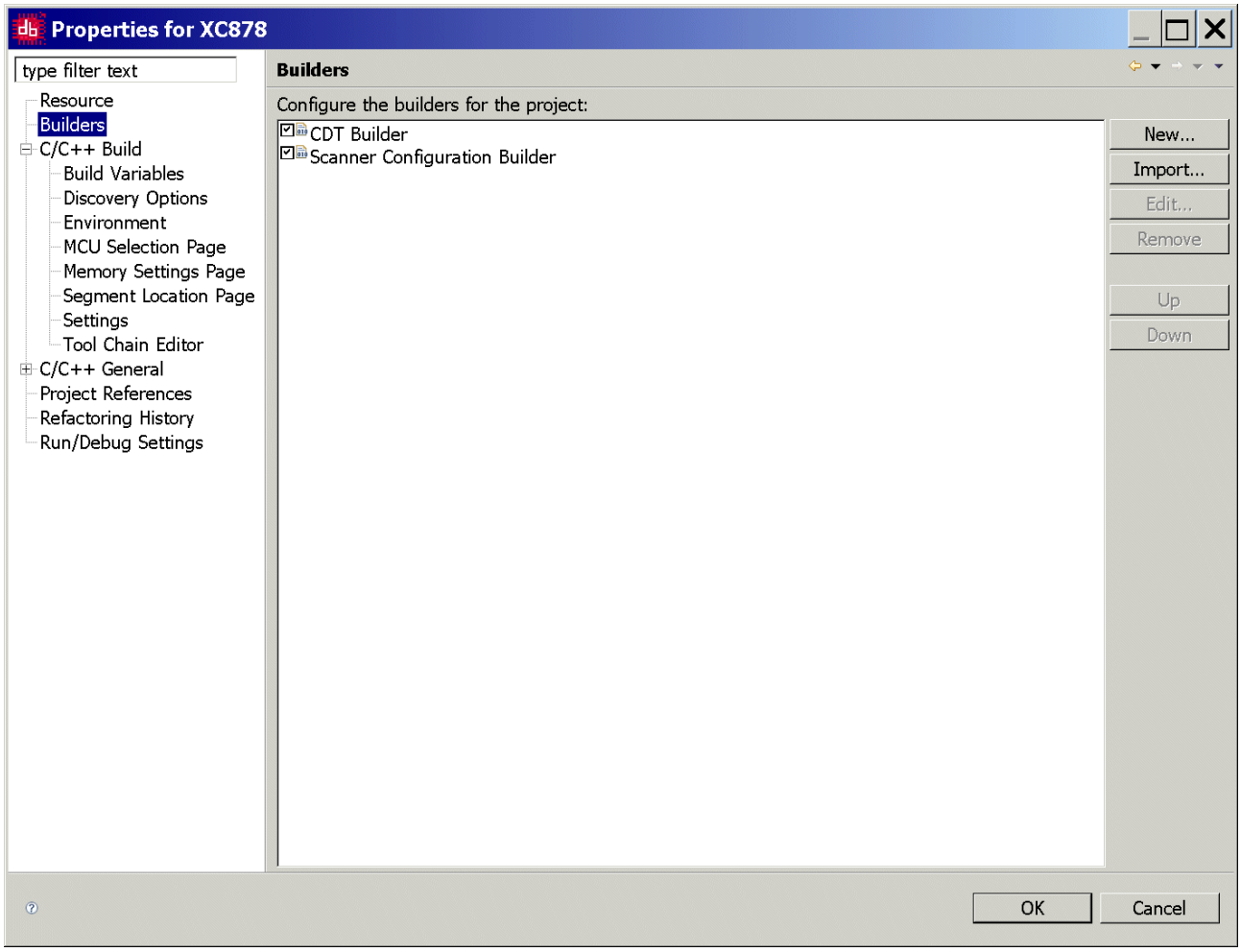
Project - Properties



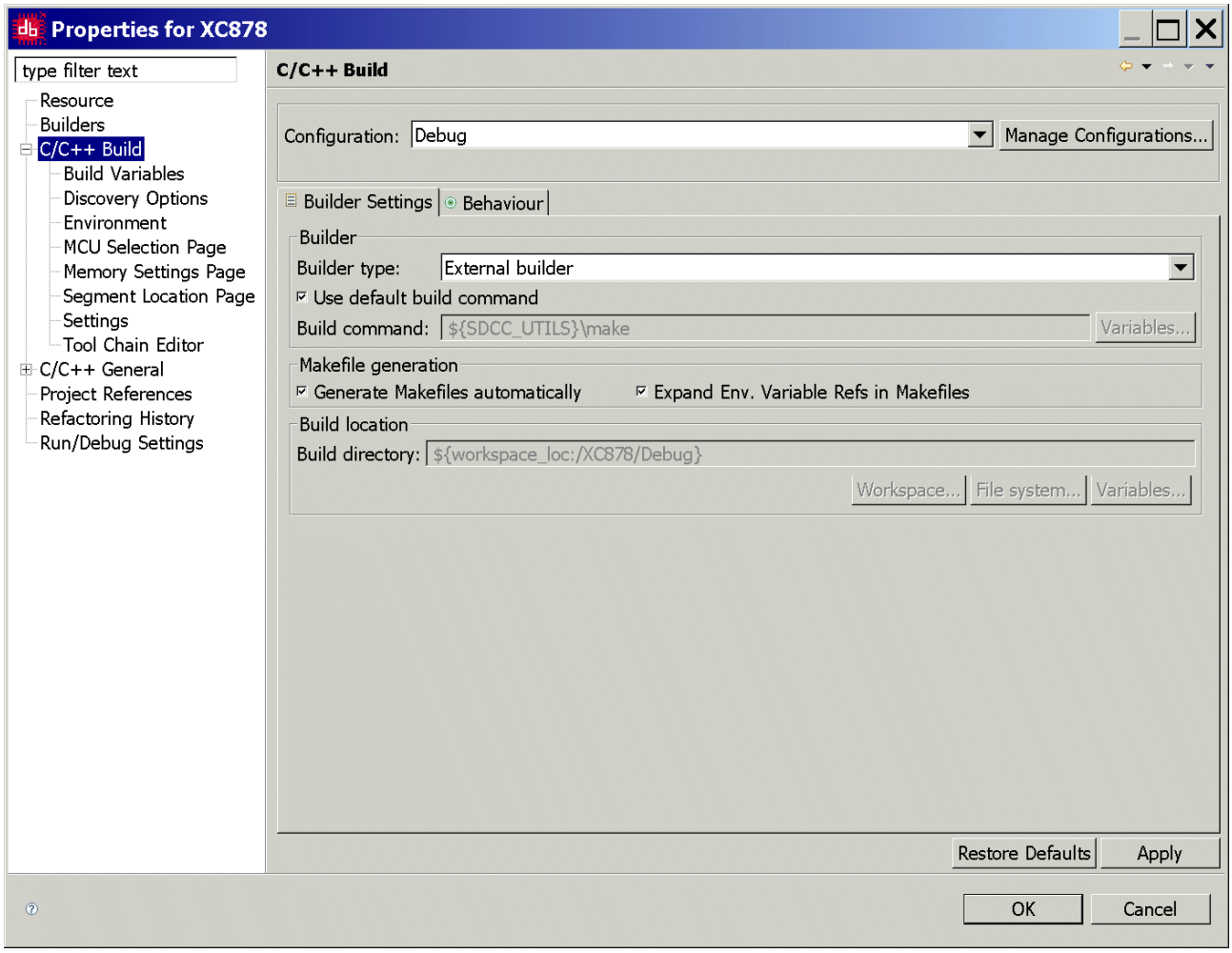
Project – Properties: Resource:



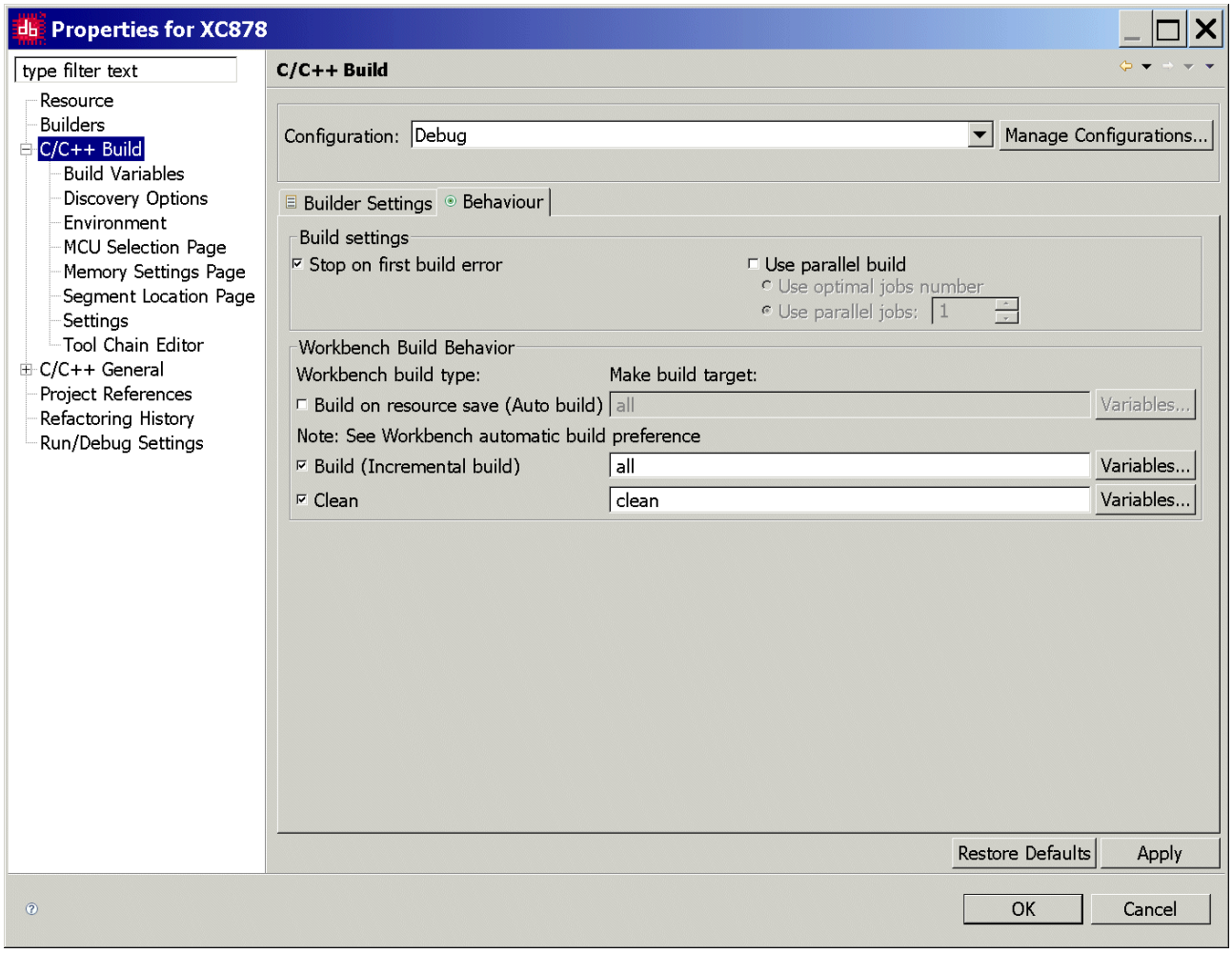
Project – Properties: Builders:



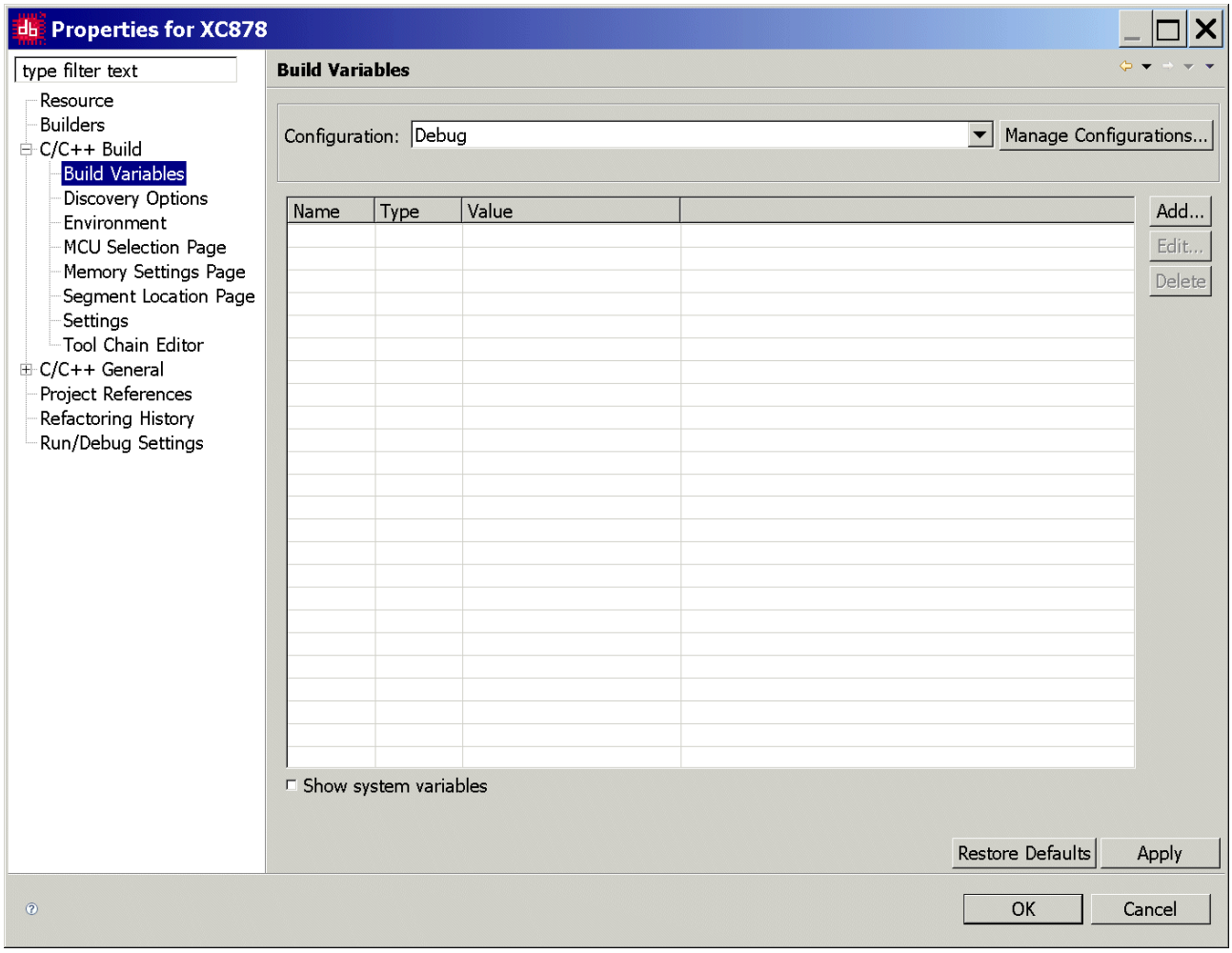
Project – Properties: C/C++ Build: Builder Settings:



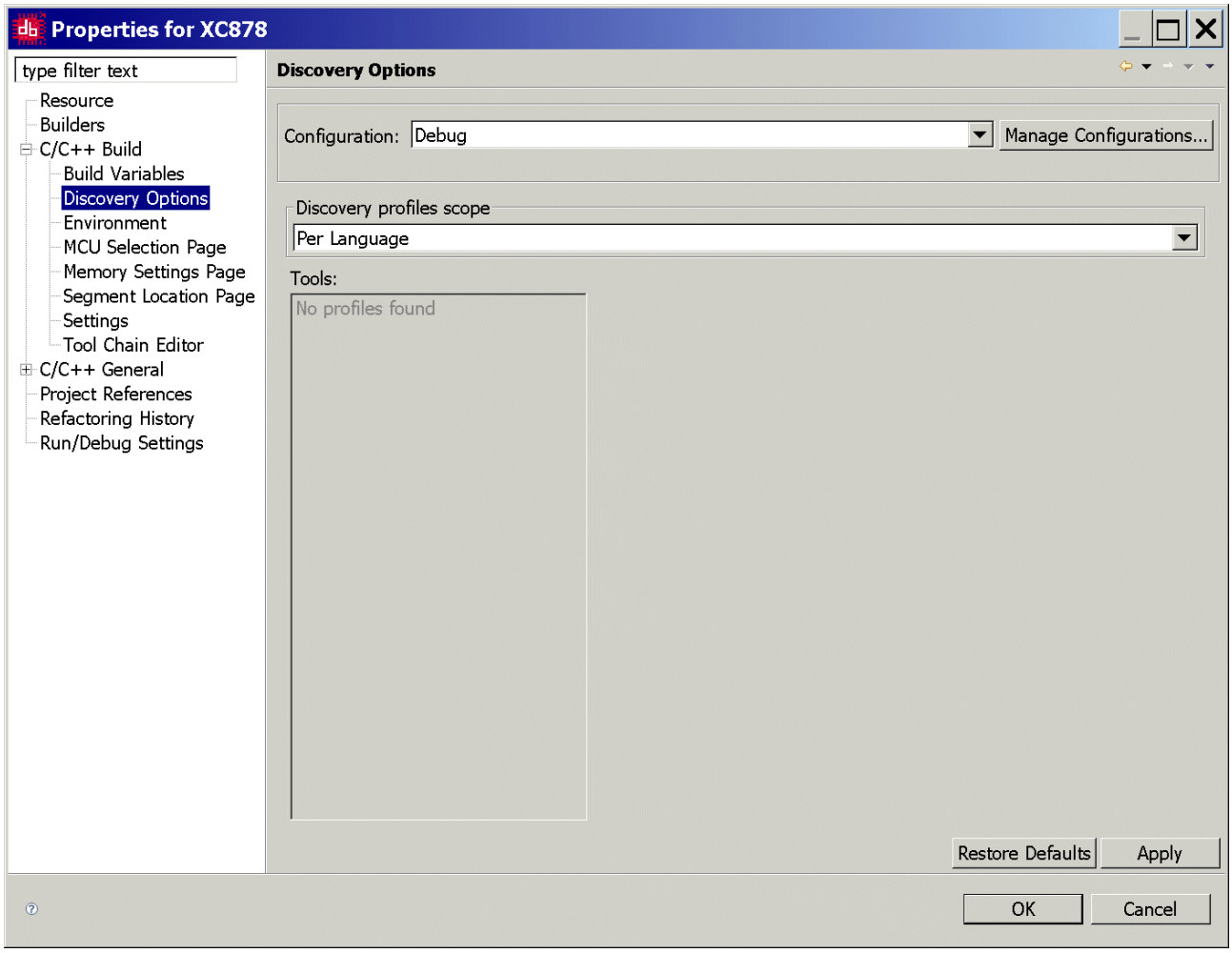
Project – Properties: C/C++ Build: Behaviour:



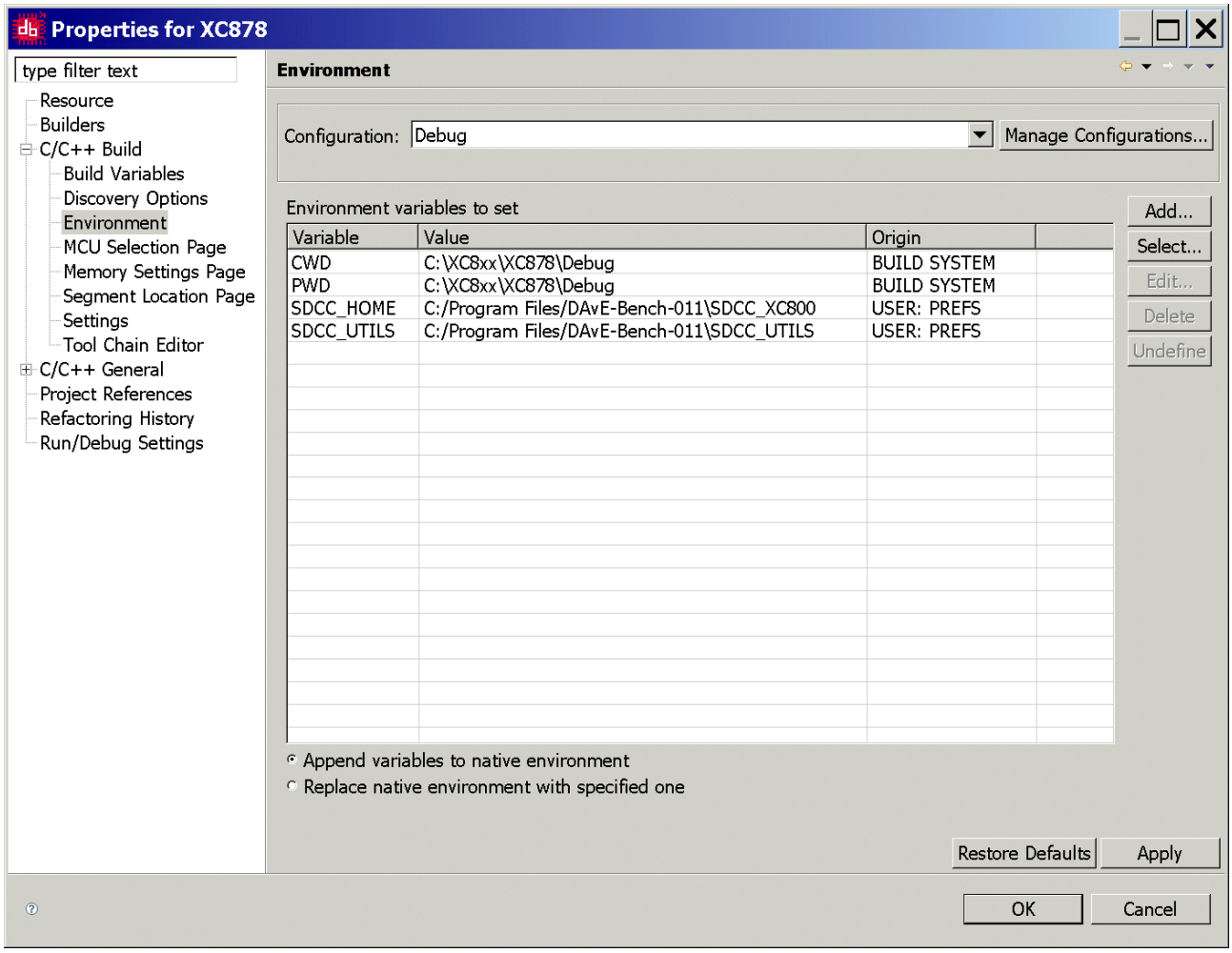
Project – Properties: C/C++ Build: Build Variables:



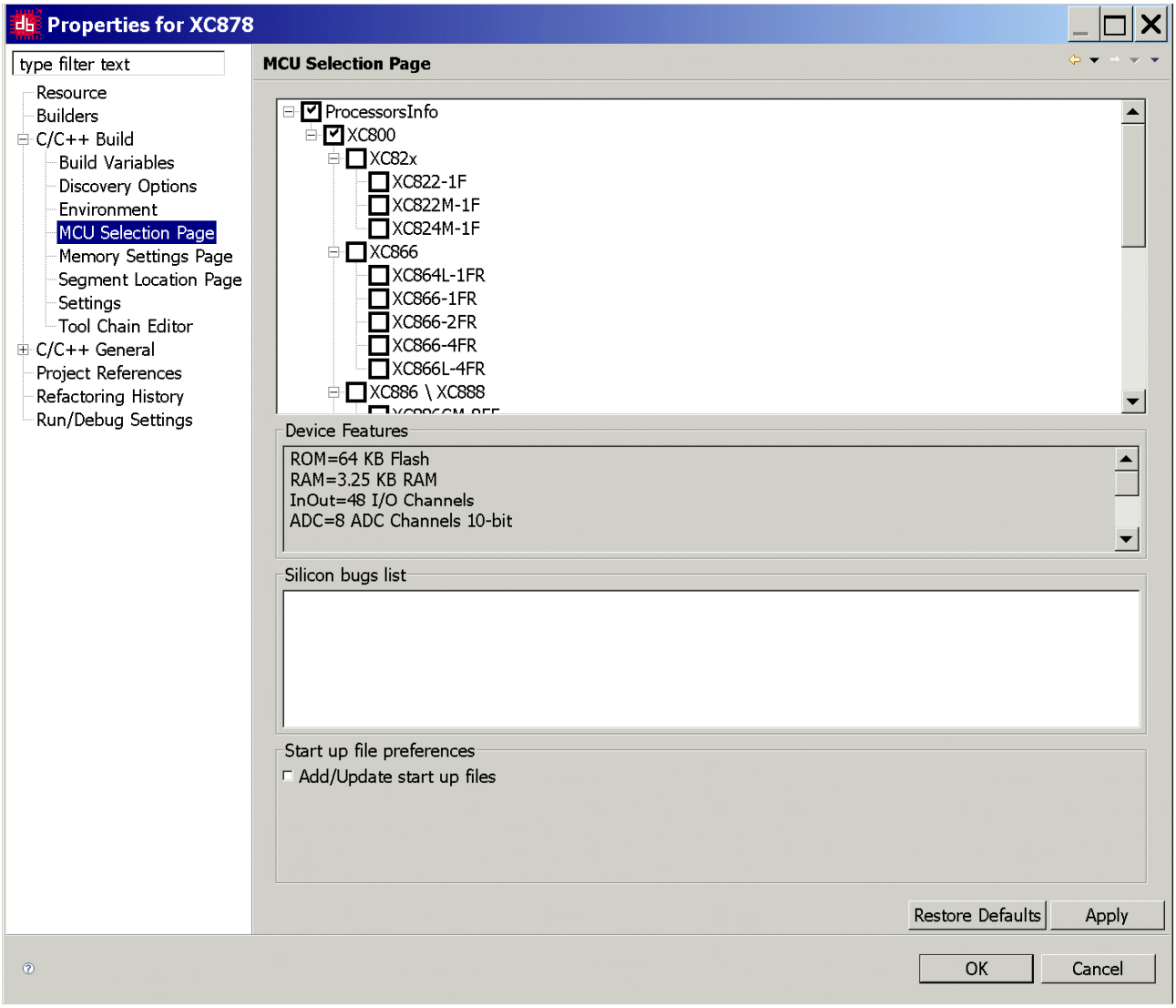
Project – Properties: C/C++ Build: Discovery Options:



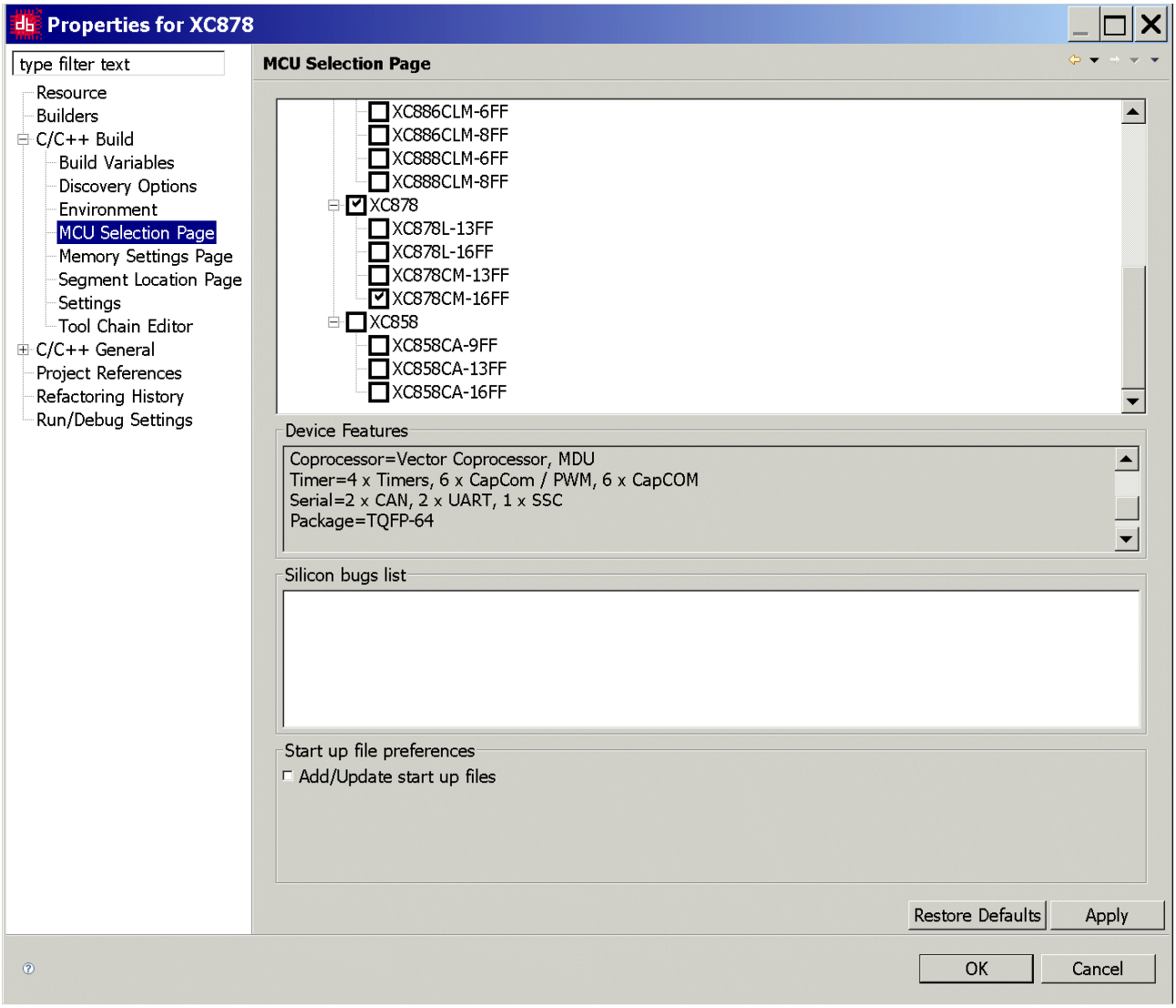
Project – Properties: C/C++ Build: Environment:



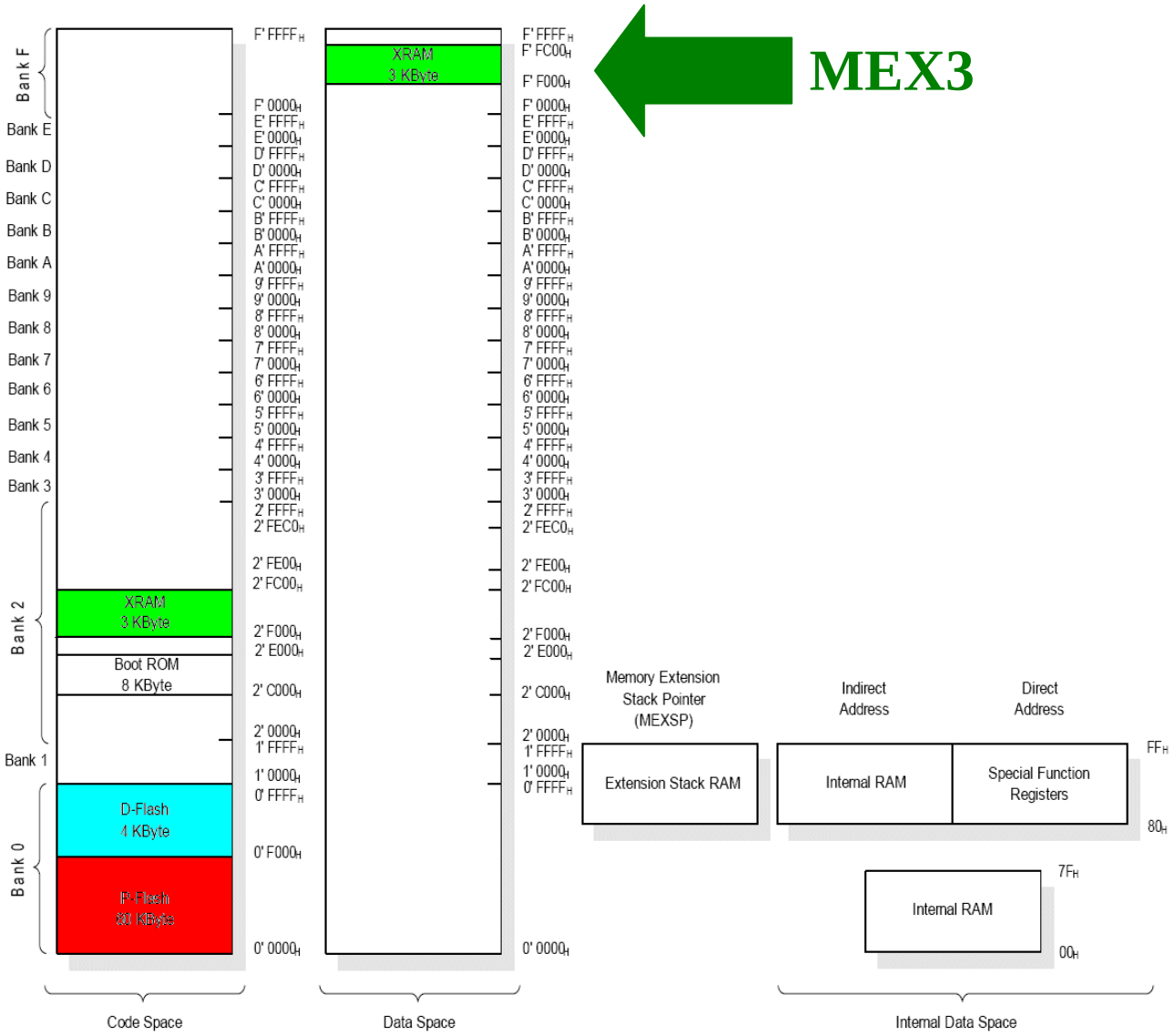
Project – Properties: C/C++ Build: MCU Selection Page (part 1 of 2):



Project – Properties: C/C++ Build: MCU Selection Page (part 2 of 2):



Note:
On-chip Memories,
Additional information: Memory Map (Source: User's Manual):



Additional information: Memory Map (Source: User's Manual):

Note (Source: User's Manual):

The standard amount of addressable program or external data memory (or a Bank) in an 8051 system is 64 Kbytes. The XC800 core supports memory expansion of up to 1 Mbyte and this is enabled by the availability of a Memory Management Unit (MMU) and a Memory Extension Stack. The MMU adds a set of Memory Extension registers (MEX1, MEX2, and MEX3) to control access to the extended memory space by different addressing modes.

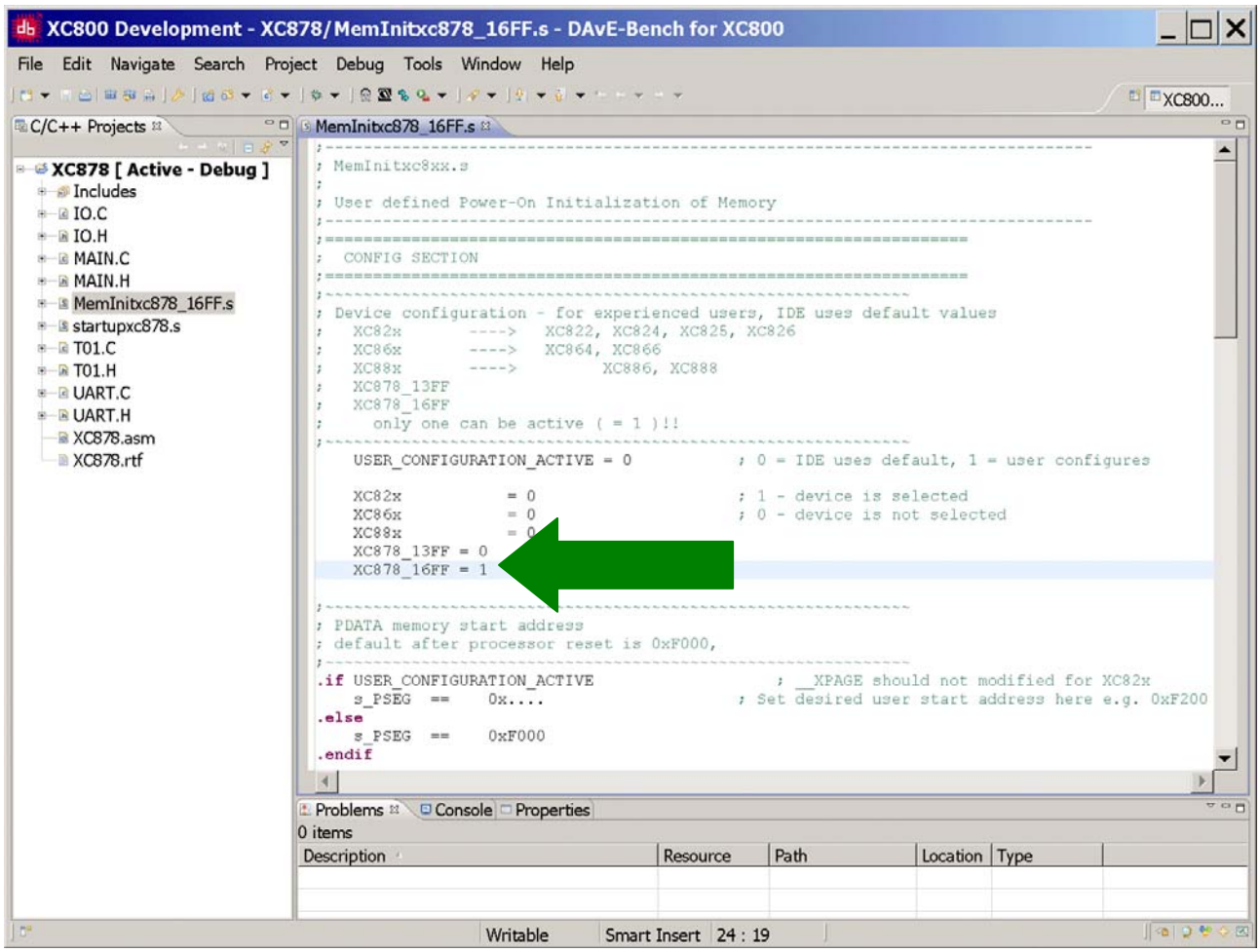
External Data Memory:
The 3-Kbyte XRAM is mapped to both the external data memory area and the program memory area. It can be accessed using both 'MOVX' and 'MOVC' instructions.
The bank where the memories resides must also be selected with the 4-bit XRAM Bank pointer in MEX3.MX (XRAM bank) or the 4-bit Current Bank pointer in MEX1.CB (current bank), depending on bit MXM.

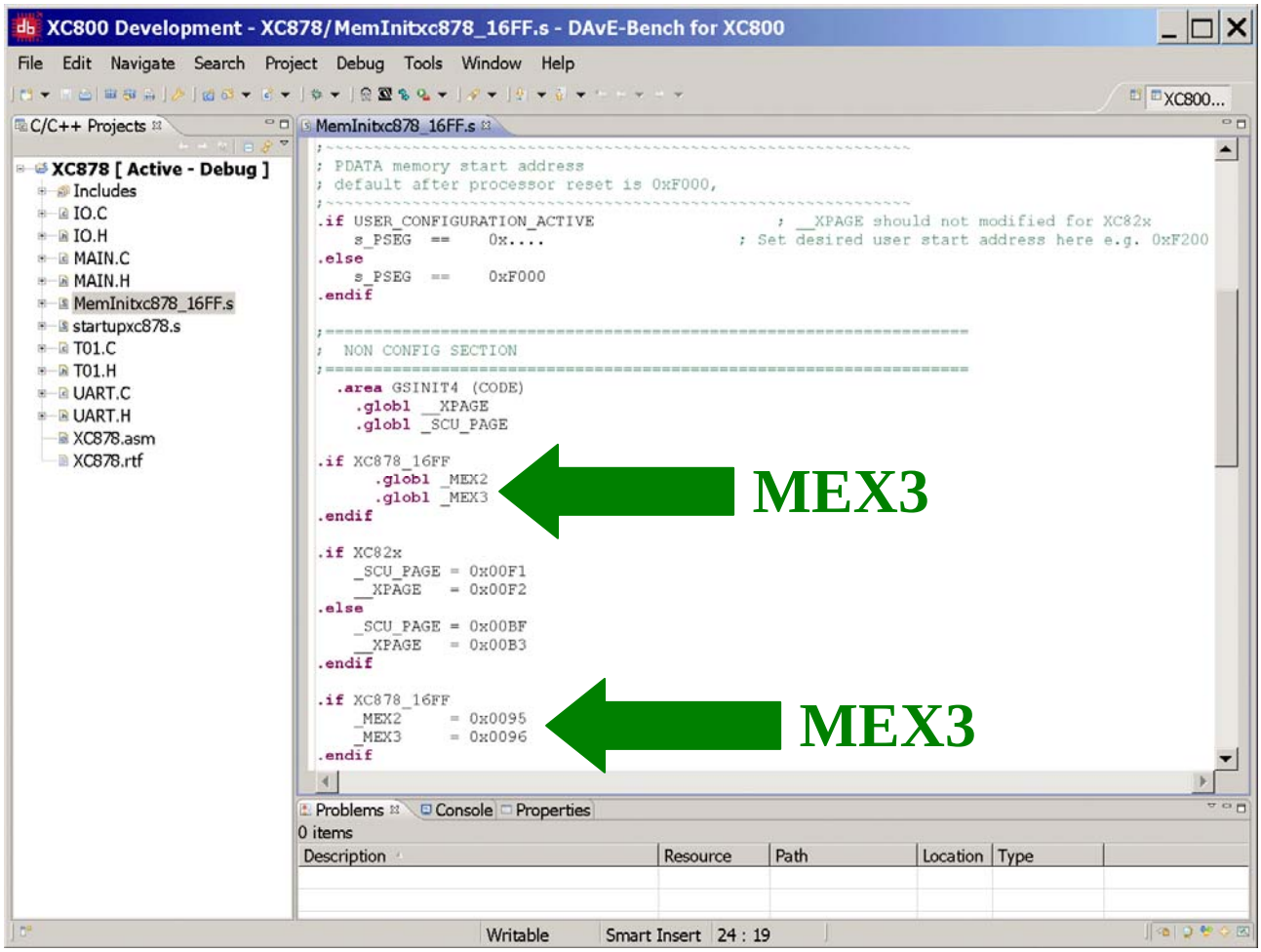
MEX3
Memory Extension Register 3 **Reset Value: 00_H**

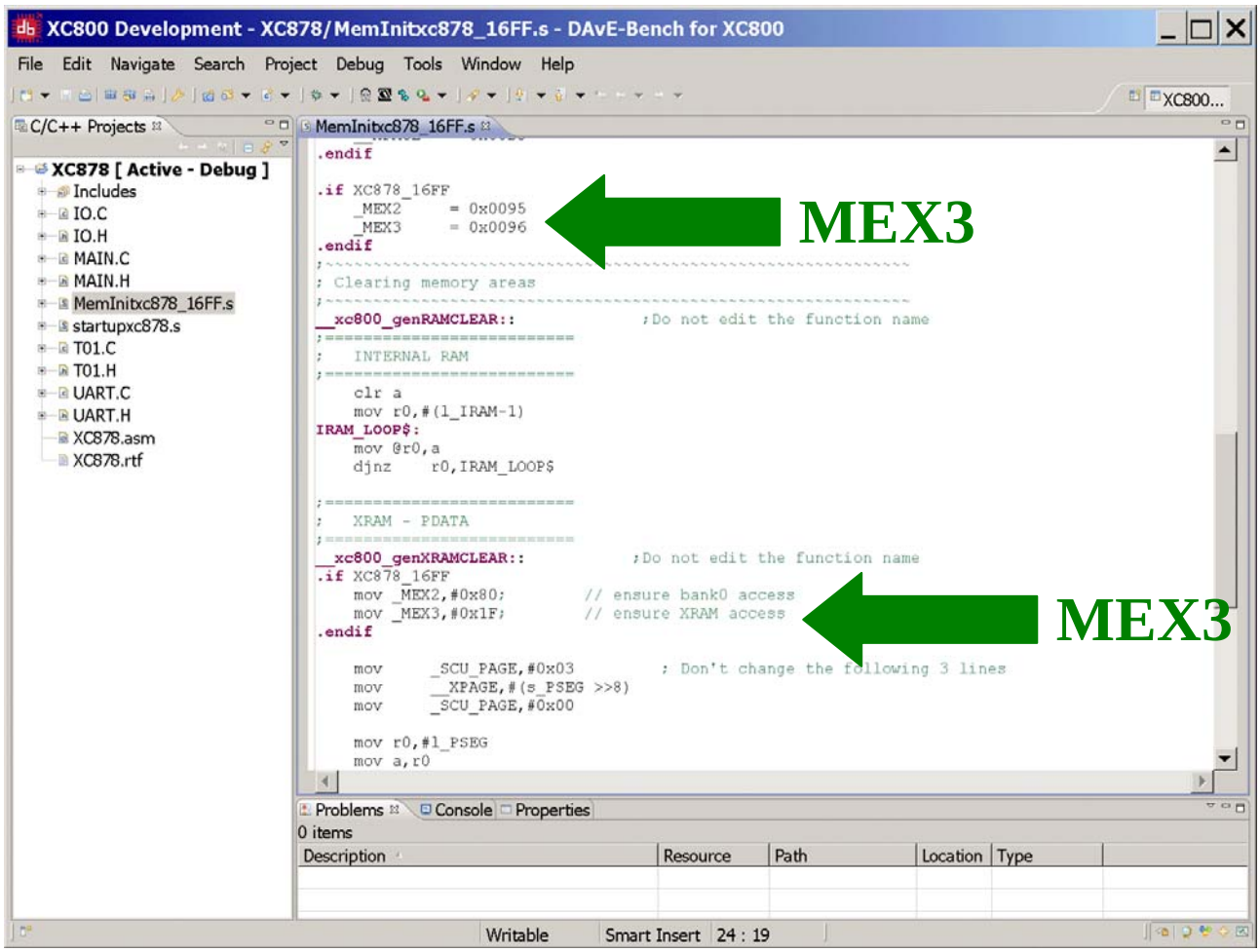
7	6	5	4	3	2	1	0
MCB19	0		MXB19	MXM	MXB[18:16]		
rw	rw		rw	rw	rw		

Field	Bits	Type	Description
MXB[19:16]	4, [2:0]	rw	**XRAM Bank Number**
MXM	3	rw	**XRAM Bank Selector** 0 MOVX access data in the current bank 1 MOVX access data in the Memory XRAM bank
MCB19	7	rw	**Memory Constant Bank Number MSB**
0	[6:5]	rw	**Reserved** Returns 0 if read; should be written with 0.

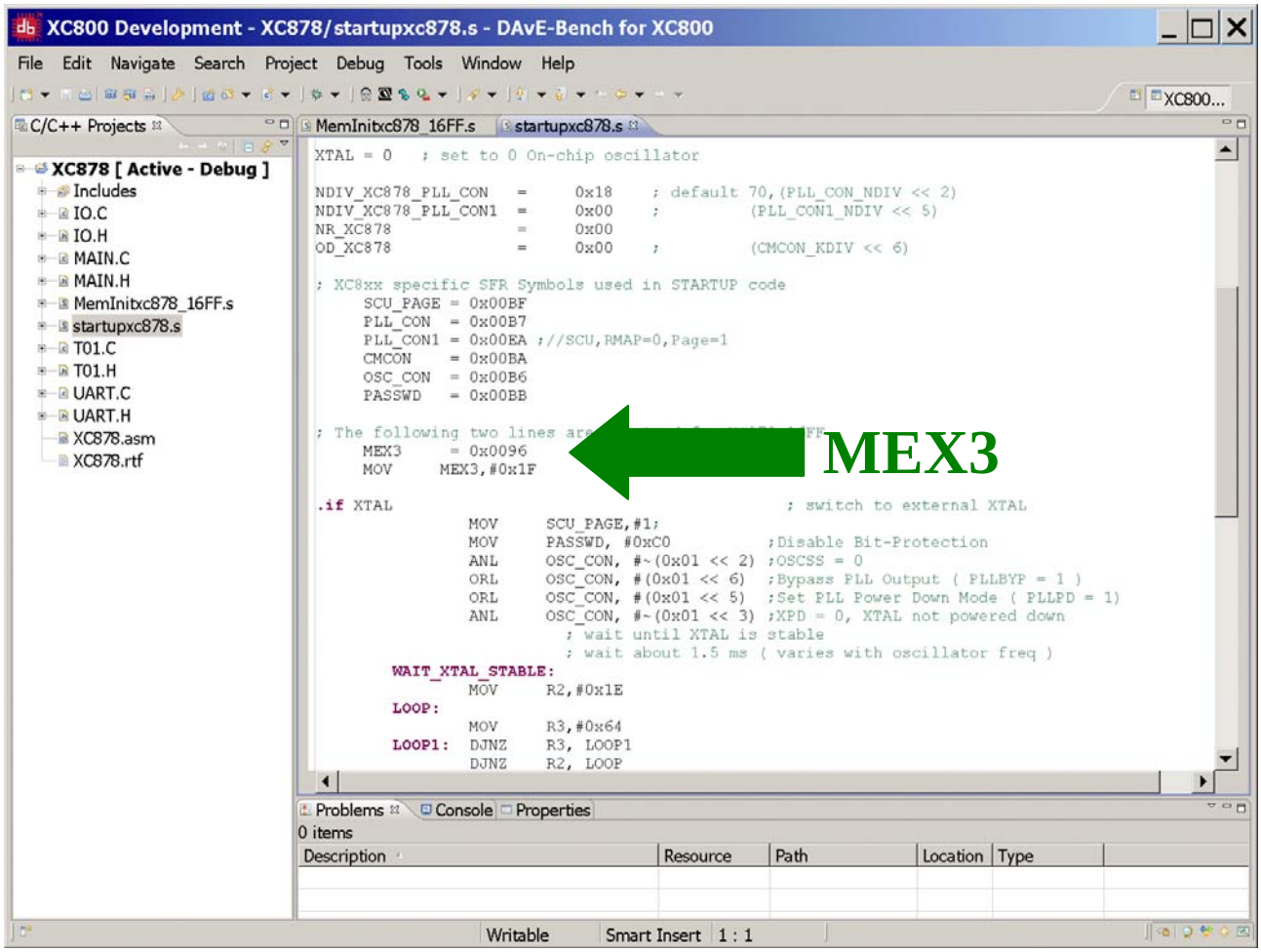
Additional information: Memory Map (Source: MemInitxc878_16FF.s):



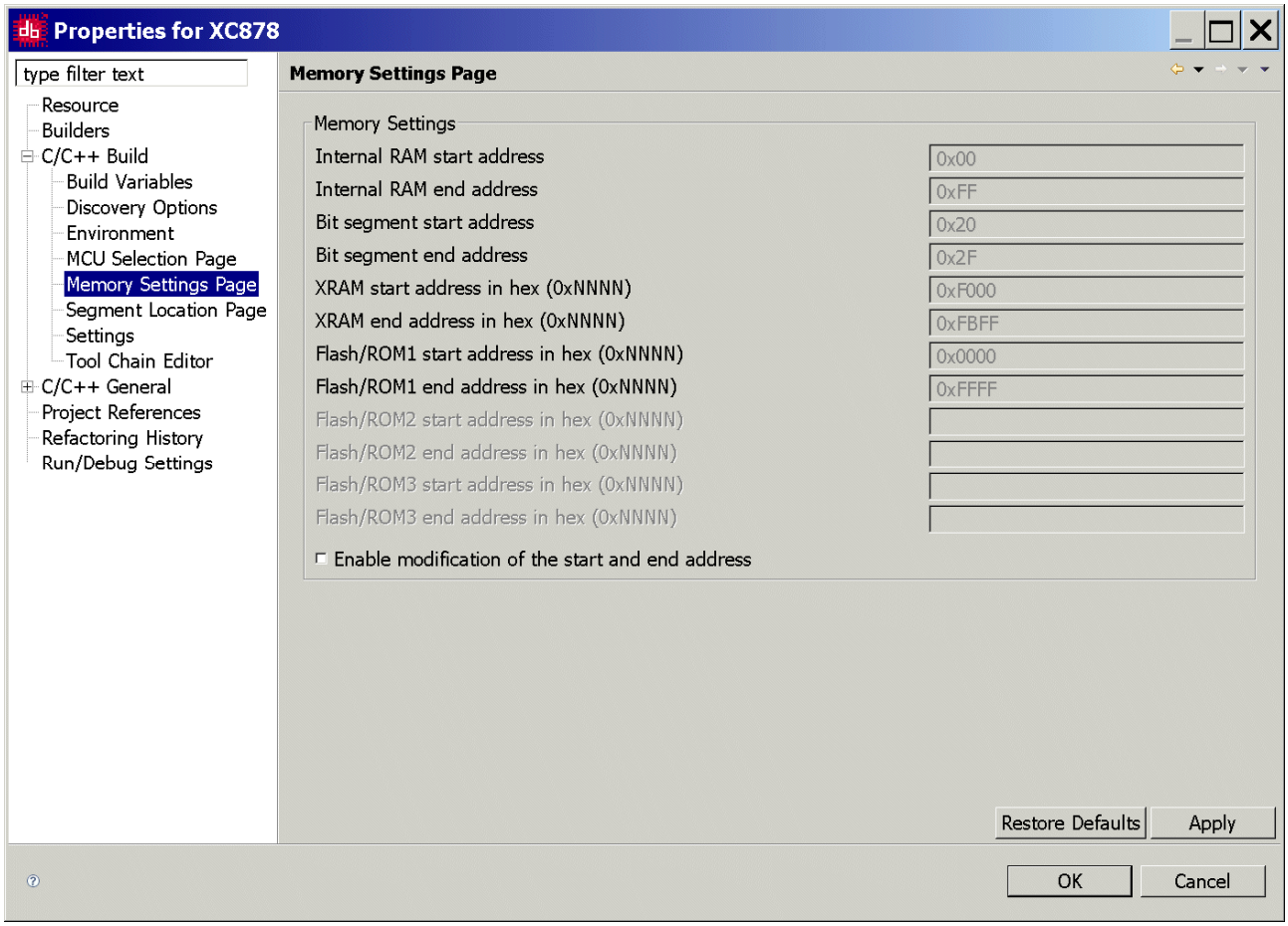

XC800 Development - XC878/MemInitxc878_16FF.s - DAvE-Bench for XC800
File Edit Navigate Search Project Debug Tools Window Help
C/C++ Projects
XC878 [Active - Debug]
Includes
IO.C
IO.H
MAIN.C
MAIN.H
MemInitxc878_16FF.s
startupxc878.s
T01.C
T01.H
UART.C
UART.H
XC878.asm
XC878.rtf
MemInitxc878_16FF.s
; PDATA memory start address
; default after processor reset is 0xF000,
.if USER_CONFIGURATION_ACTIVE ; __XPAGE should not modified for XC82x
s_PSEG == 0x.... ; Set desired user start address here e.g. 0xF200
.else
s_PSEG == 0xF000
.endif
; NON CONFIG SECTION
.area GSINIT4 (CODE)
.globl __XPAGE
.globl _SCU_PAGE
.if XC878_16FF
.globl _MEX2
.globl _MEX3
.endif
MEX3
.if XC82x
_SCU_PAGE = 0x00F1
__XPAGE = 0x00F2
.else
_SCU_PAGE = 0x00BF
__XPAGE = 0x00B3
.endif
.if XC878_16FF
_MEX2 = 0x0095
_MEX3 = 0x0096
.endif
MEX3
Problems Console Properties
0 items
Description Resource Path Location Type
Writable Smart Insert 24 : 19


XC800 Development - XC878/MemInitxc878_16FF.s - DAvE-Bench for XC800
File Edit Navigate Search Project Debug Tools Window Help
C/C++ Projects
XC878 [Active - Debug]
Includes
IO.C
IO.H
MAIN.C
MAIN.H
MemInitxc878_16FF.s
startupxc878.s
T01.C
T01.H
UART.C
UART.H
XC878.asm
XC878.rtf
MemInitxc878_16FF.s
.endif
.if XC878_16FF
_MEX2 = 0x0095
_MEX3 = 0x0096
.endif
; Clearing memory areas
__xc800_genRAMCLEAR:: ;Do not edit the function name
; INTERNAL RAM
clr a
mov r0,#(l_IRAM-1)
IRAM_LOOP$:
mov @r0,a
djnz r0,IRAM_LOOP$
; XRAM - PDATA
__xc800_genXRAMCLEAR:: ;Do not edit the function name
.if XC878_16FF
mov _MEX2,#0x80; // ensure bank0 access
mov _MEX3,#0x1F; // ensure XRAM access
.endif
mov _SCU_PAGE,#0x03 ; Don't change the following 3 lines
mov __XPAGE,#(s_PSEG >>8)
mov _SCU_PAGE,#0x00
mov r0,#l_PSEG
mov a,r0
MEX3
MEX3
Problems Console Properties
0 items
Description Resource Path Location Type
Writable Smart Insert 24 : 19

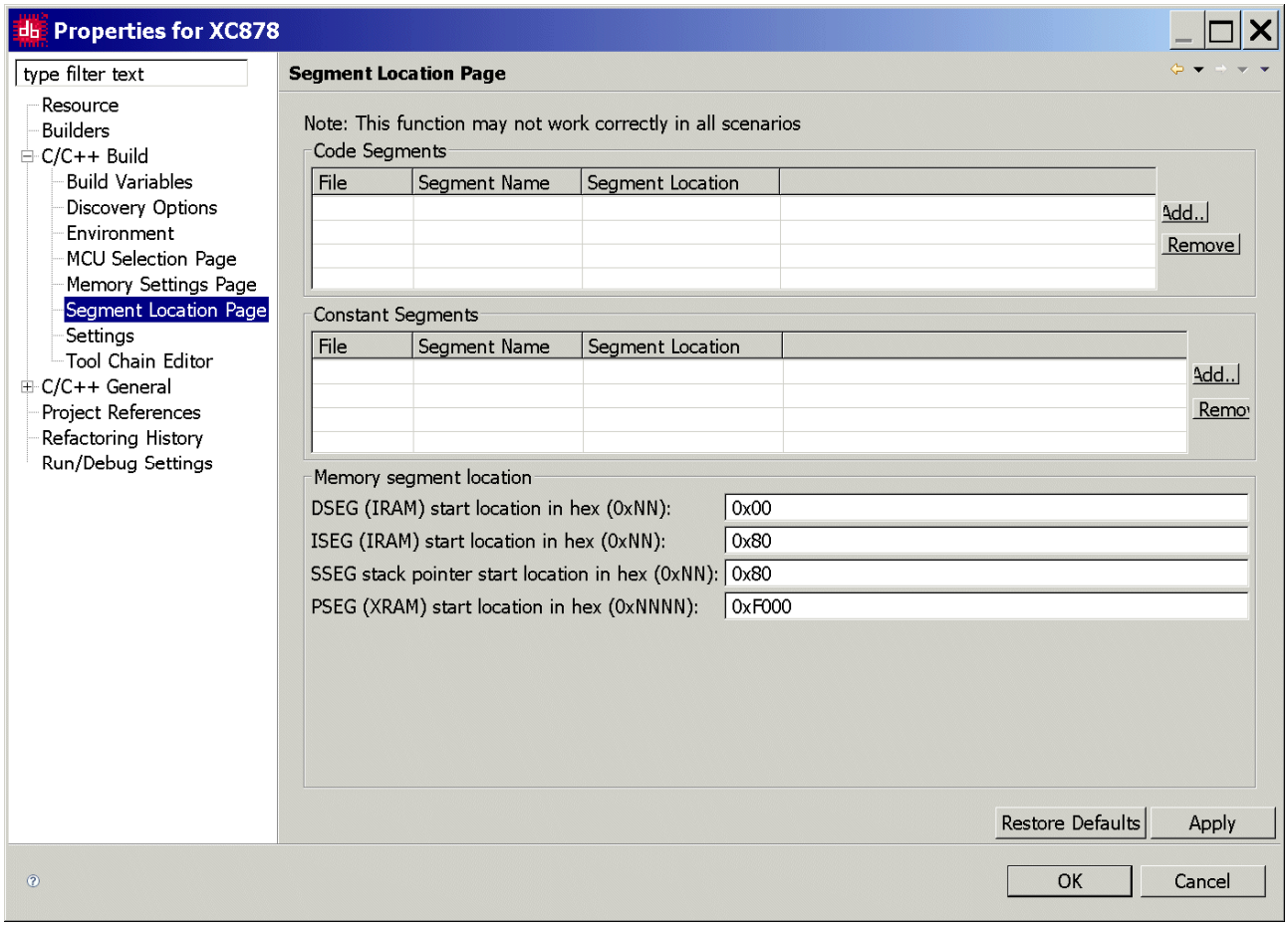
Additional information: Memory Map (Source: startupxc878.s):



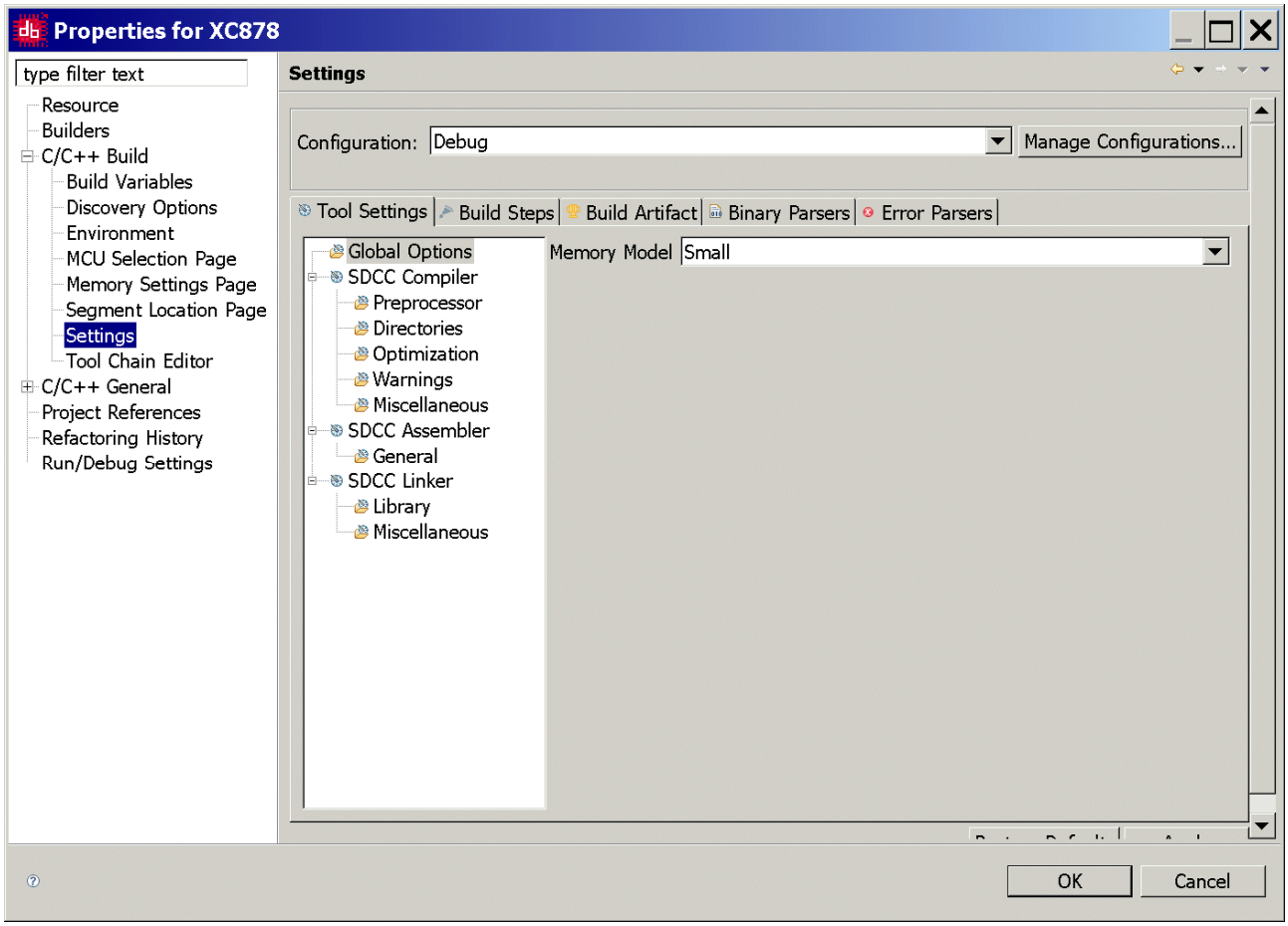
Project – Properties: C/C++ Build: Memory Setting Page:



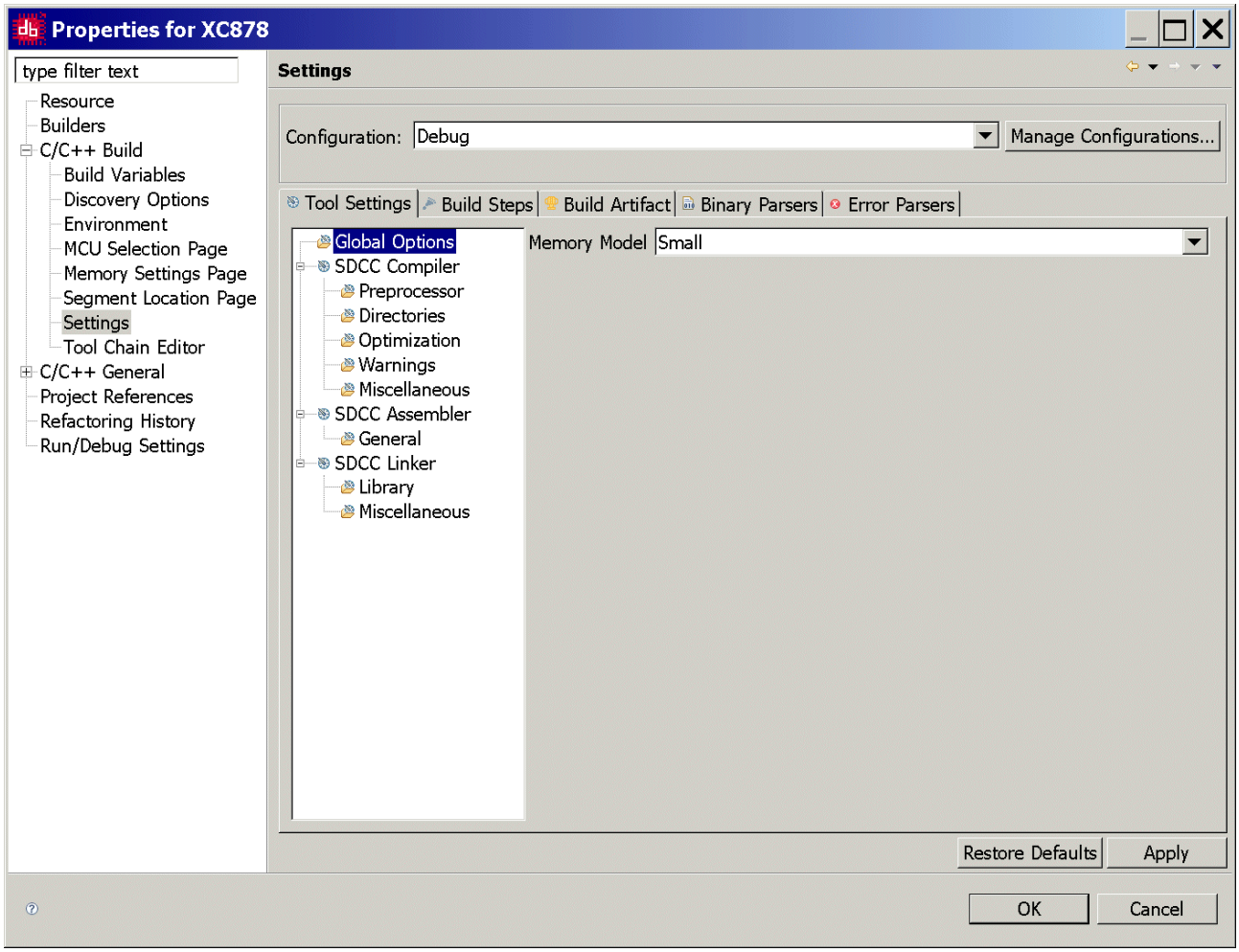
Project – Properties: C/C++ Build: Segment Location Page:



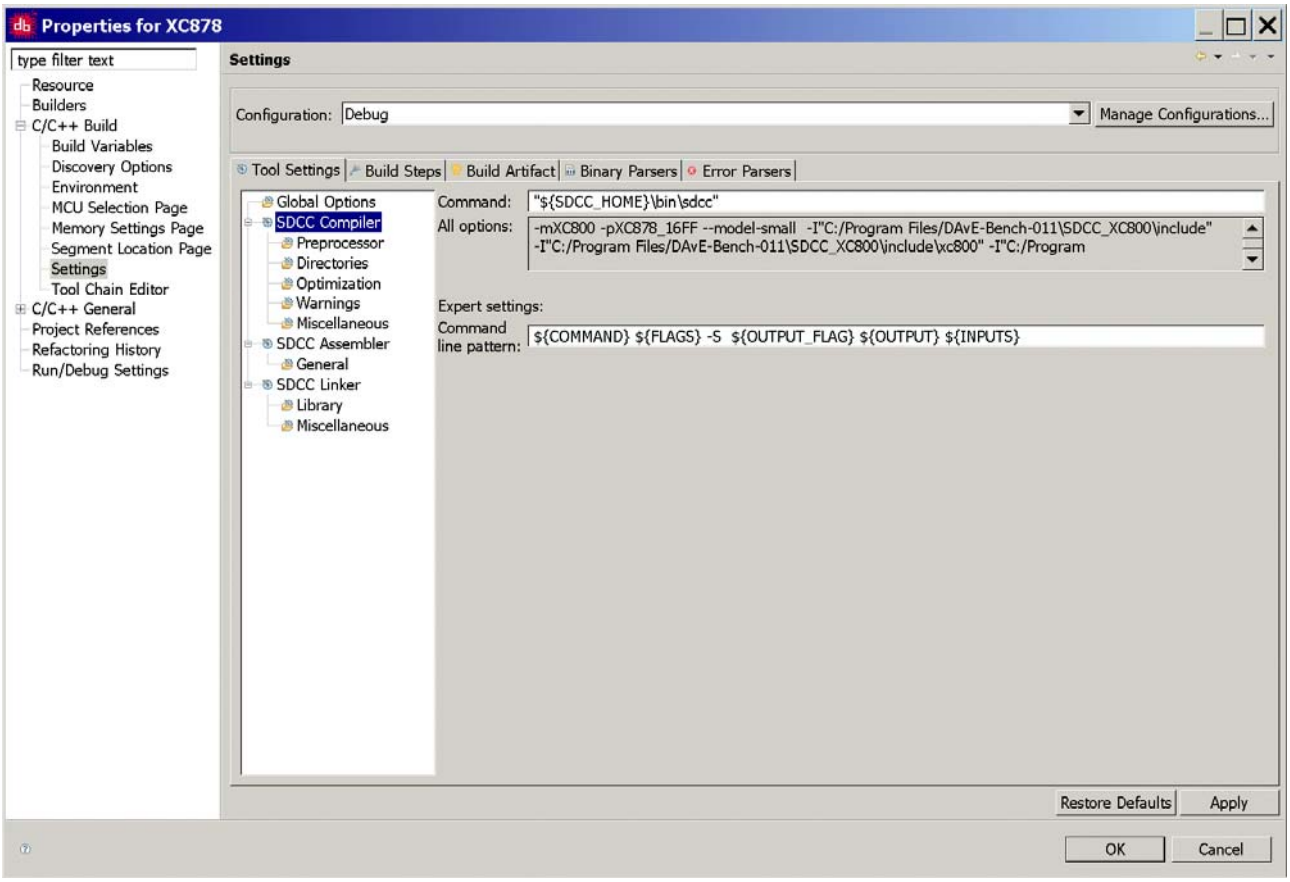
Project – Properties: C/C++ Build: Settings:



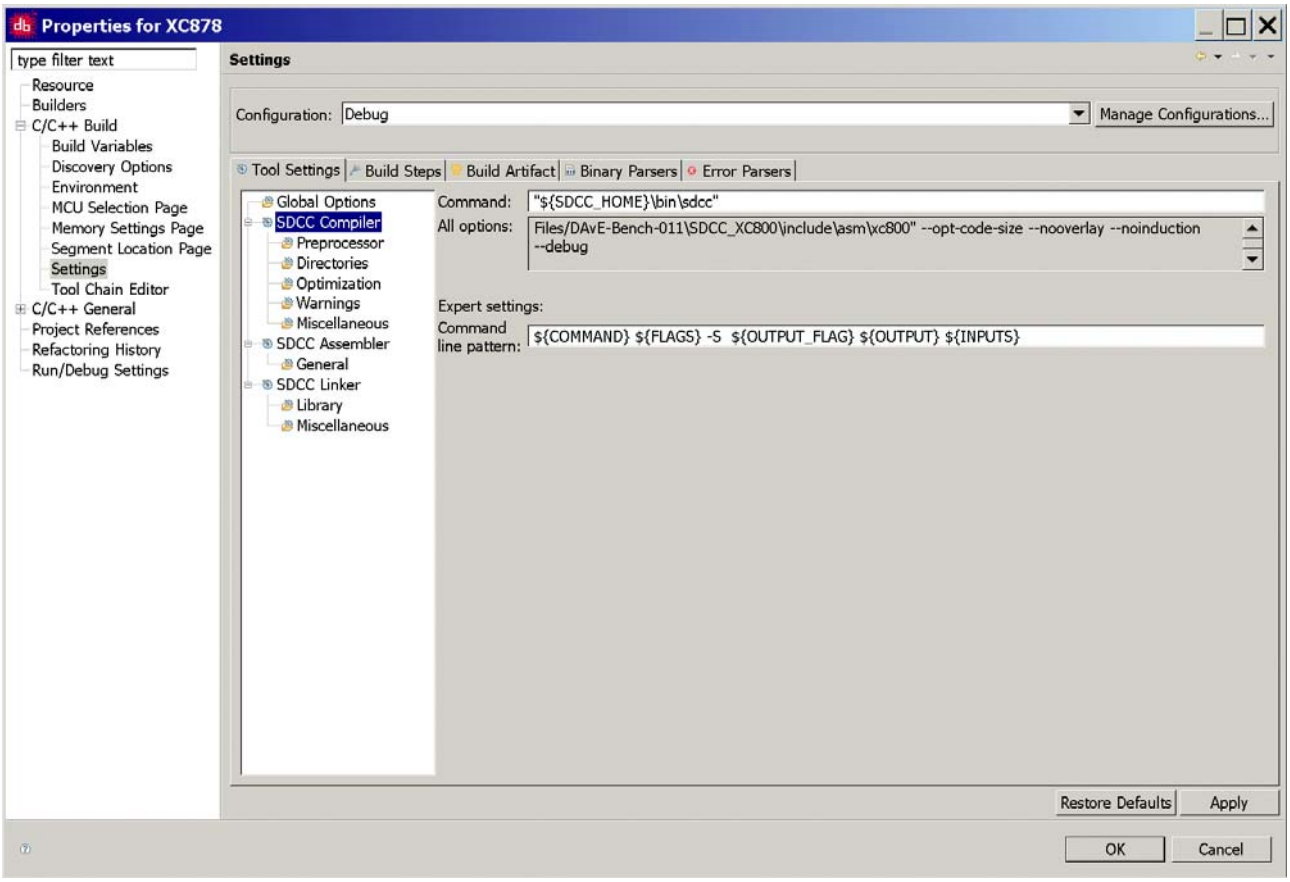
Project – Properties: C/C++ Build: Settings: Tool Settings: Global Options:



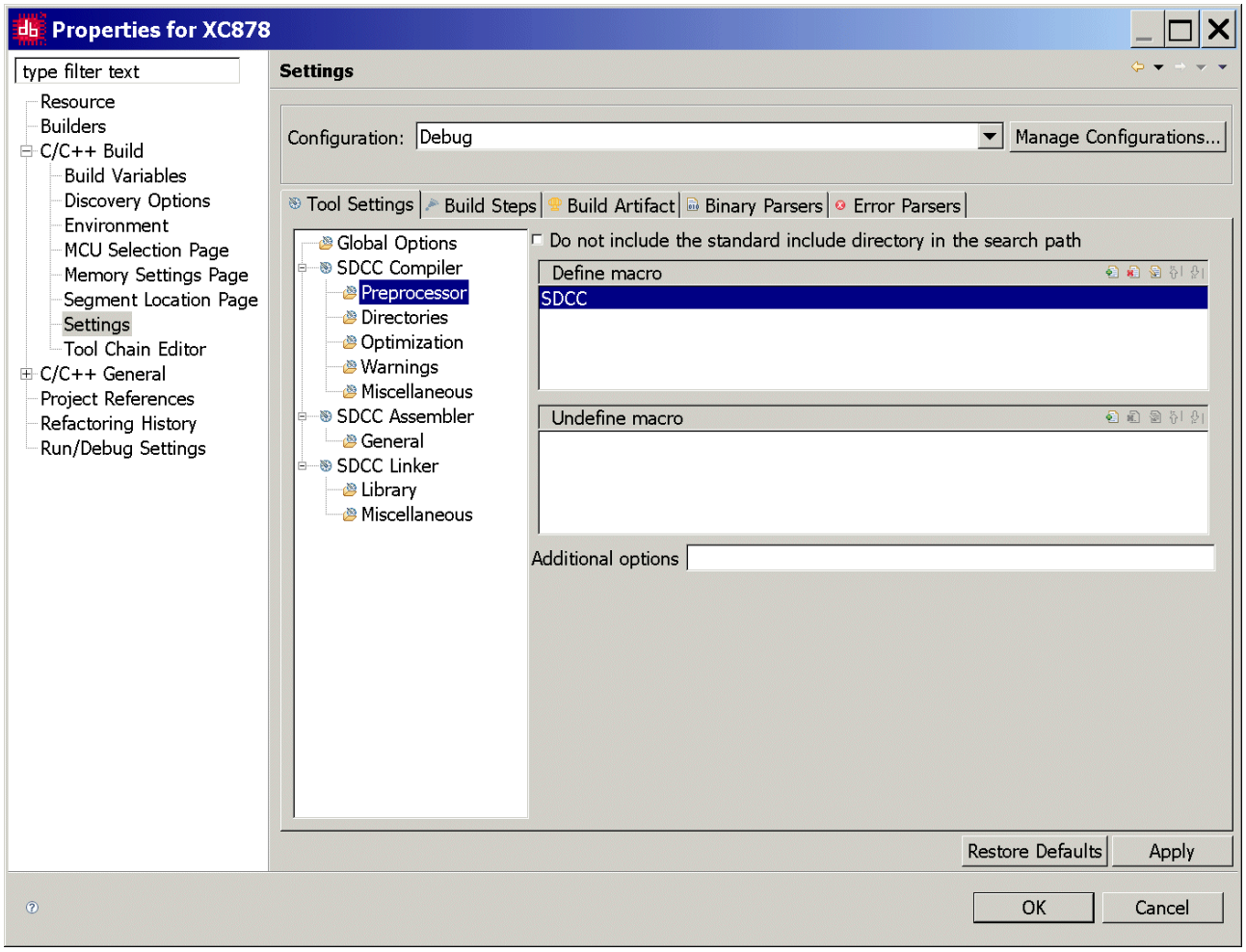
Project – Properties: C/C++ Build: Settings: Tool Settings: SDCC Compiler (1/2):



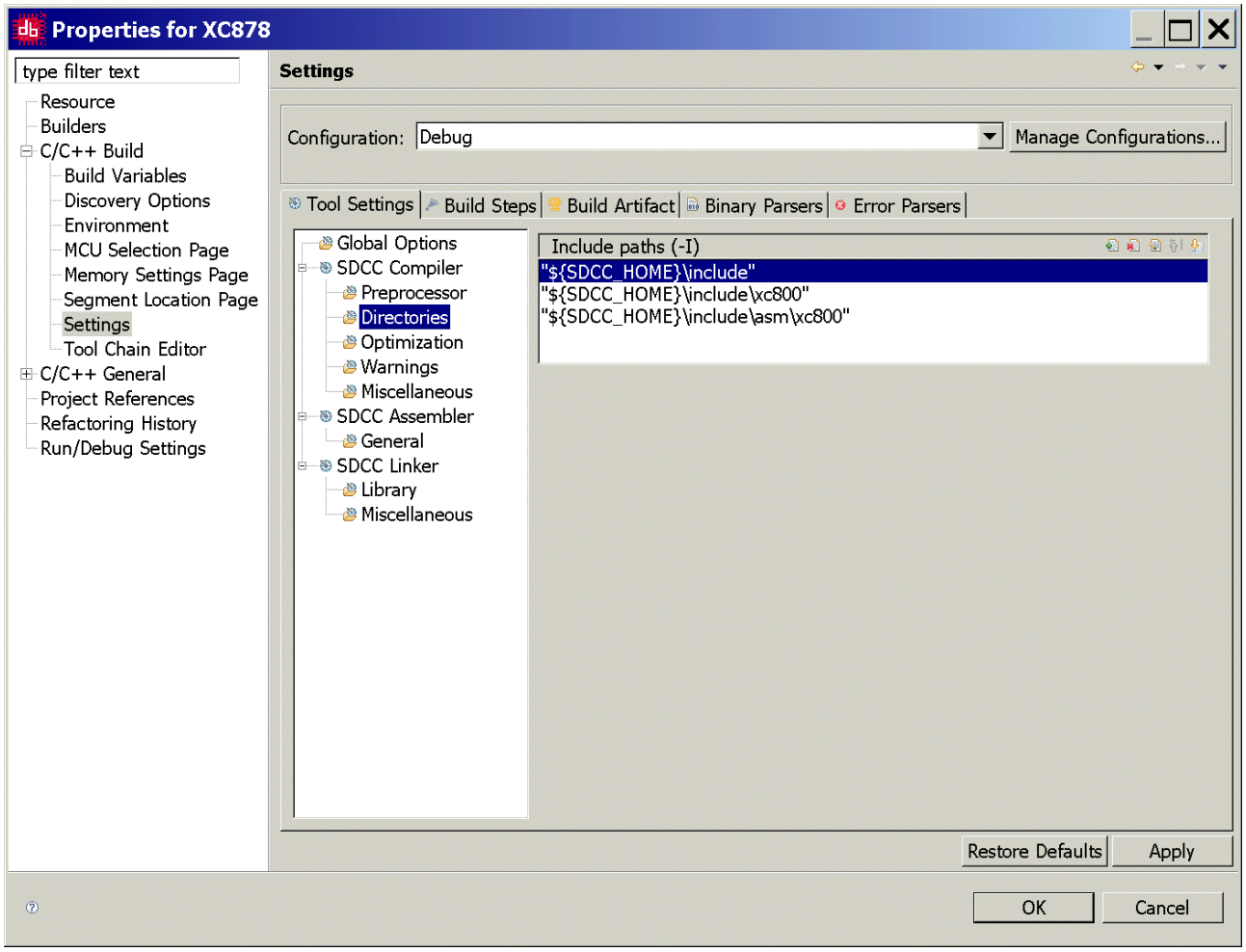
Project – Properties: C/C++ Build: Settings: Tool Settings: SDCC Compiler (2/2):



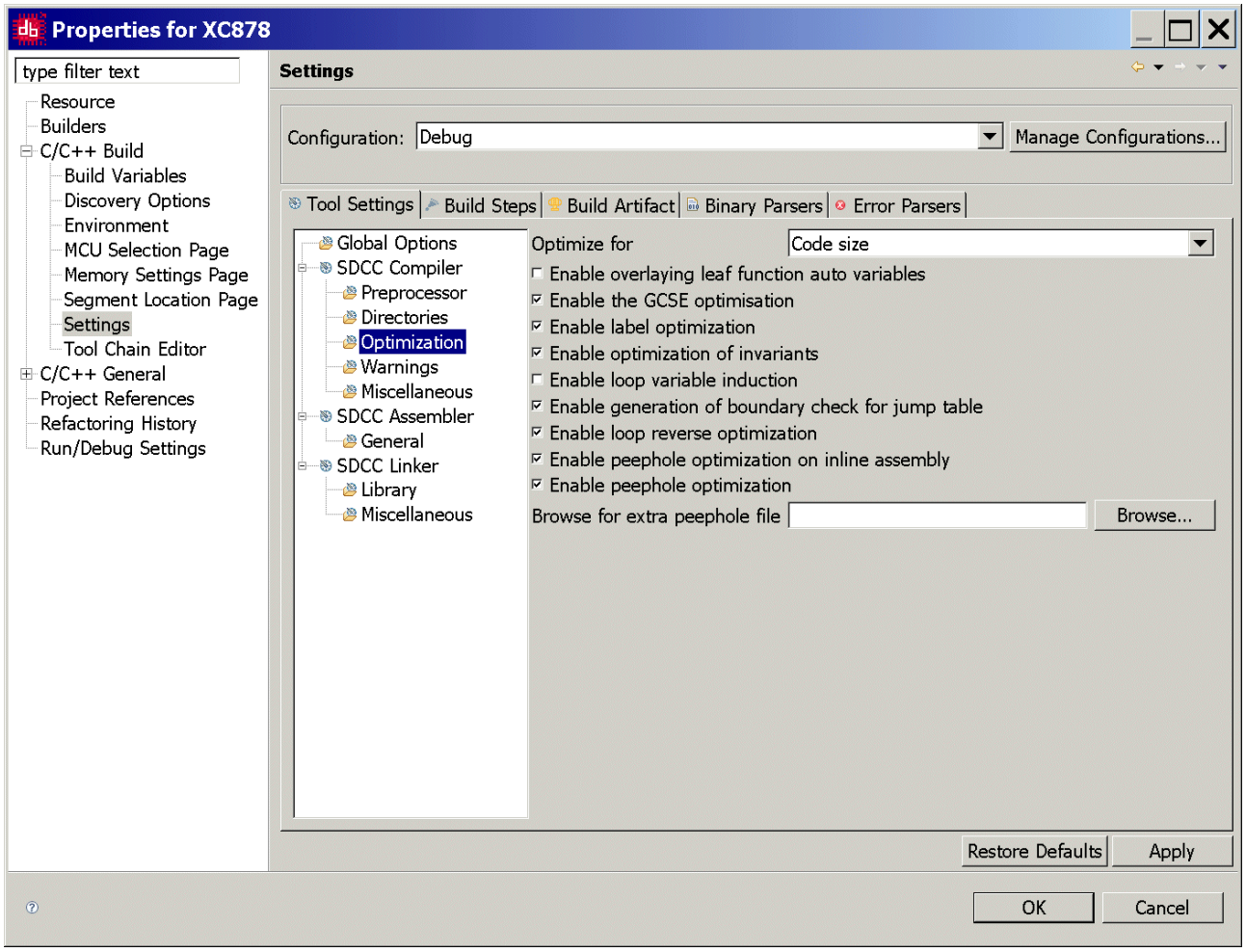
Project – Properties: C/C++ Build: Settings: Tool Settings: SDCC Compiler: Preprocessor:



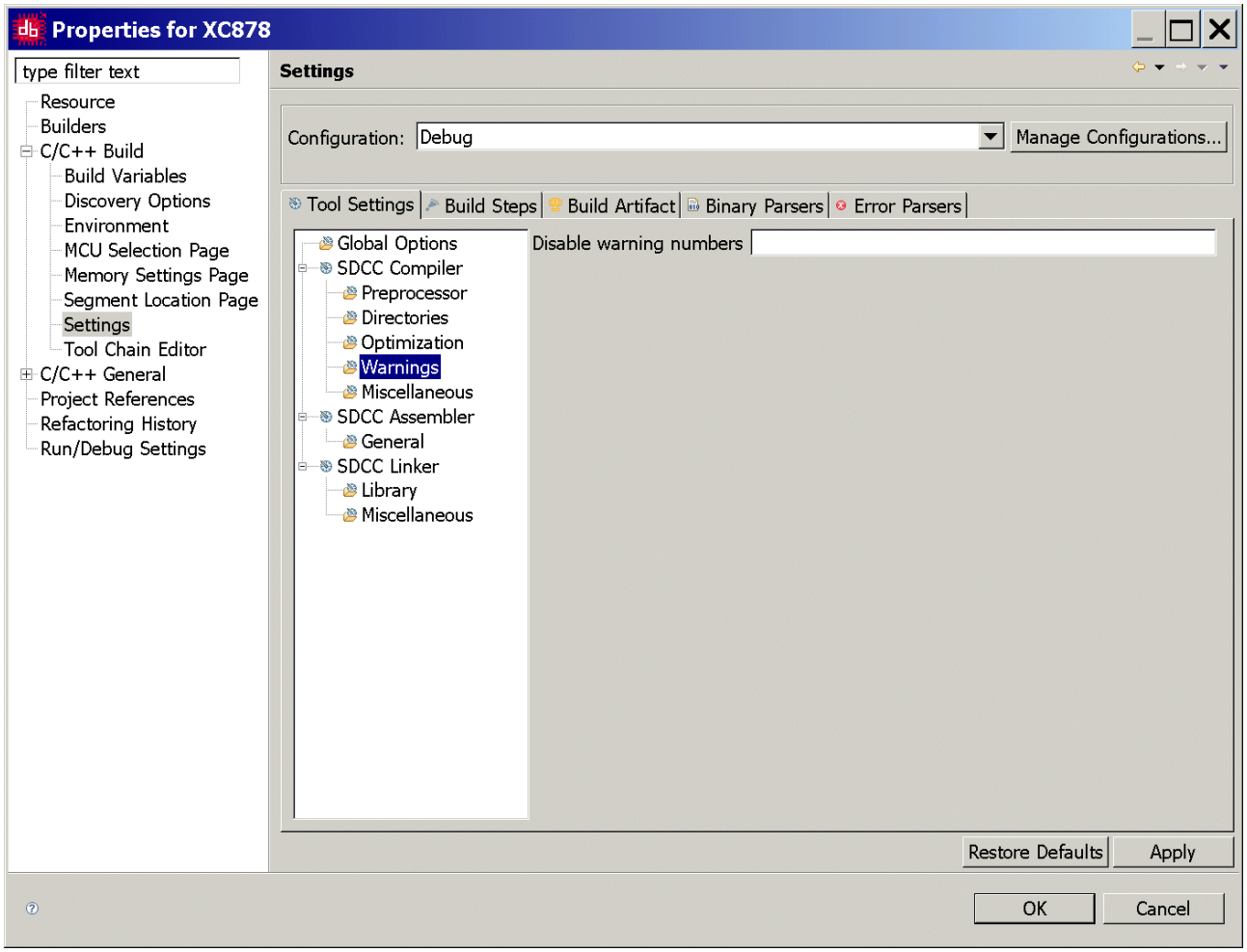
Project – Properties: C/C++ Build: Settings: Tool Settings: SDCC Compiler: Directories



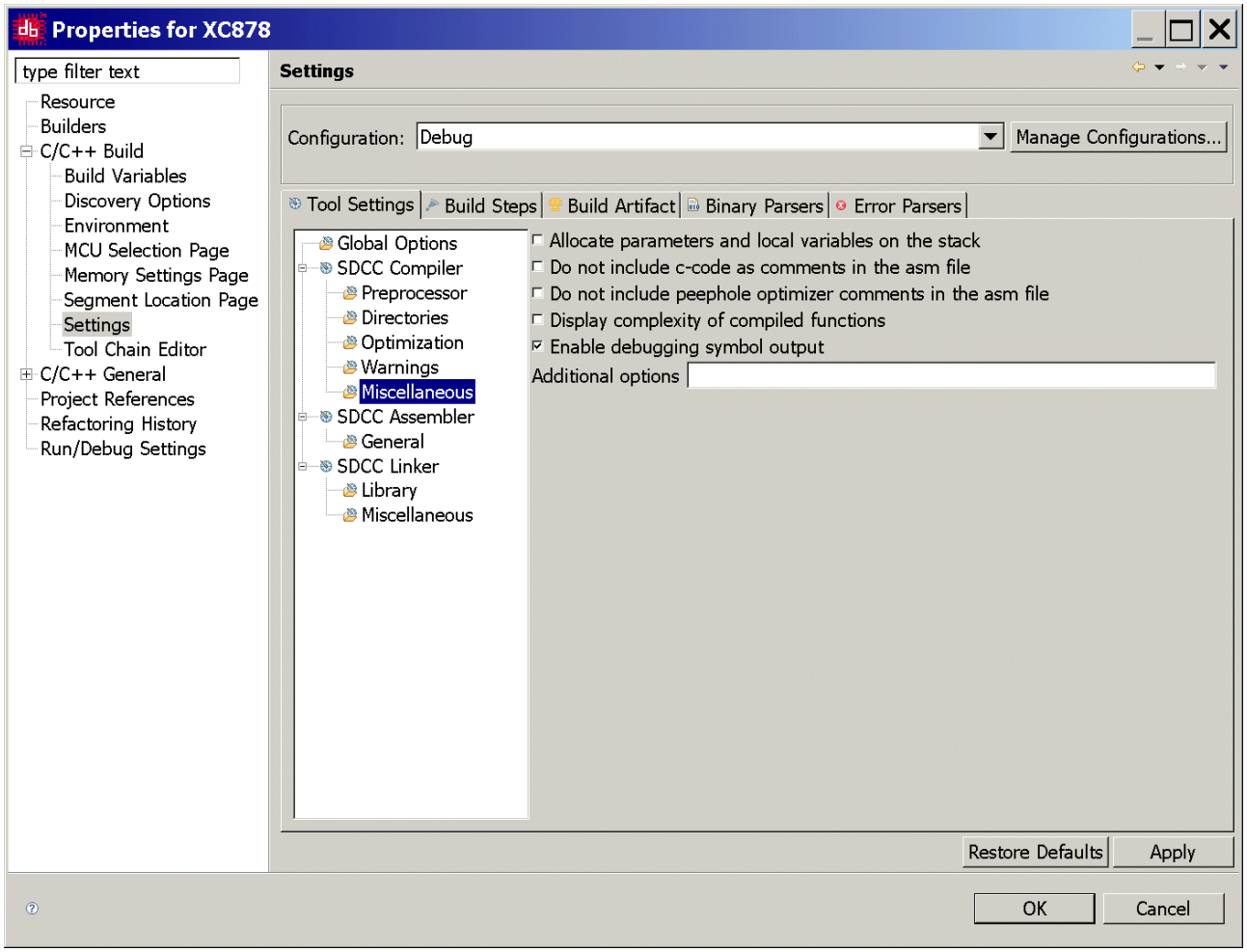
Project – Properties: C/C++ Build: Settings: Tool Settings: SDCC Compiler: Optimization:



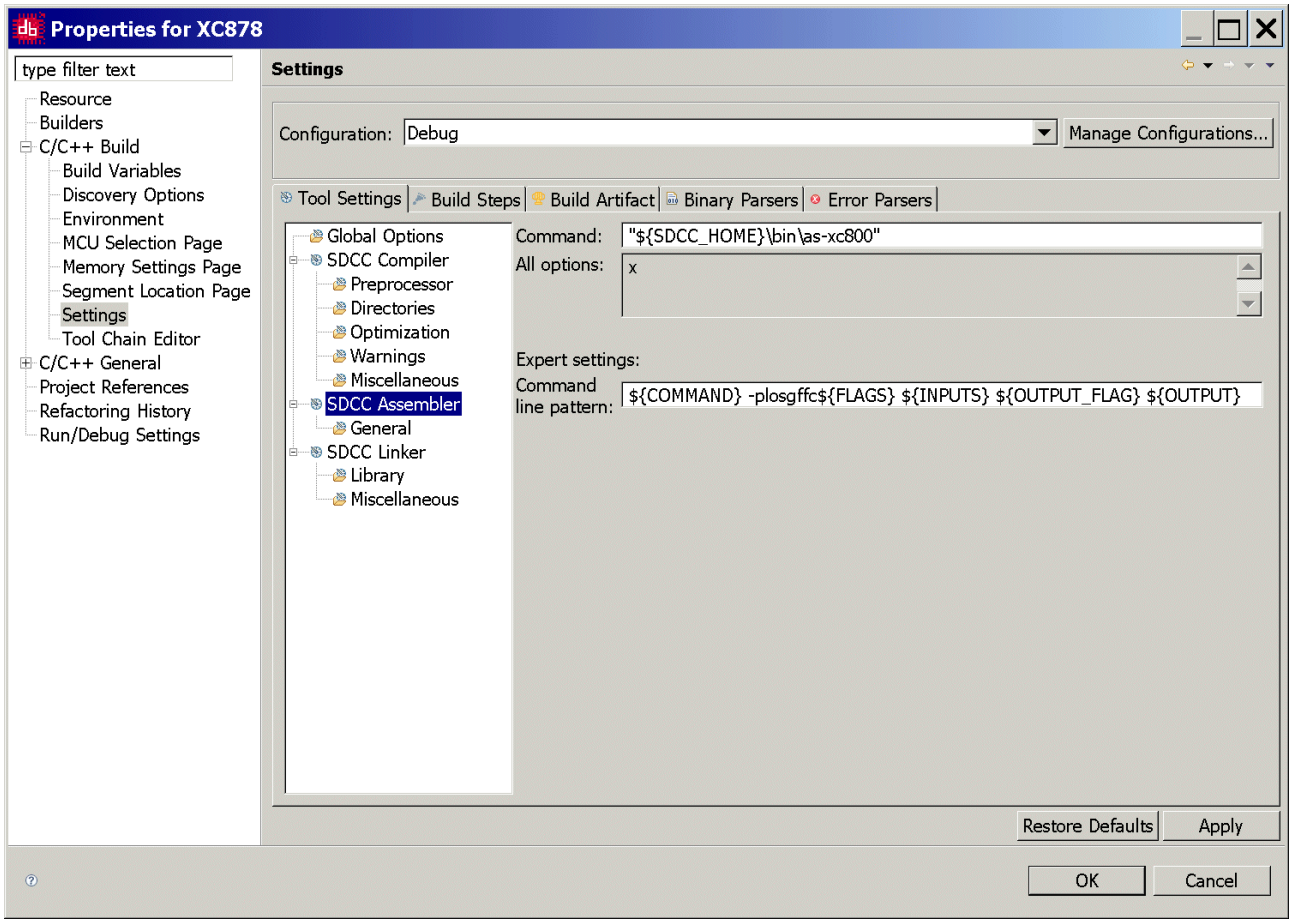
Project – Properties: C/C++ Build: Settings: Tool Settings: SDCC Compiler: Warnings:



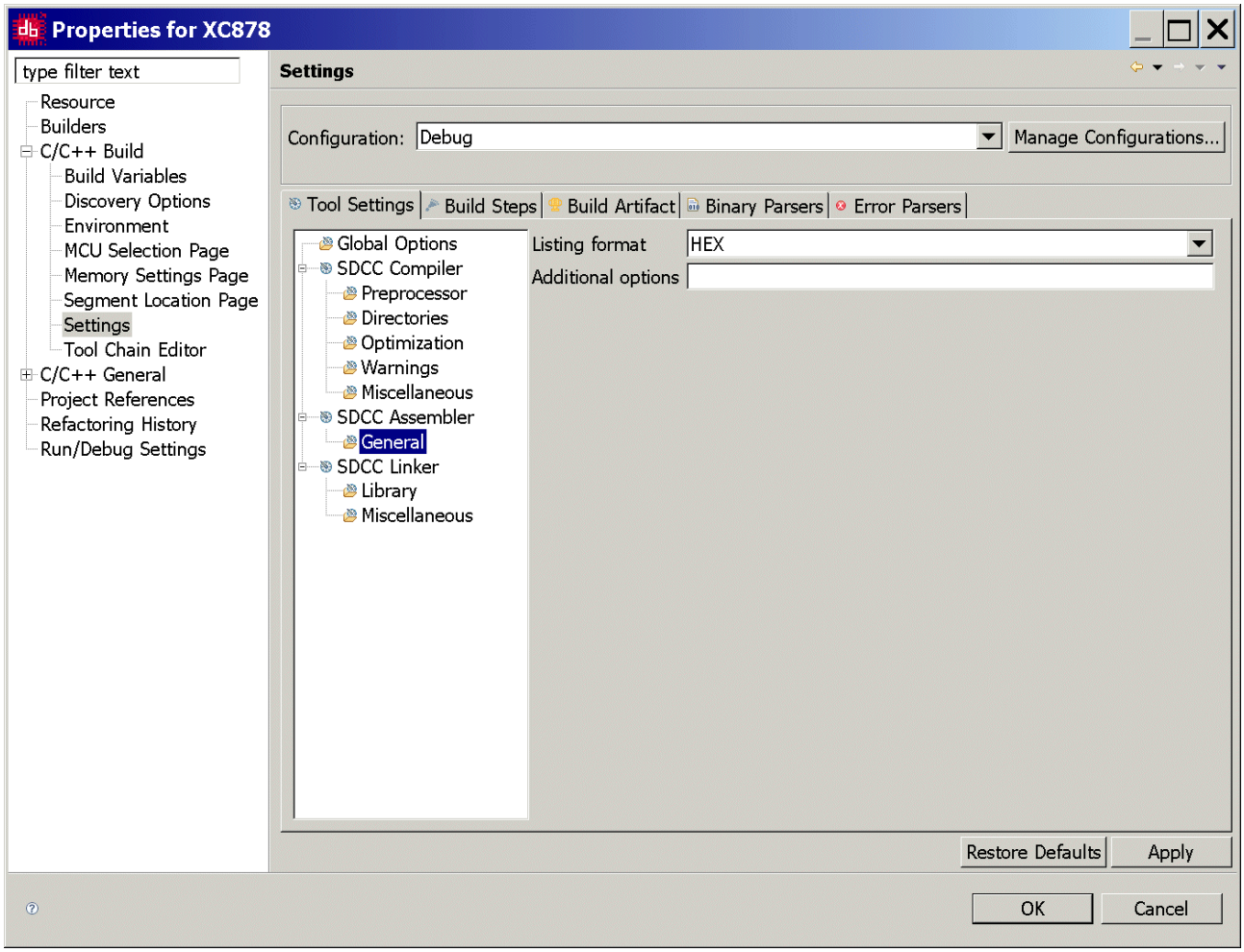
Project – Properties: C/C++ Build: Settings: Tool Settings: SDCC Compiler: Miscellaneous:



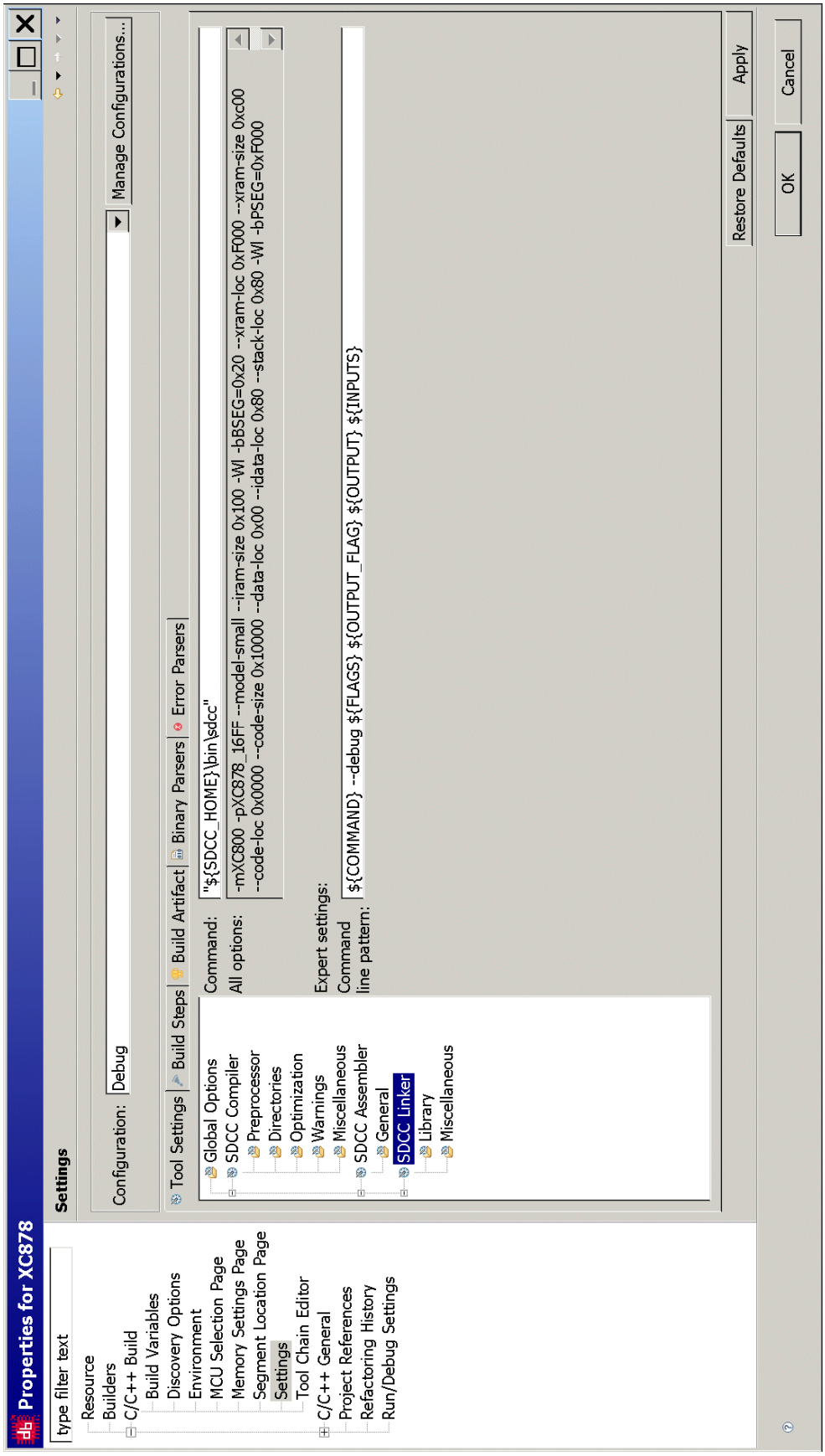
Project – Properties: C/C++ Build: Settings: Tool Settings: SDCC Assembler:



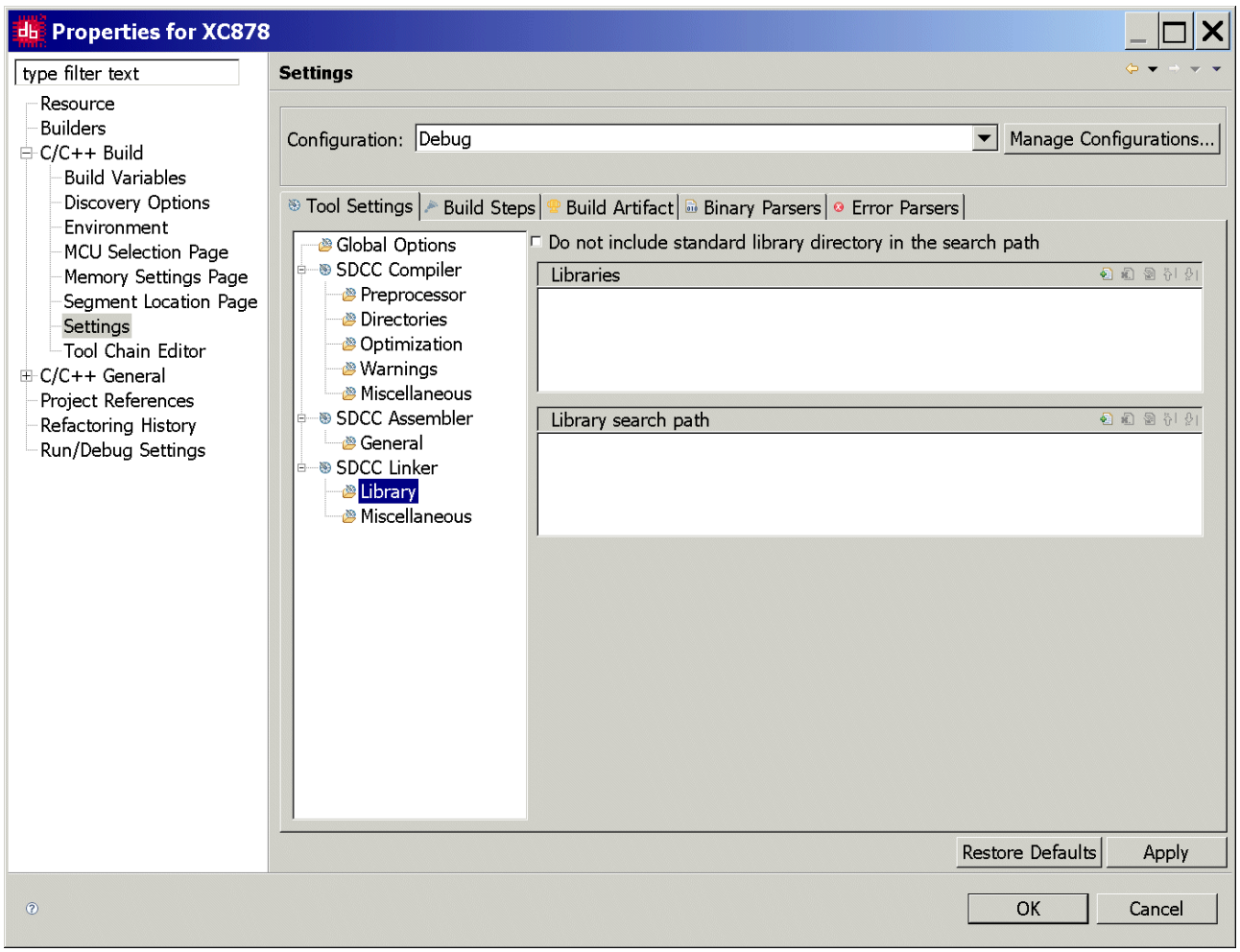
Project – Properties: C/C++ Build: Settings: Tool Settings: SDCC Assembler: General:



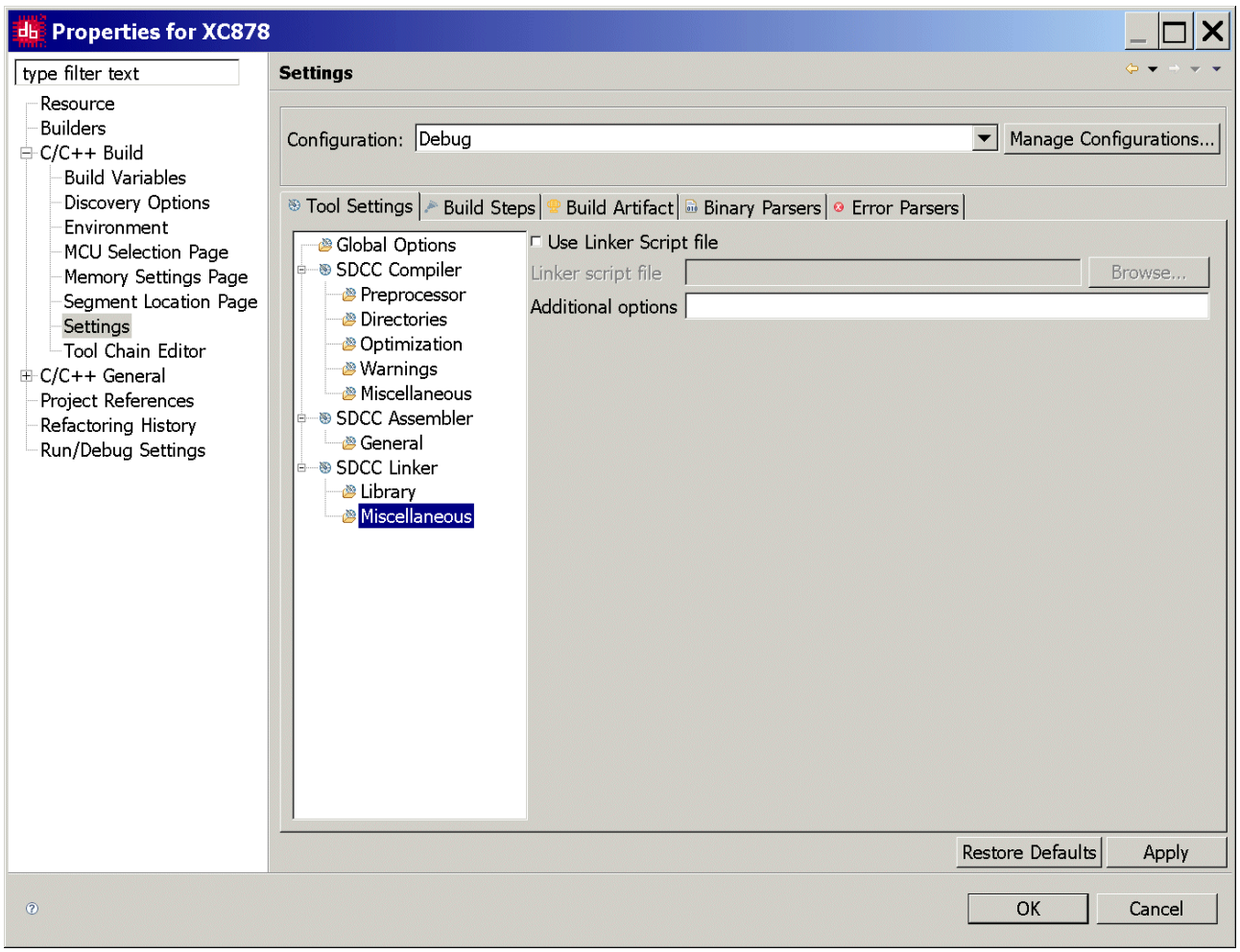
Project – Properties: C/C++ Build: Settings: Tool Settings: SDCC Linker:



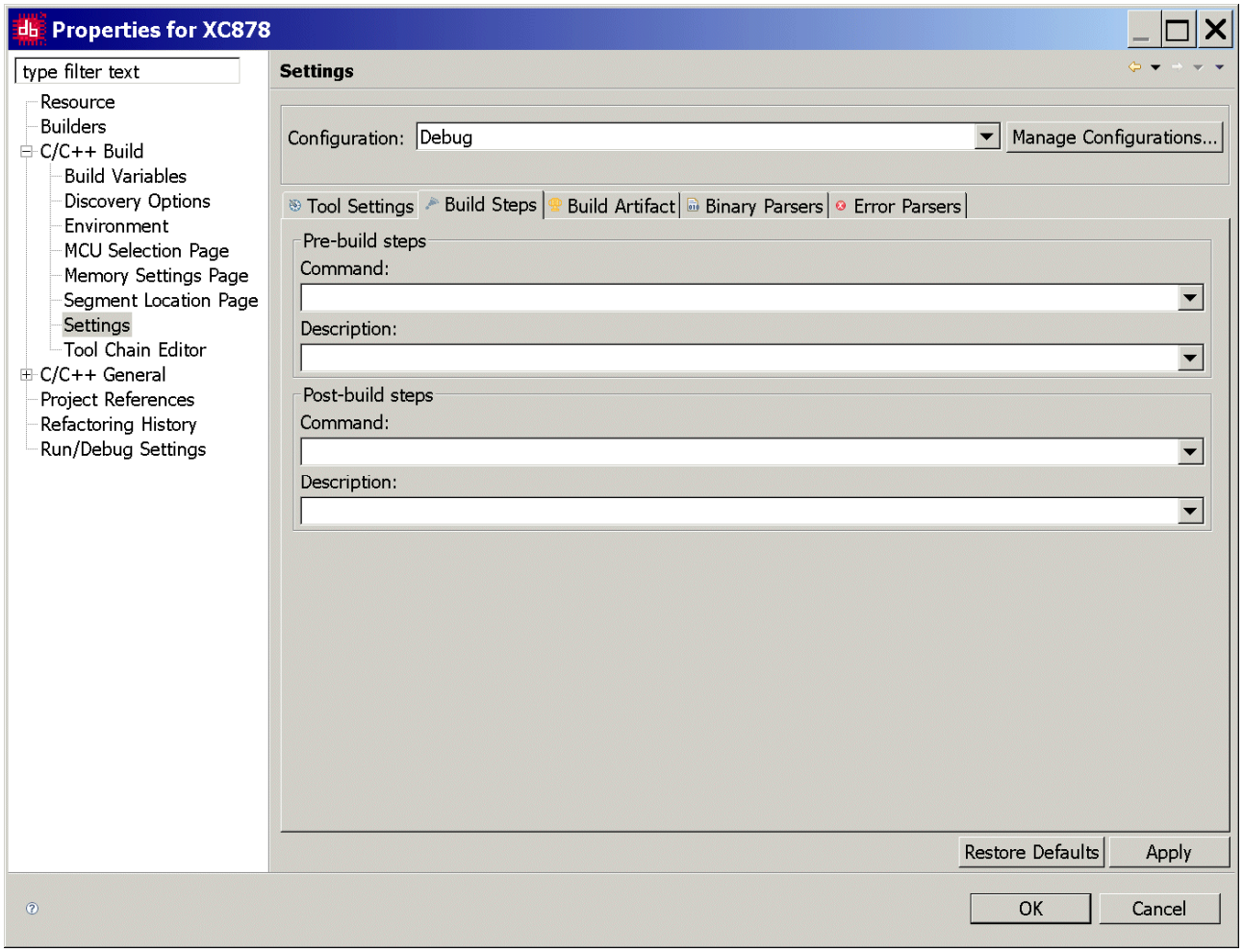
Project – Properties: C/C++ Build: Settings: Tool Settings: SDCC Linker: Library:



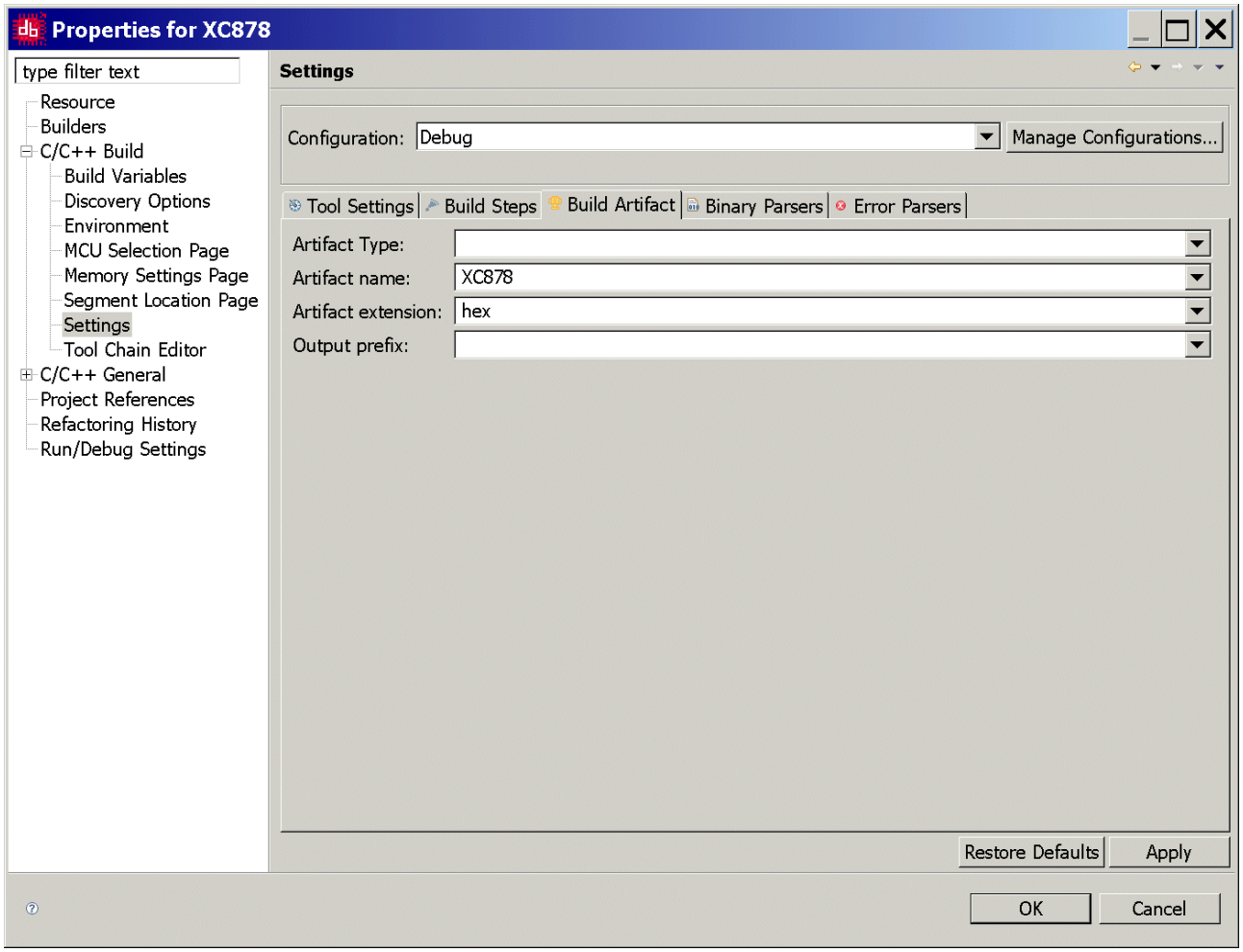
Project – Properties: C/C++ Build: Settings: Tool Settings: SDCC Linker: Miscellaneous:



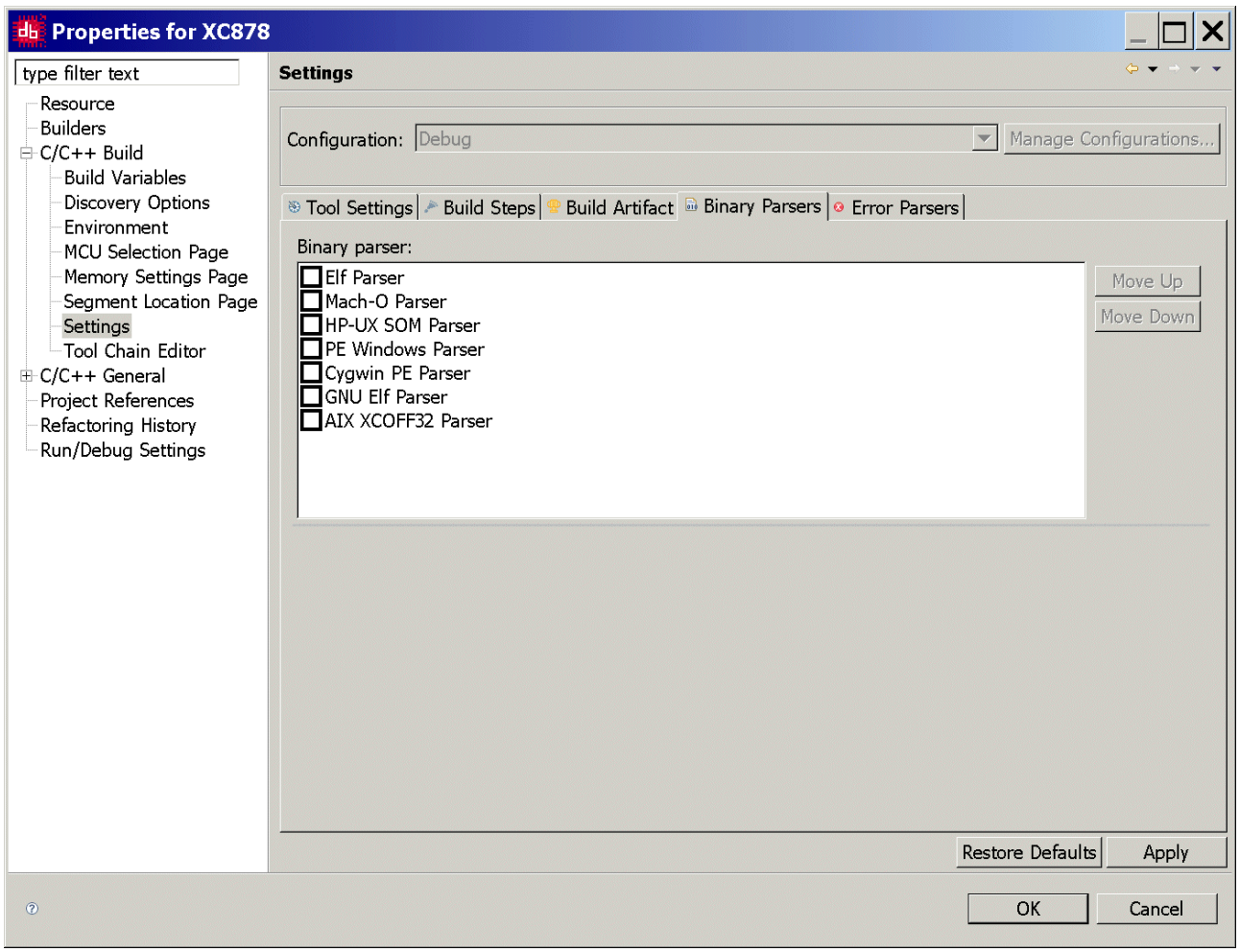
Project – Properties: C/C++ Build: Settings: Build Steps:



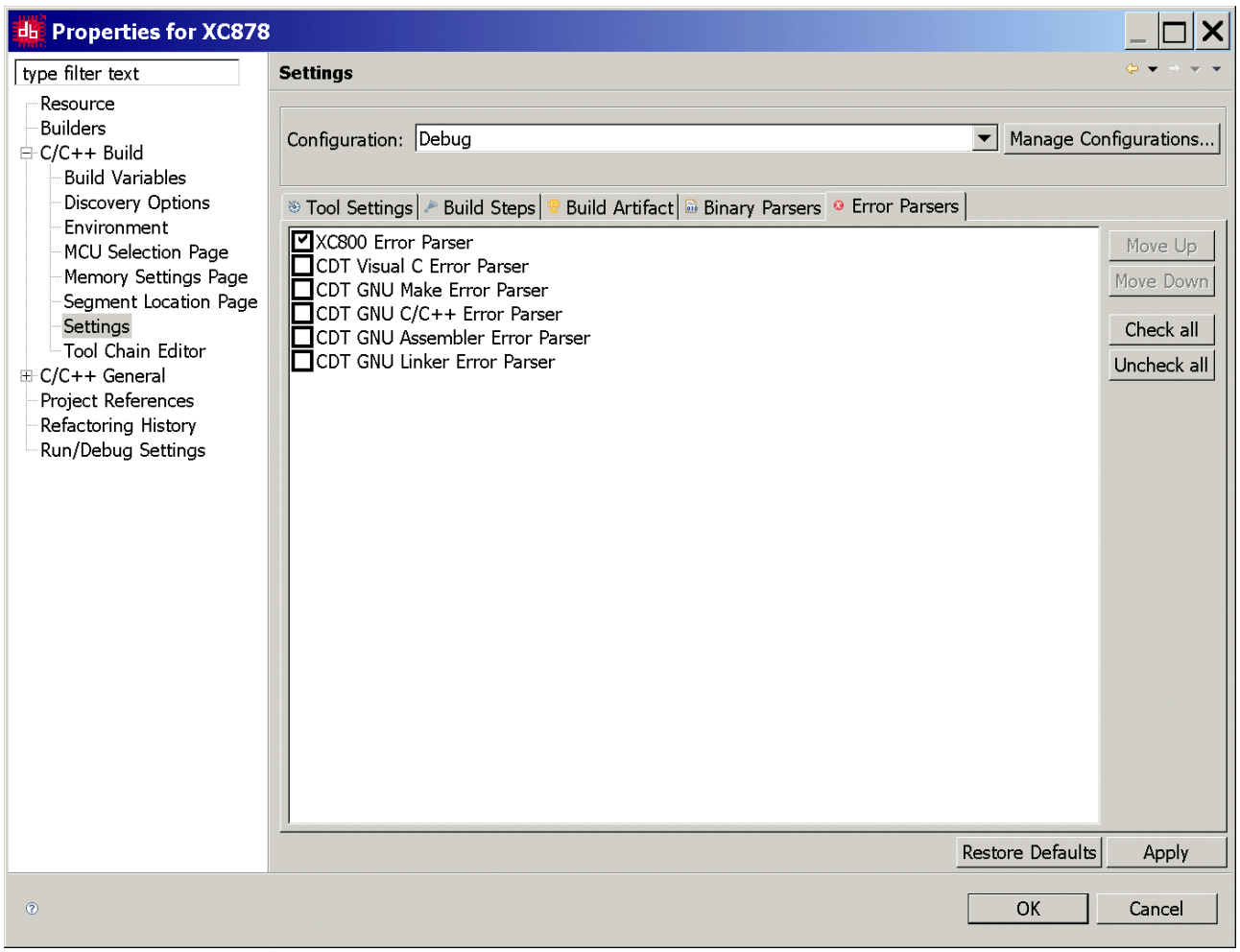
Project – Properties: C/C++ Build: Settings: Build Artifact:



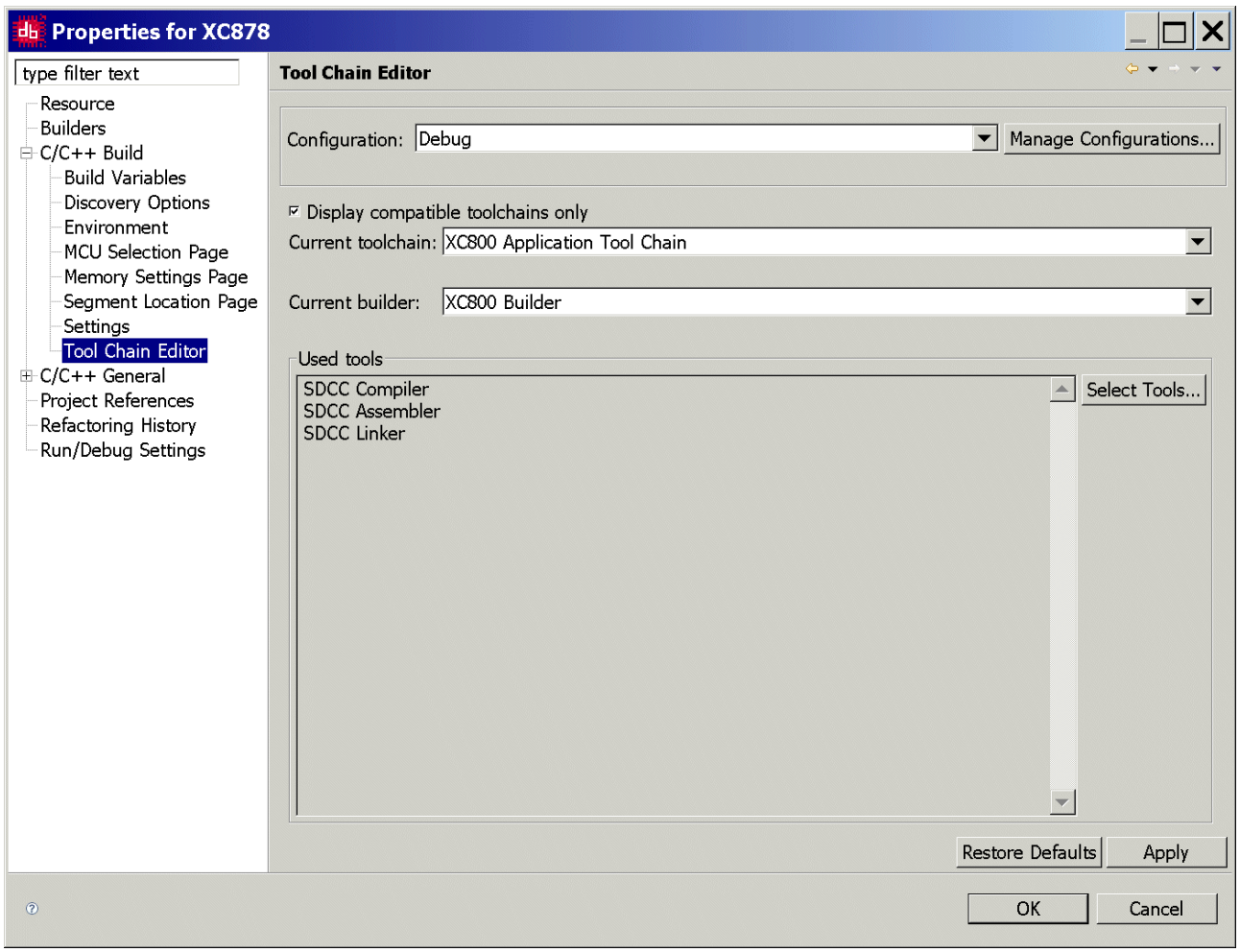
Project – Properties: C/C++ Build: Settings: Binary Parsers:



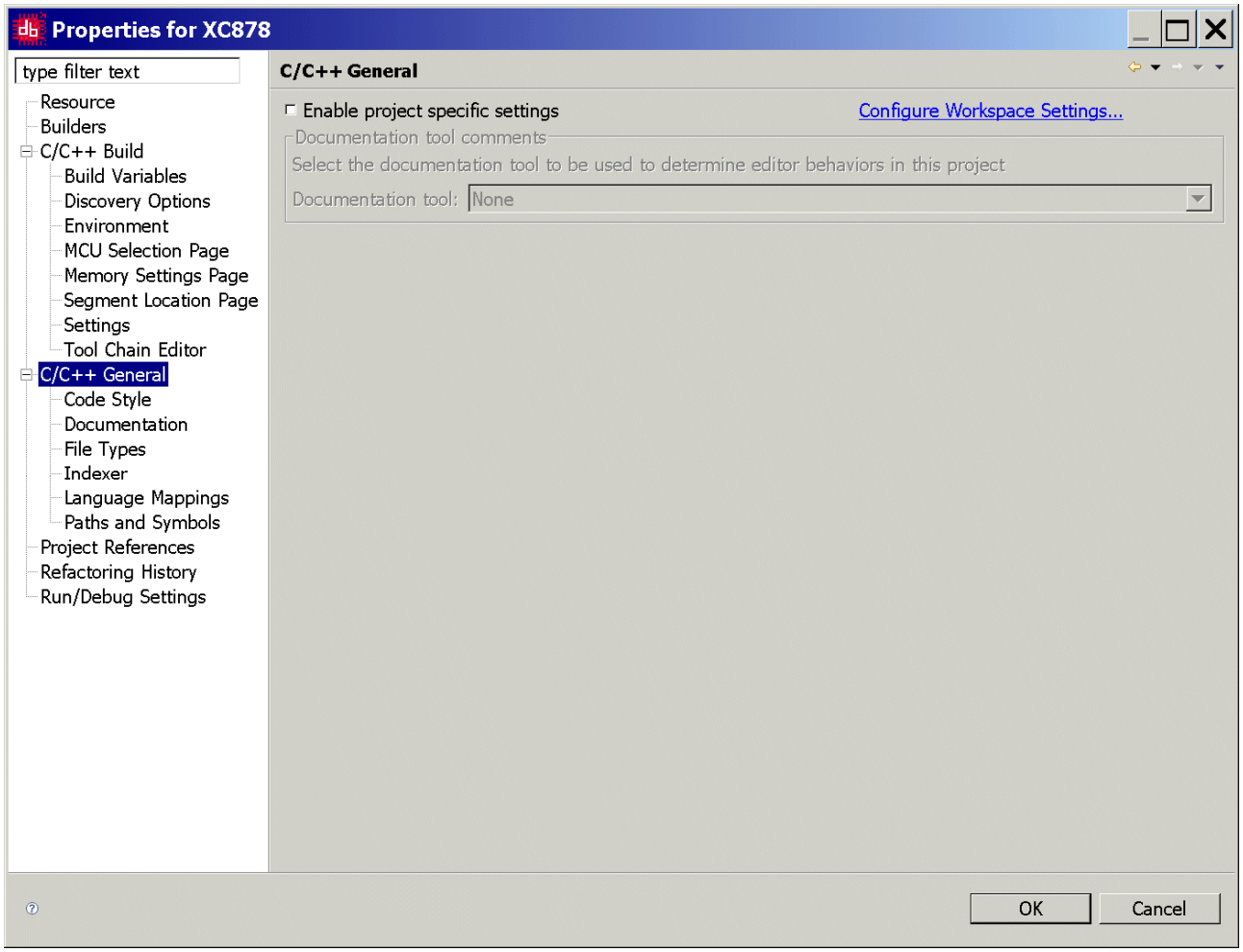
Project – Properties: C/C++ Build: Settings: Error Parsers:



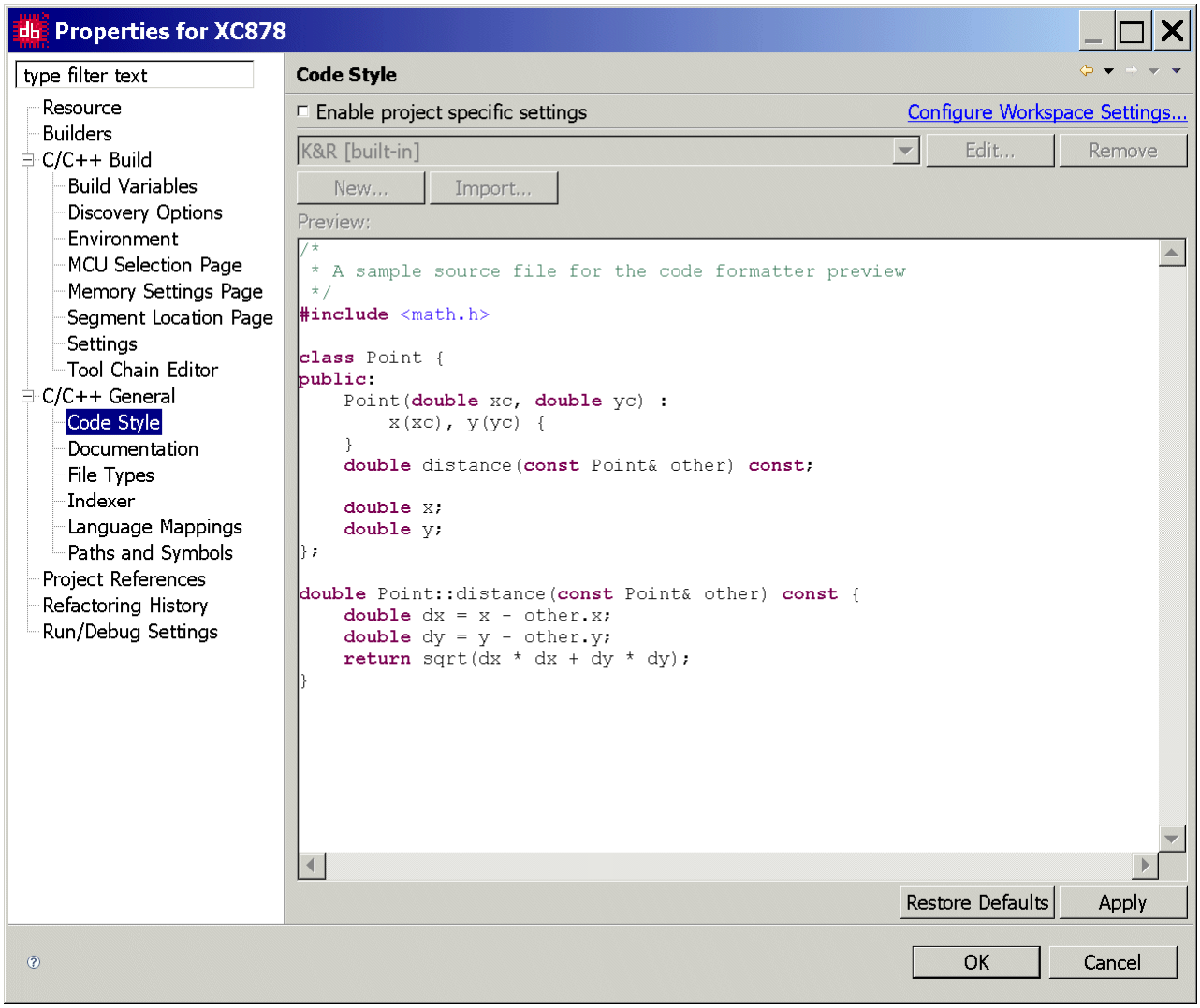
Project – Properties: C/C++ Build: Tool Chain Editor:



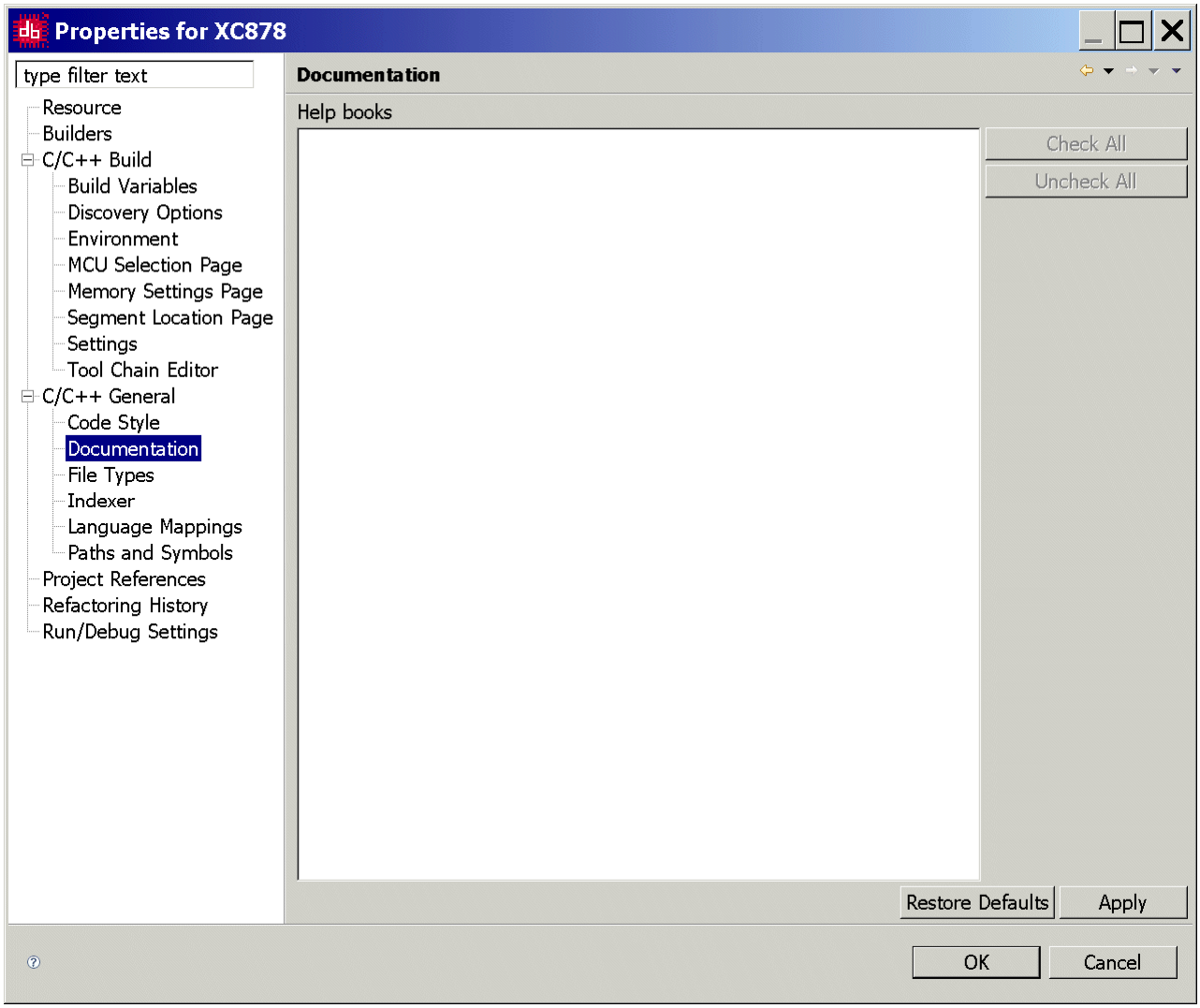
Project – Properties: C/C++ General:



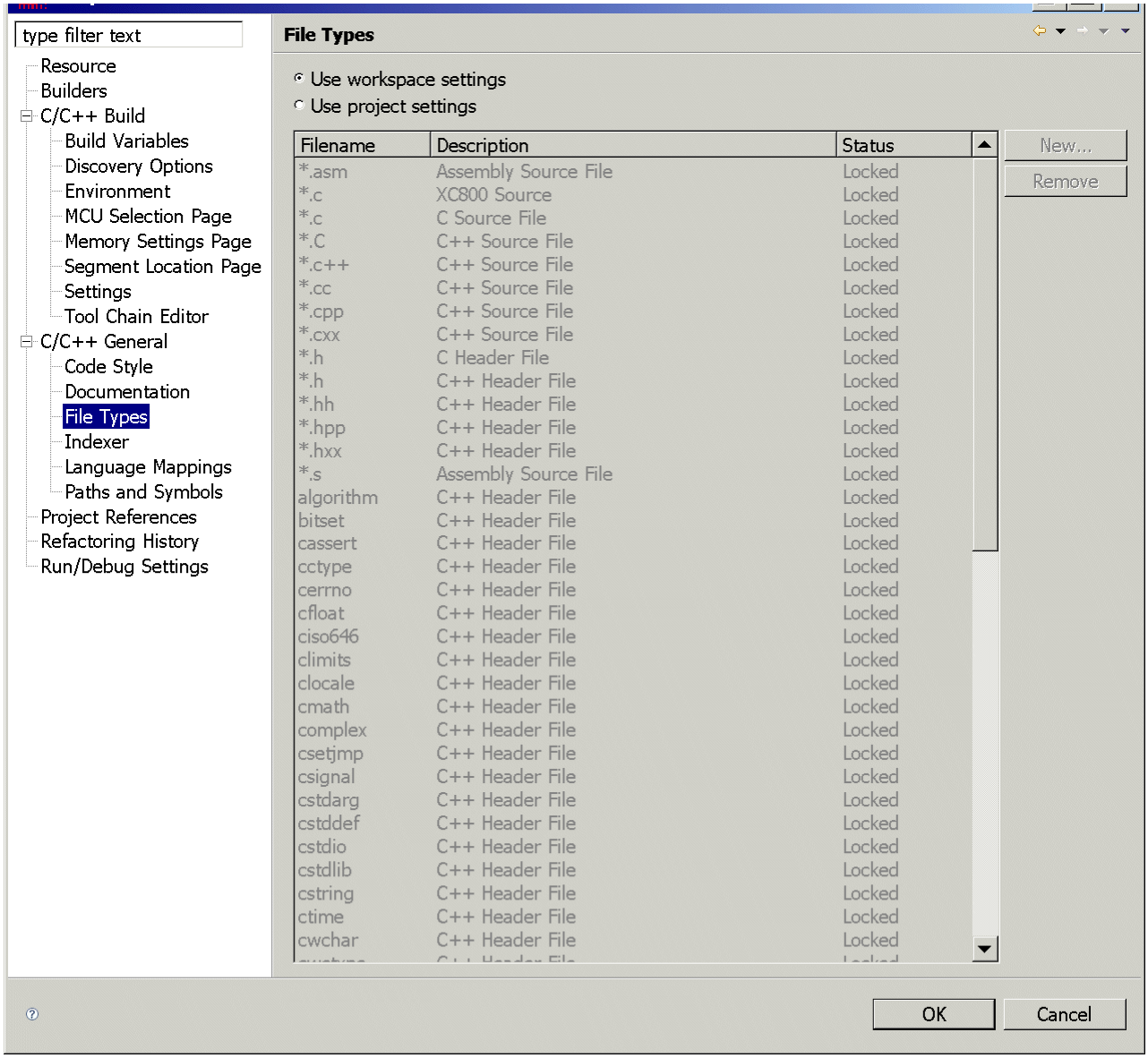
Project – Properties: C/C++ General: Code Style:



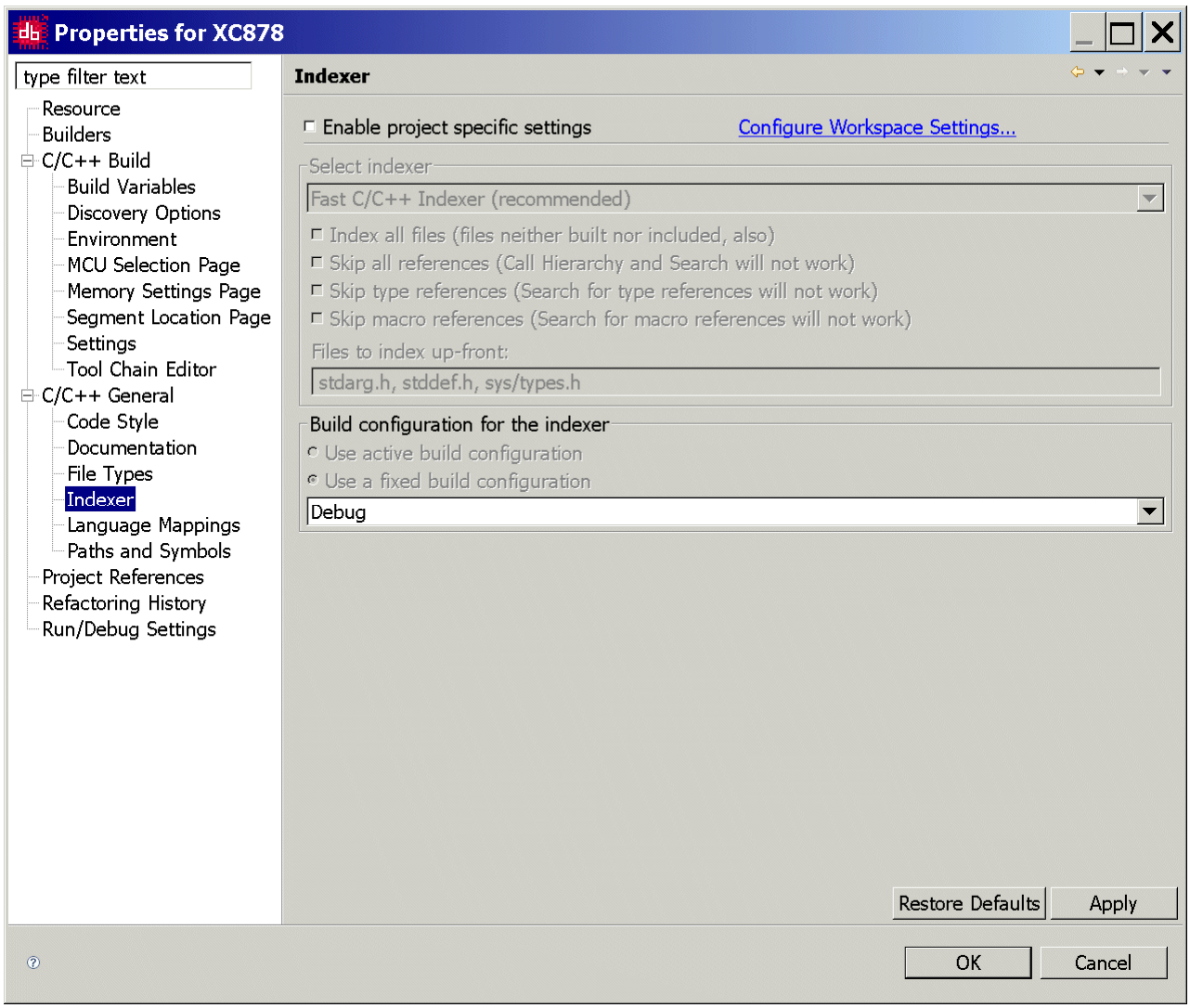
Project – Properties: C/C++ General: Documentation:



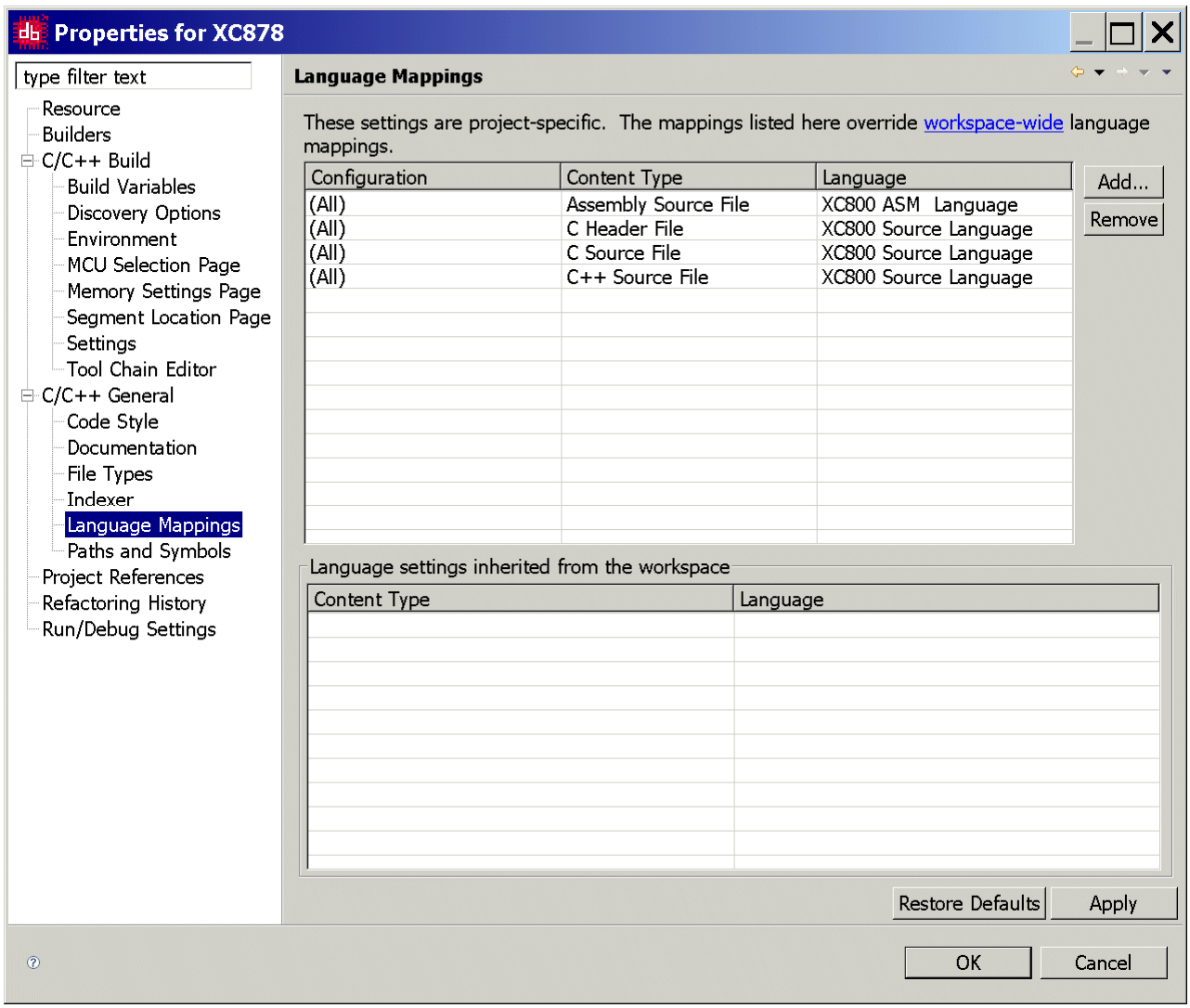
Project – Properties: C/C++ General: File Types:



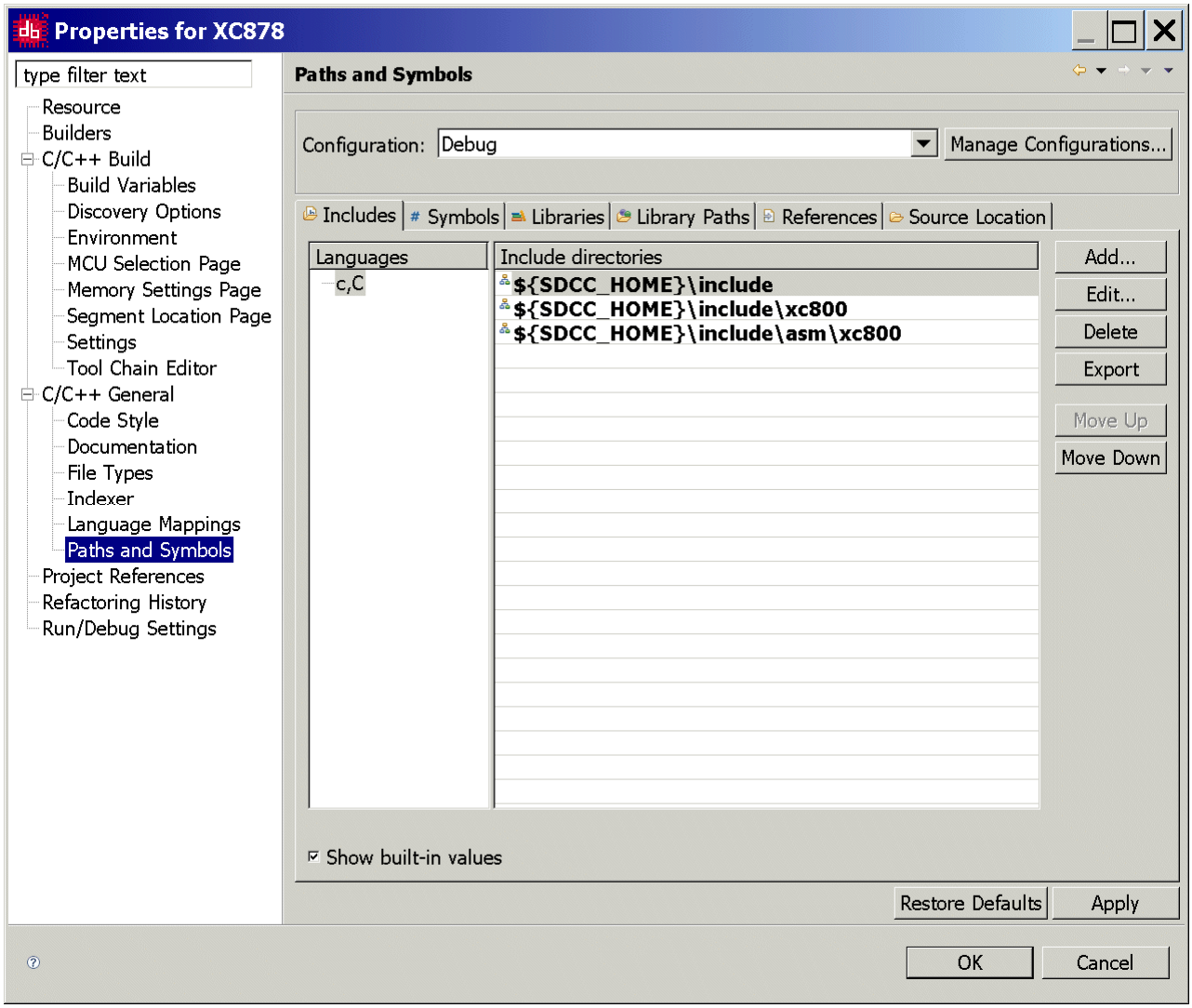
Project – Properties: C/C++ General: Indexer:



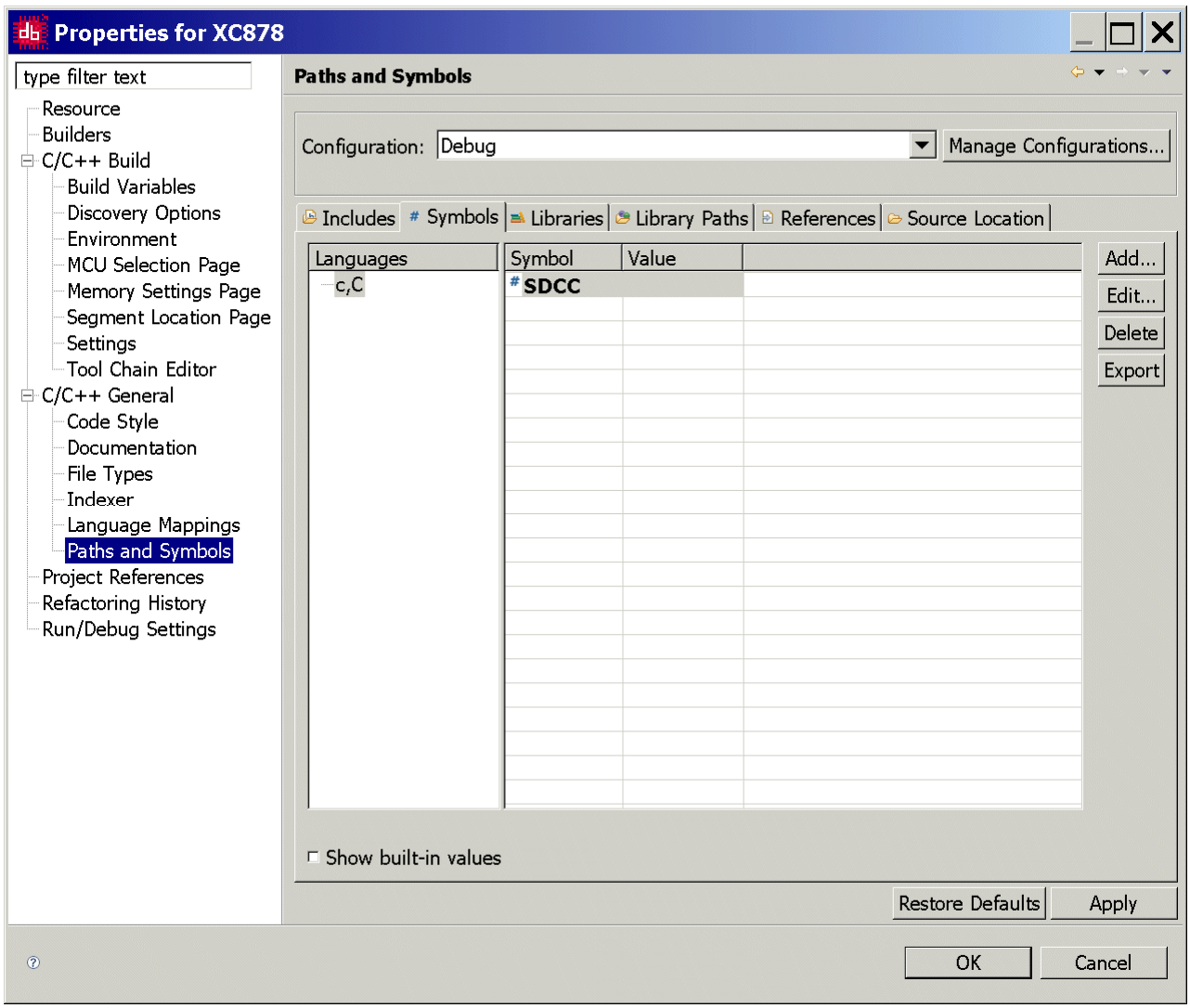
Project – Properties: C/C++ General: Language Mappings:



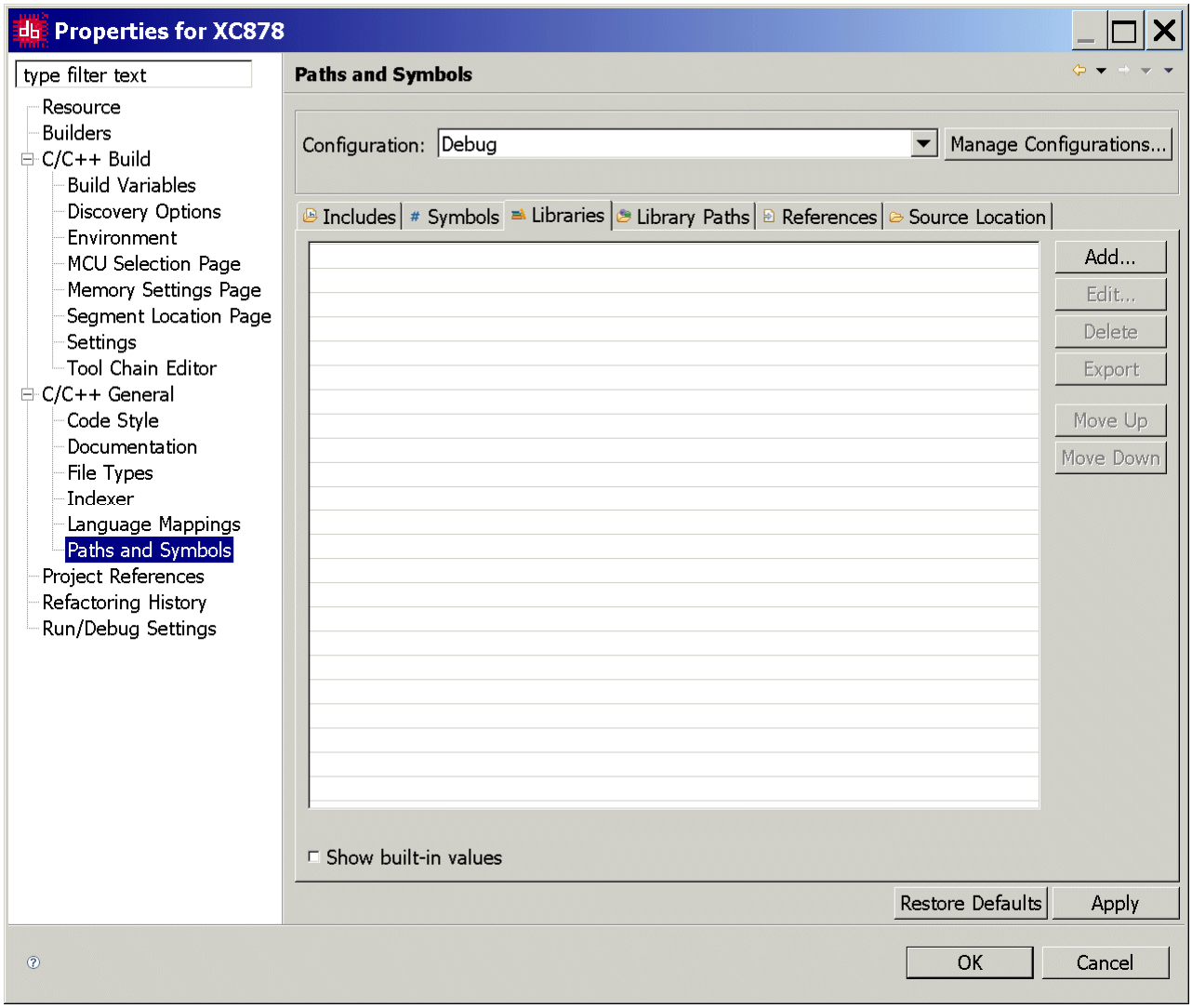
Project – Properties: C/C++ General: Paths and Symbols: Includes:



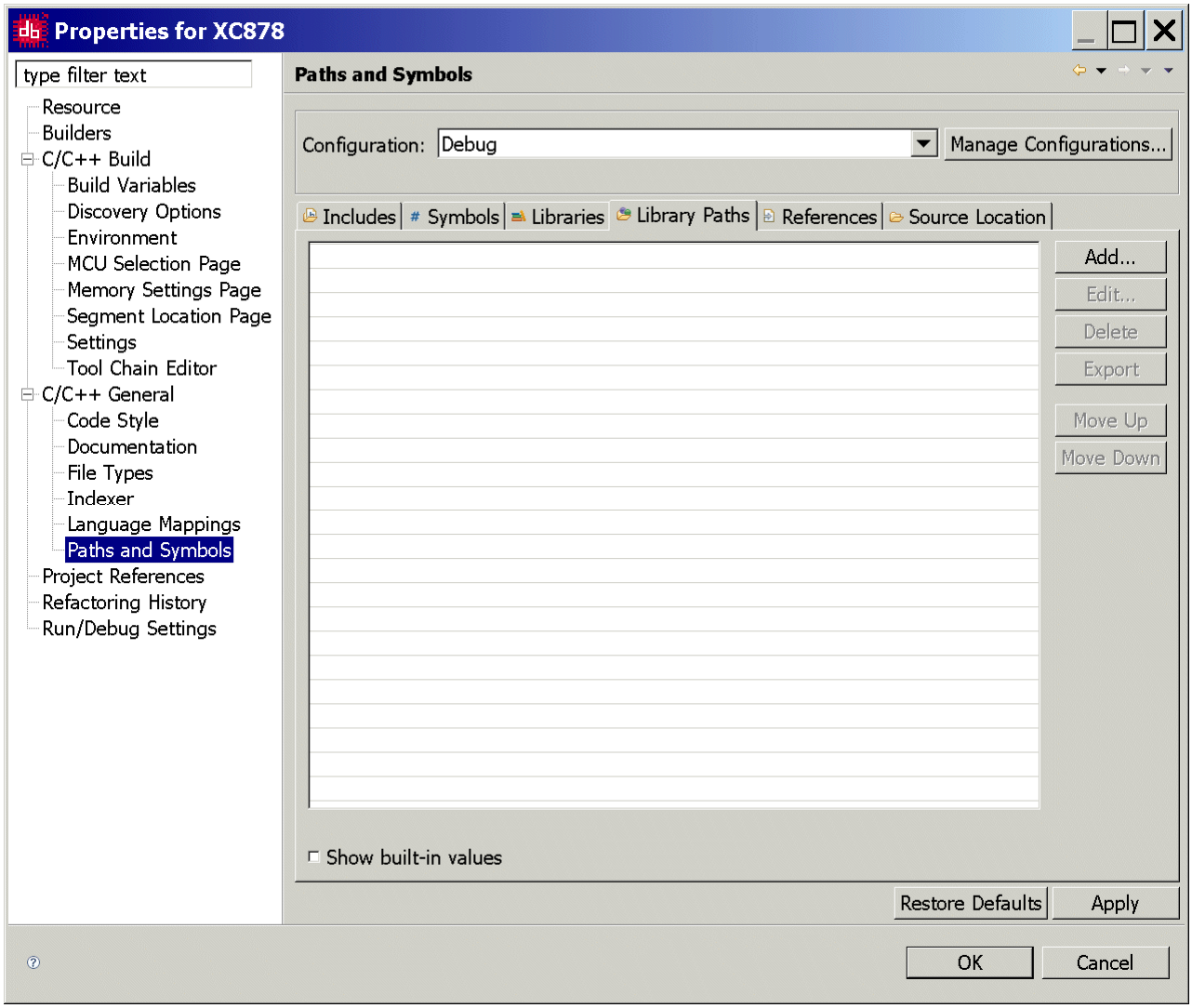
Project – Properties: C/C++ General: Paths and Symbols: Symbols:



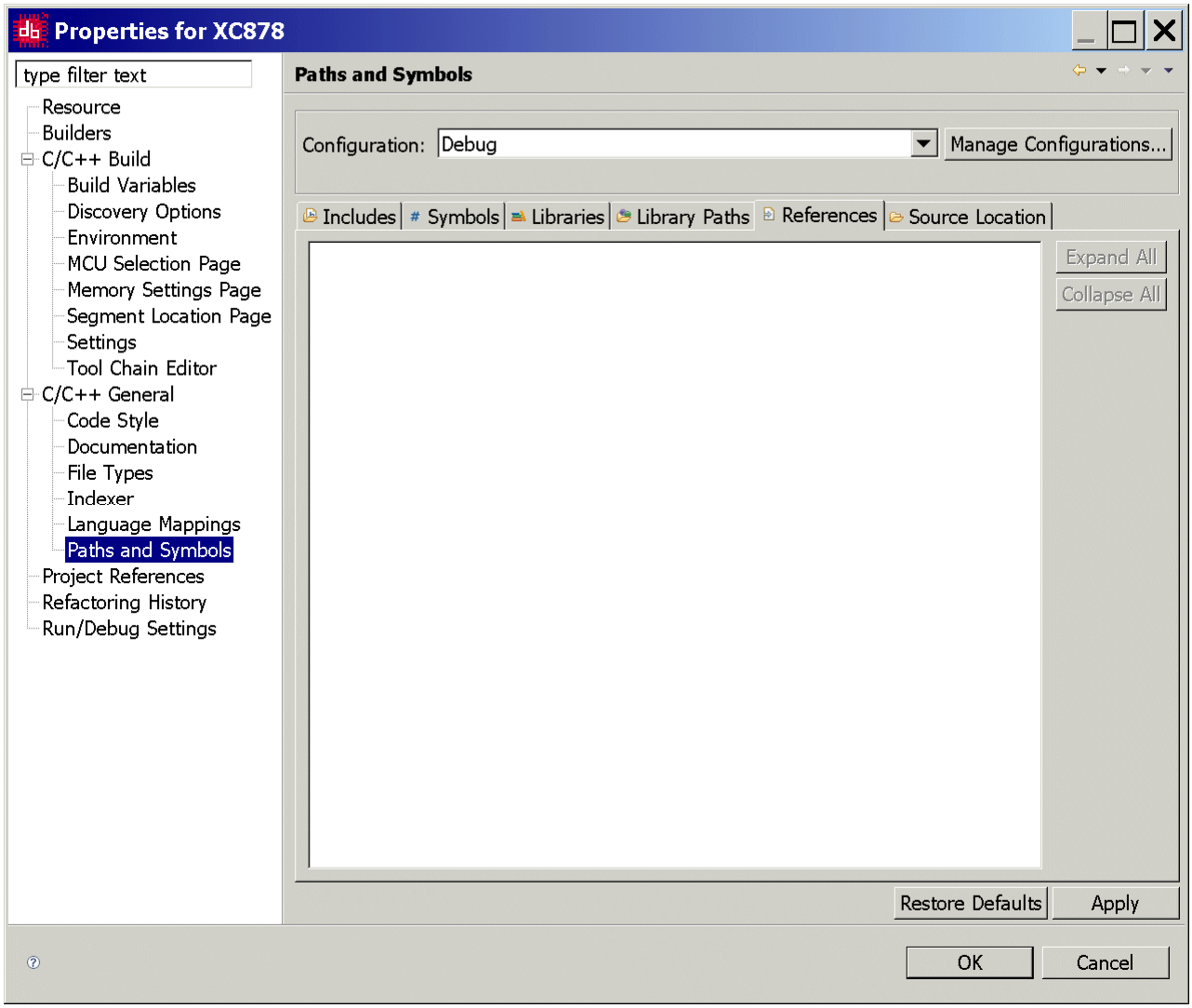
Project – Properties: C/C++ General: Paths and Symbols: Libraries:



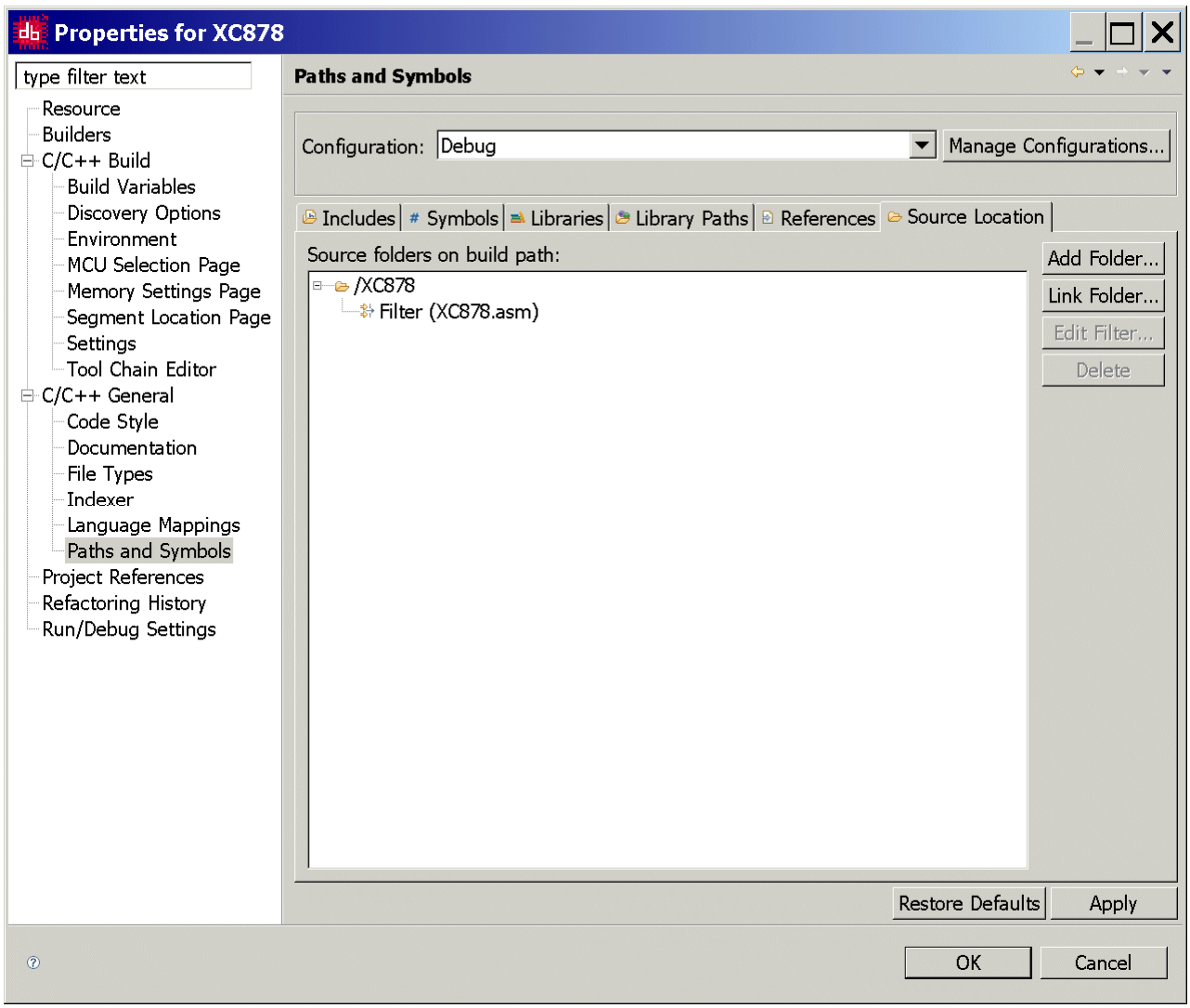
Project – Properties: C/C++ General: Paths and Symbols: Library Paths:



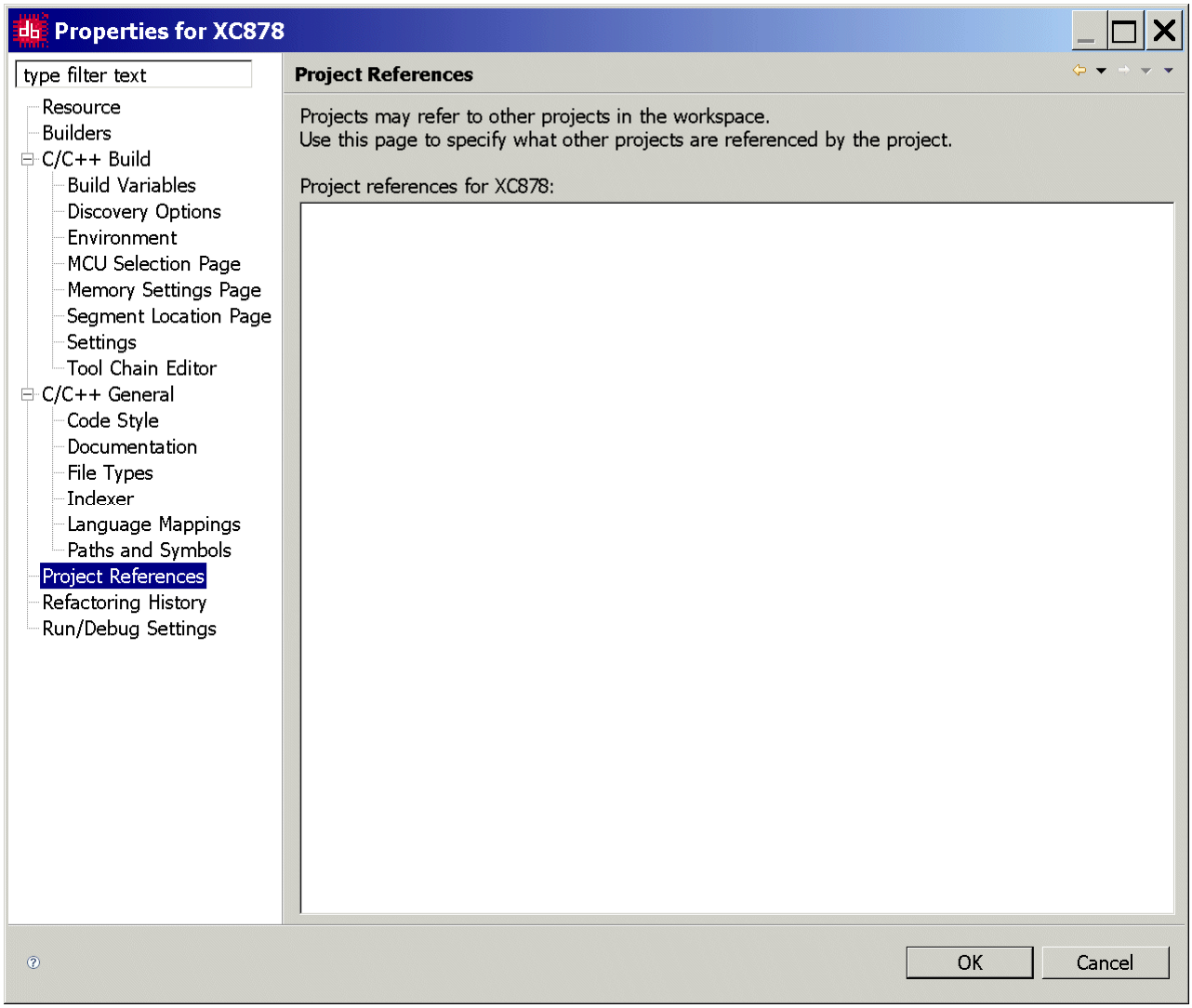
Project – Properties: C/C++ General: Paths and Symbols: References:



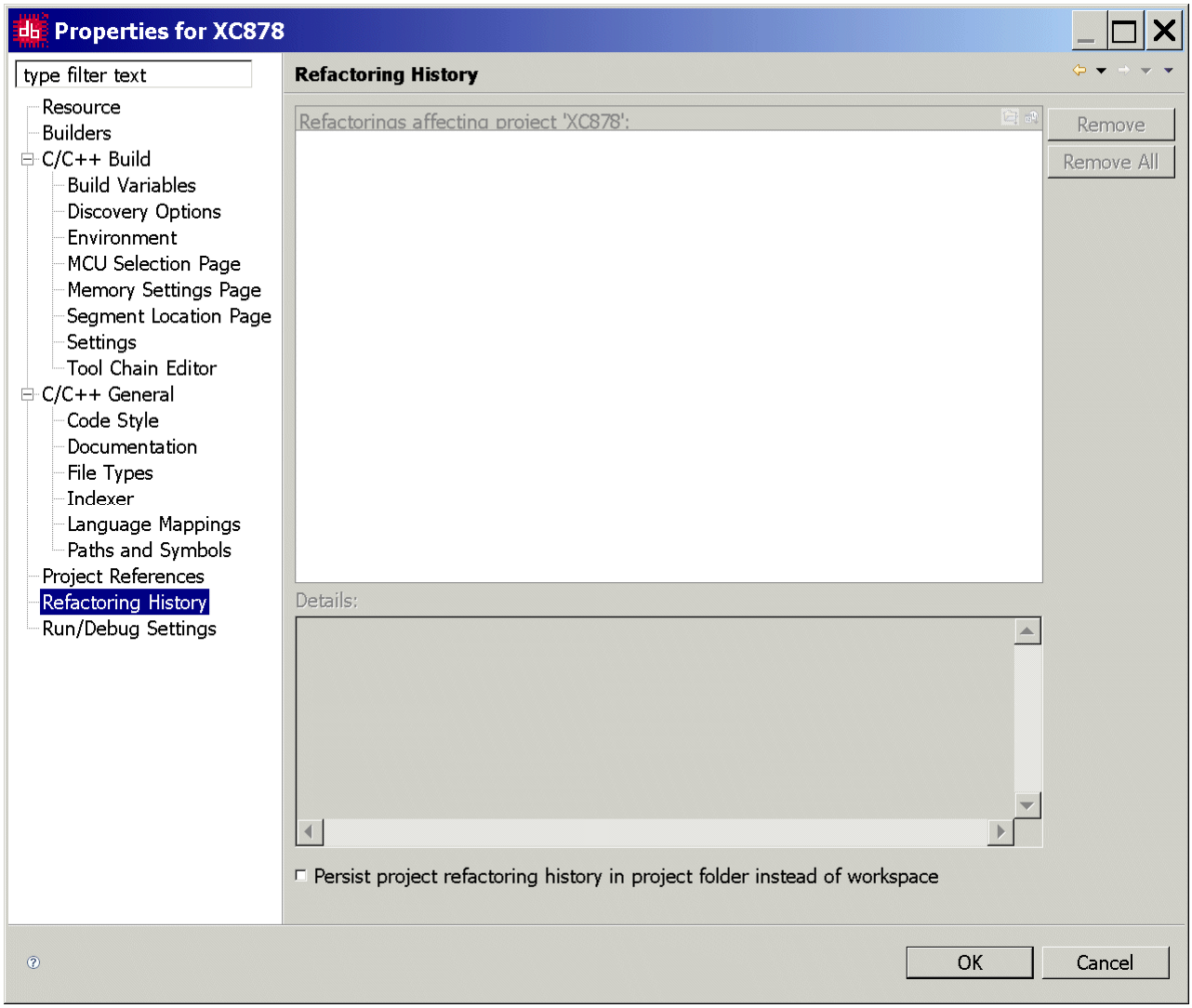
Project – Properties: C/C++ General: Paths and Symbols: Source Location:



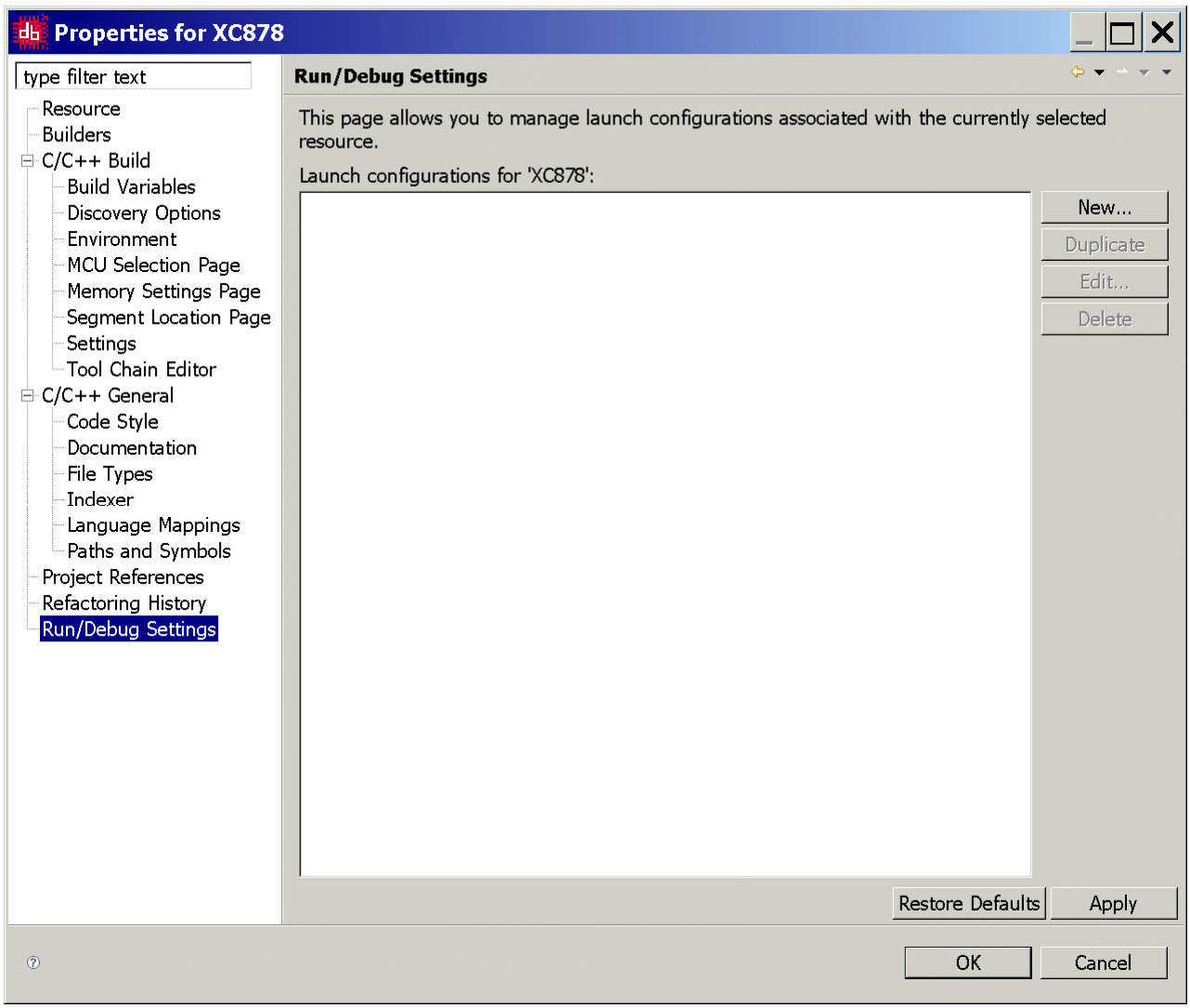
Project – Properties: Project References:



Project – Properties: Refactoring History:

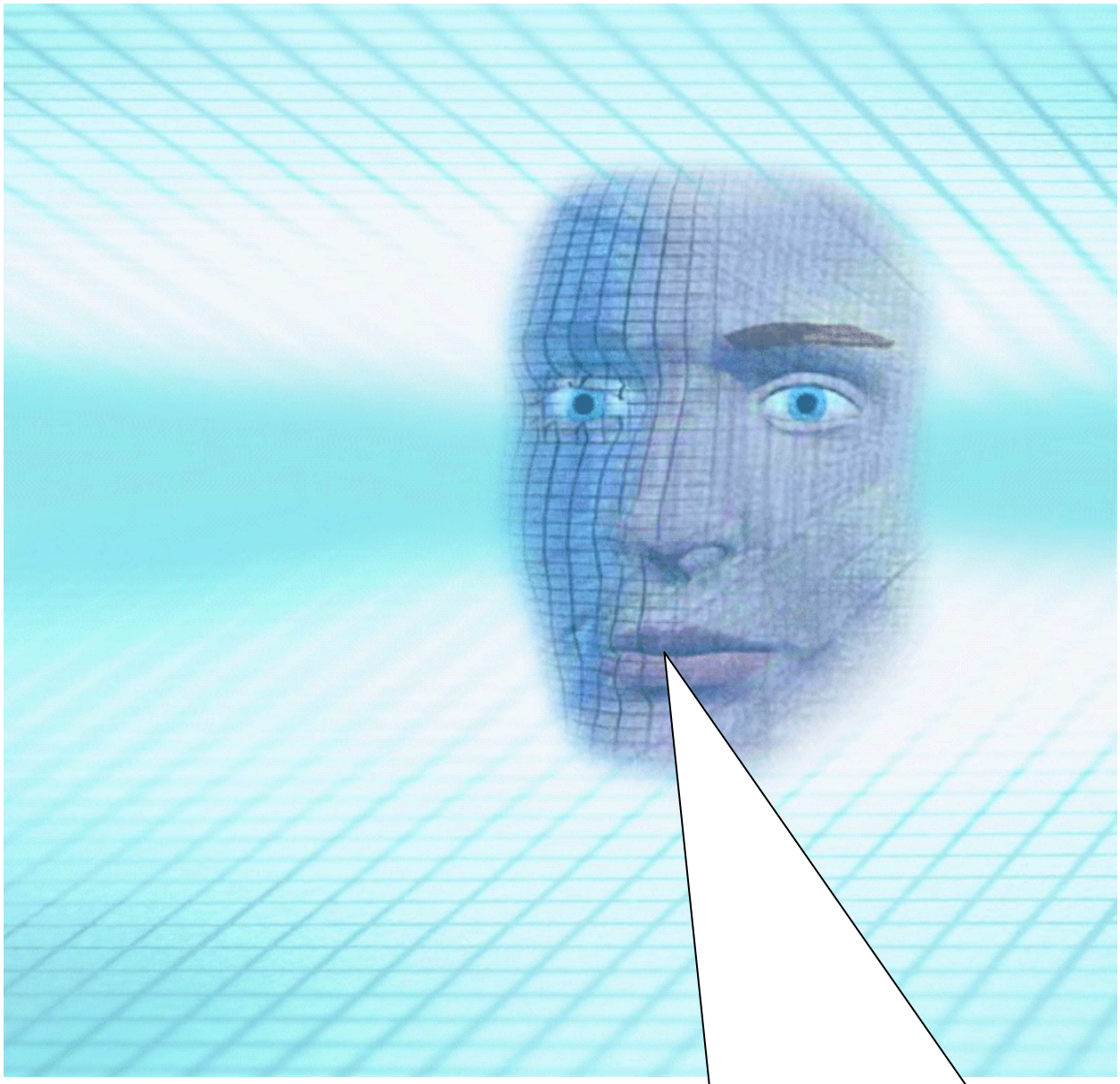


Project – Properties: Run/Debug Settings:



OK

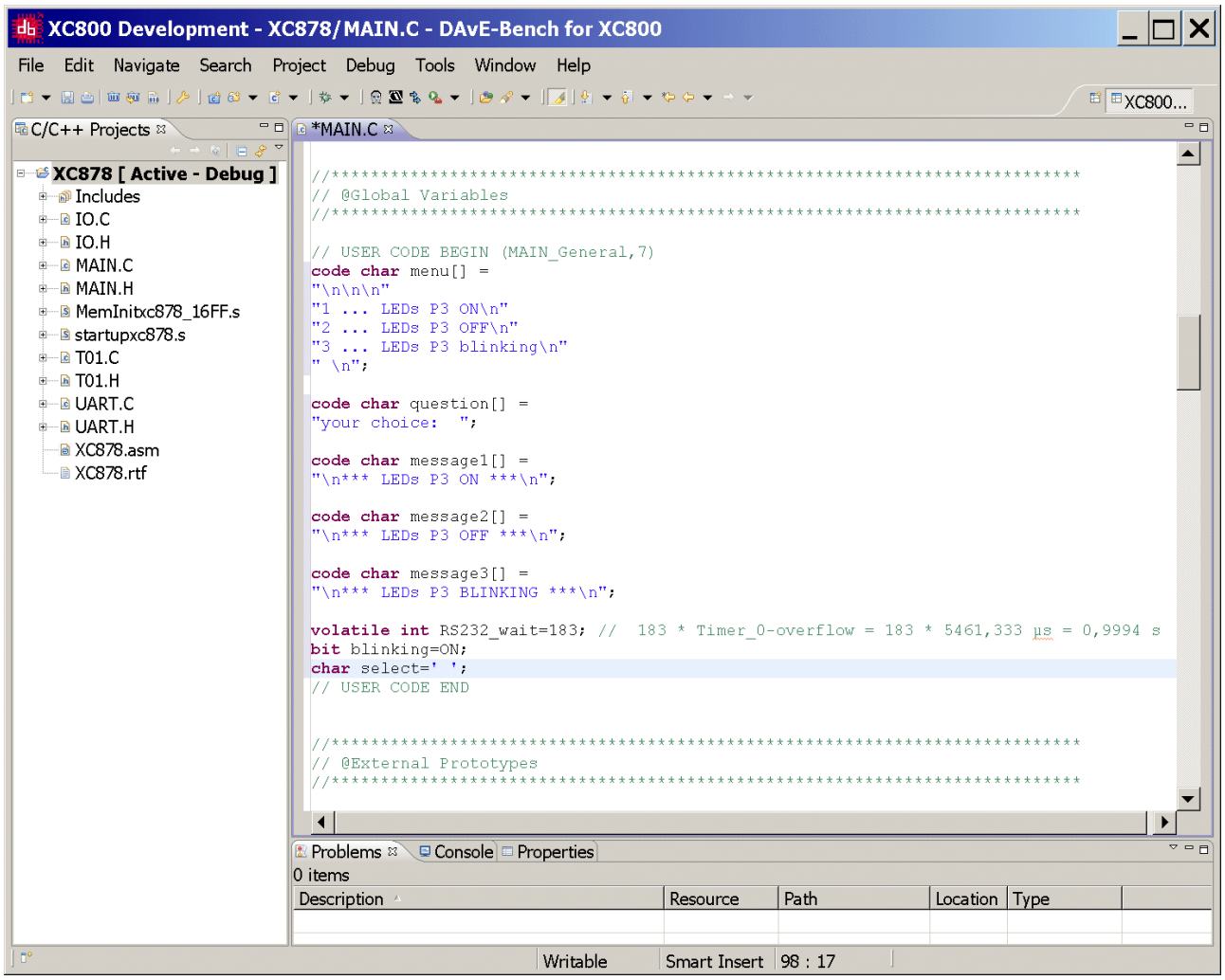
Insert your application specific program:



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!

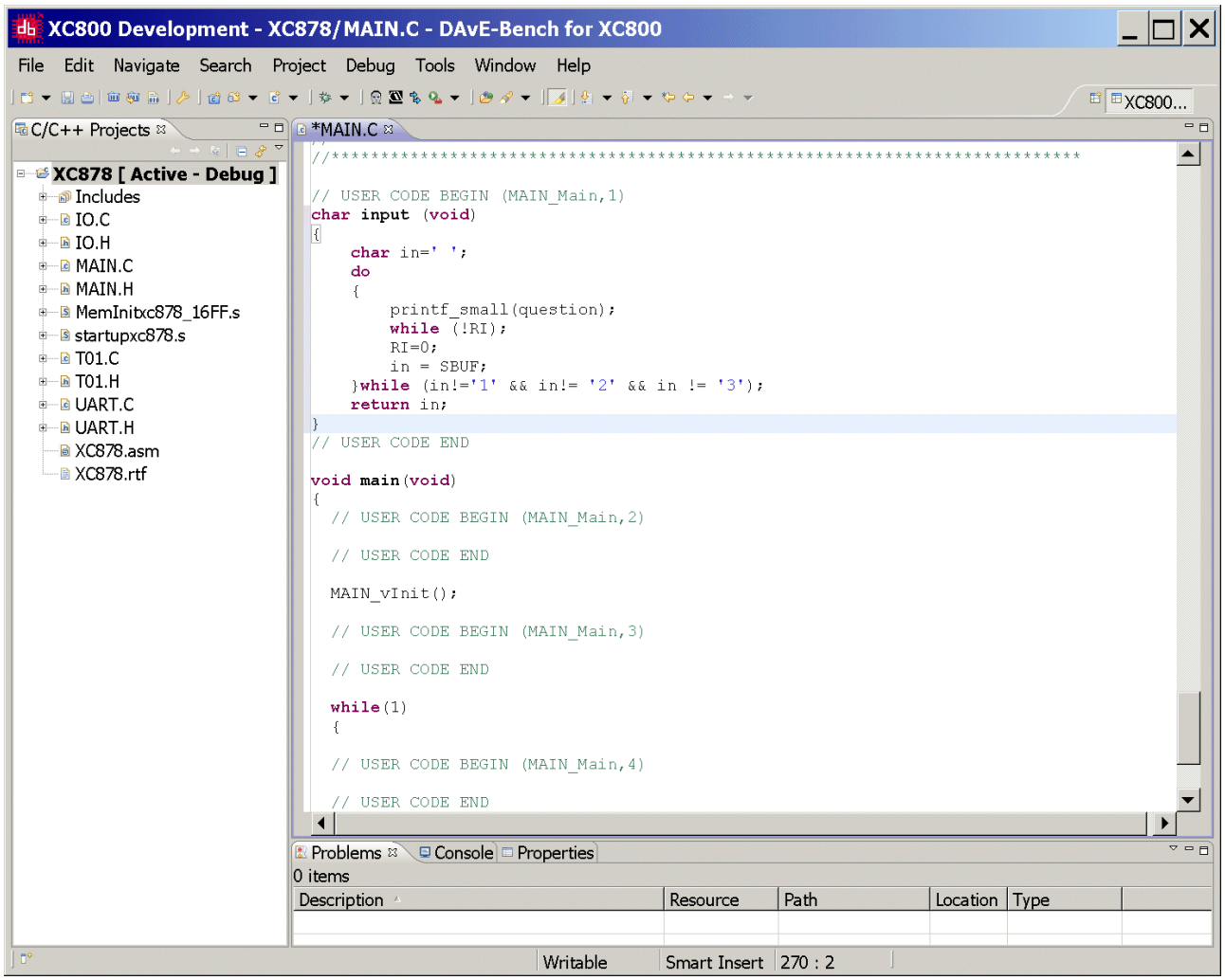
Double click MAIN.C and insert Global Variables:

```
code char menu[] =
"\n\n\n"
"1 ... LEDs P3 ON\n"
"2 ... LEDs P3 OFF\n"
"3 ... LEDs P3 blinking\n"
" \n";

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

code char message1[] =
"\n*** LEDs P3 ON ***\n";

code char message2[] =
"\n*** LEDs P3 OFF ***\n";

code char message3[] =
"\n*** LEDs P3 BLINKING ***\n";

volatile int RS232_wait=183; //  183 * Timer_0-overflow = 183 * 5461,333 µs = 0,9994 s
bit blinking=ON;
char select=' ';
```


XC800 Development - XC878/MAIN.C - DAvE-Bench for XC800
File Edit Navigate Search Project Debug Tools Window Help
C/C++ Projects
XC878 [Active - Debug]
Includes
IO.C
IO.H
MAIN.C
MAIN.H
MemInitxc878_16FF.s
startupxc878.s
T01.C
T01.H
UART.C
UART.H
XC878.asm
XC878.rtf
*MAIN.C
//***
// @Global Variables
//***
// USER CODE BEGIN (MAIN_General,7)
code char menu[] =
"\n\n\n"
"1 ... LEDs P3 ON\n"
"2 ... LEDs P3 OFF\n"
"3 ... LEDs P3 blinking\n"
" \n";
code char question[] =
"your choice: ";
code char message1[] =
"\n*** LEDs P3 ON ***\n";
code char message2[] =
"\n*** LEDs P3 OFF ***\n";
code char message3[] =
"\n*** LEDs P3 BLINKING ***\n";
volatile int RS232_wait=183; // 183 * Timer_0-overflow = 183 * 5461,333 µs = 0,9994 s
bit blinking=ON;
char select=' ';
// USER CODE END
//***
// @External Prototypes
//***
Problems
Console
Properties
0 items
Description
Resource
Path
Location
Type
Writable
Smart Insert
98 : 17

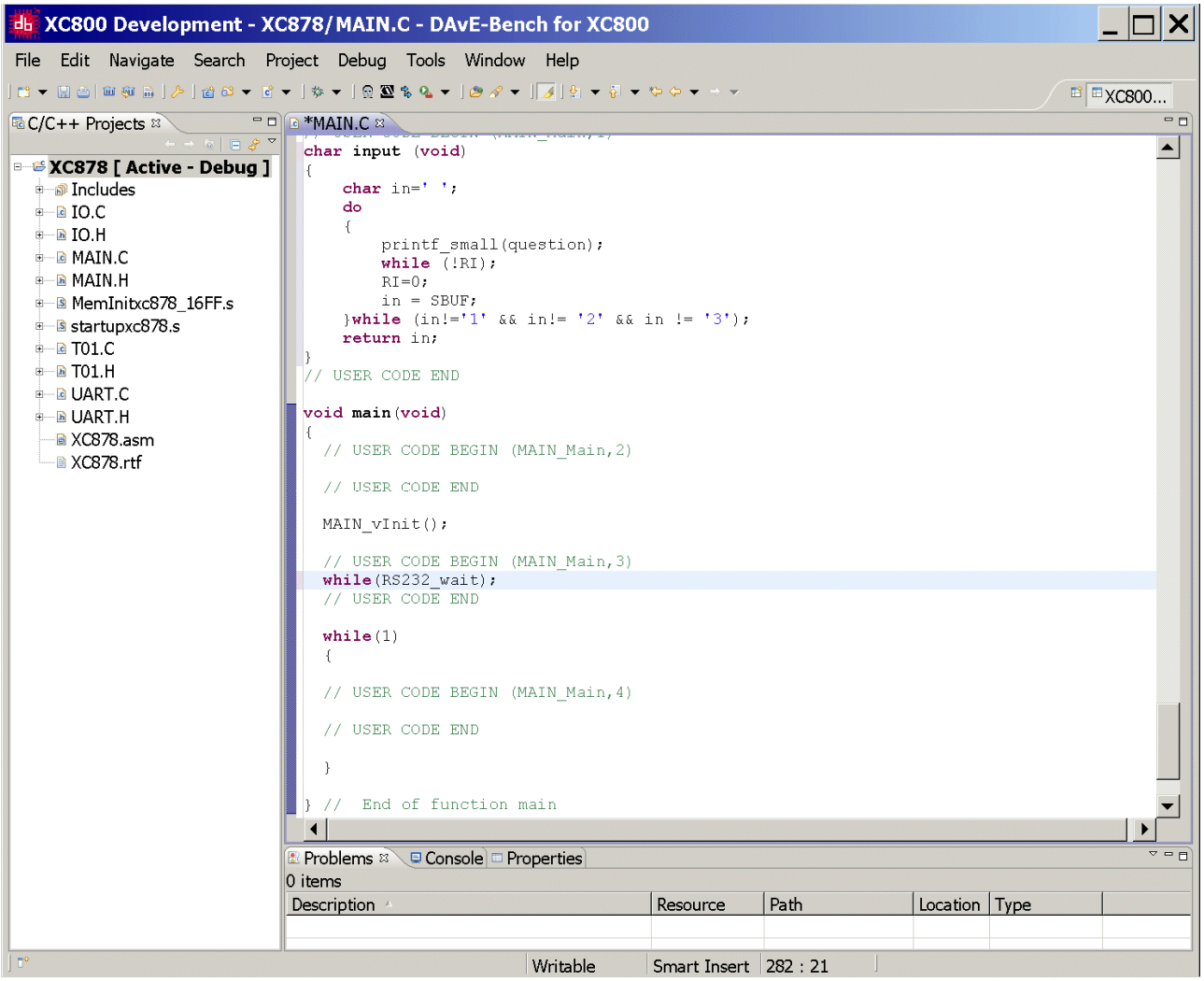
Double click MAIN.C and insert the function input():

```
char input (void)
{
        char in=' ';
        do
        {
                printf_small(question);
                while (!RI);
                RI=0;
                in = SBUF;
        }while (in!='1' && in!= '2' && in != '3');
        return in;
}
```



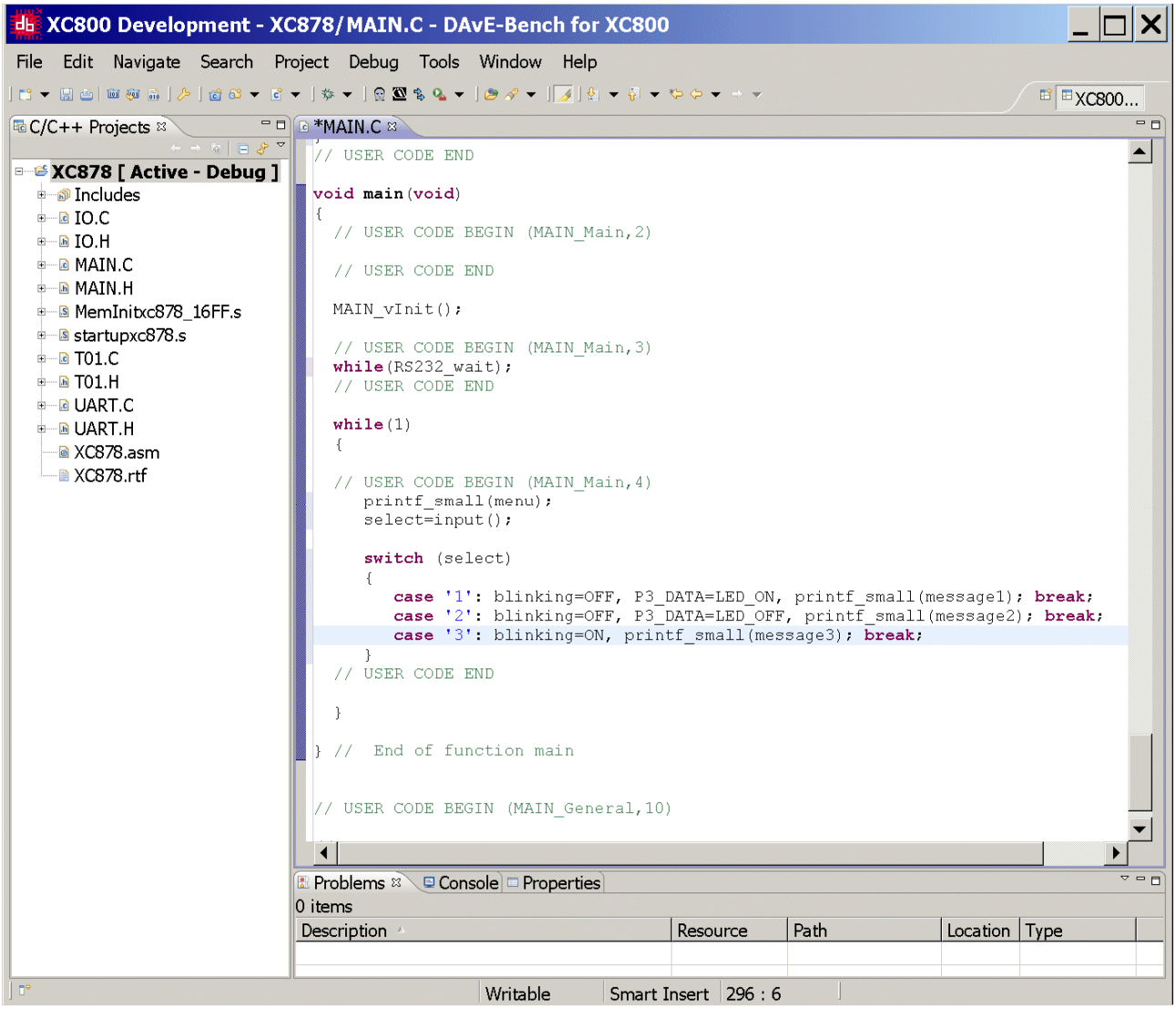
Double click MAIN.C and insert the following code in the main function:

```
while(RS232_wait);
```



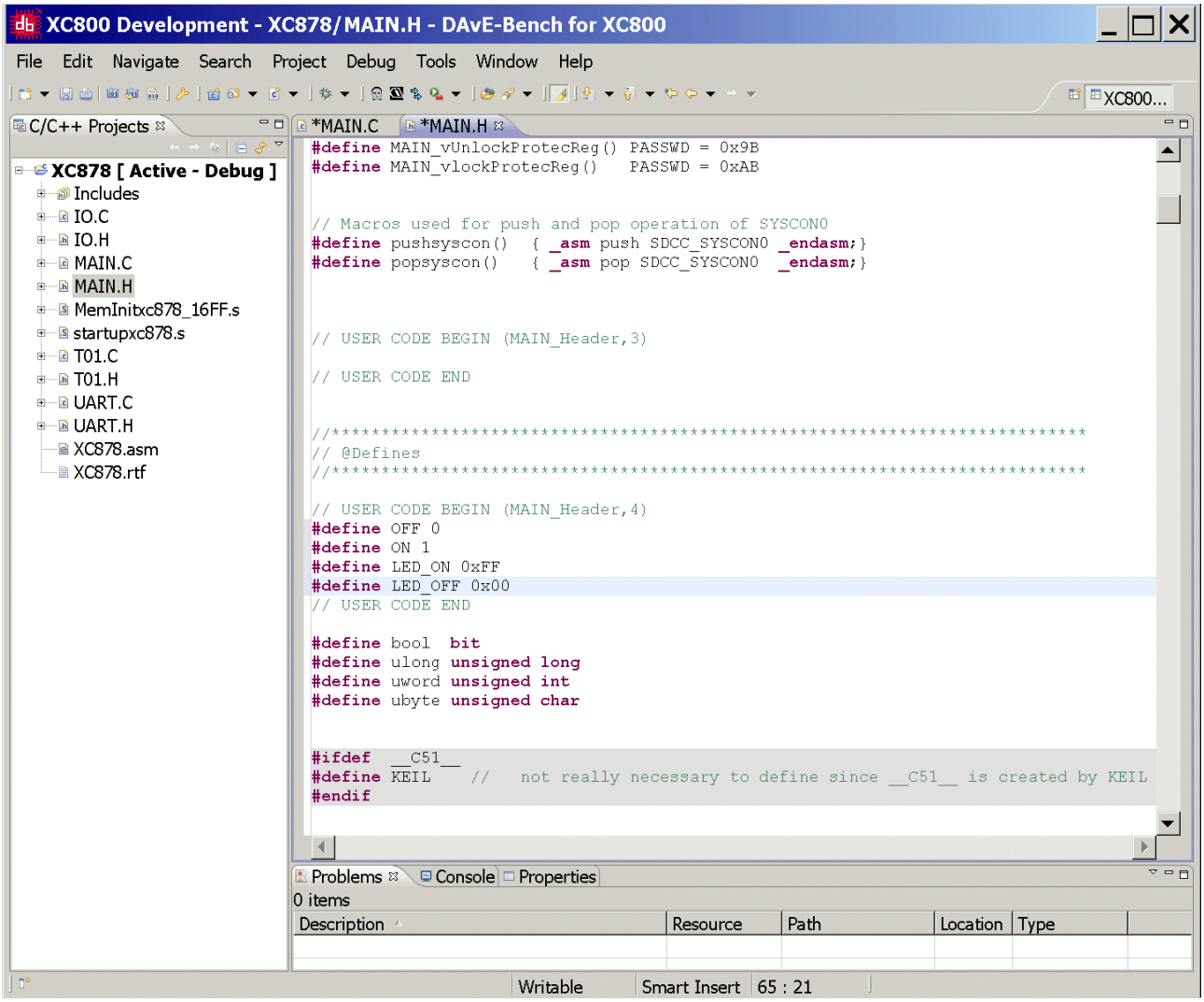
Double click MAIN.C and insert the following code in the main function into the while(1) loop:

```
printf_small(menu);
select=input();

switch (select)
{
  case '1': blinking=OFF, P3_DATA=LED_ON, printf_small(message1); break;
  case '2': blinking=OFF, P3_DATA=LED_OFF, printf_small(message2); break;
  case '3': blinking=ON, printf_small(message3); break;
}
```



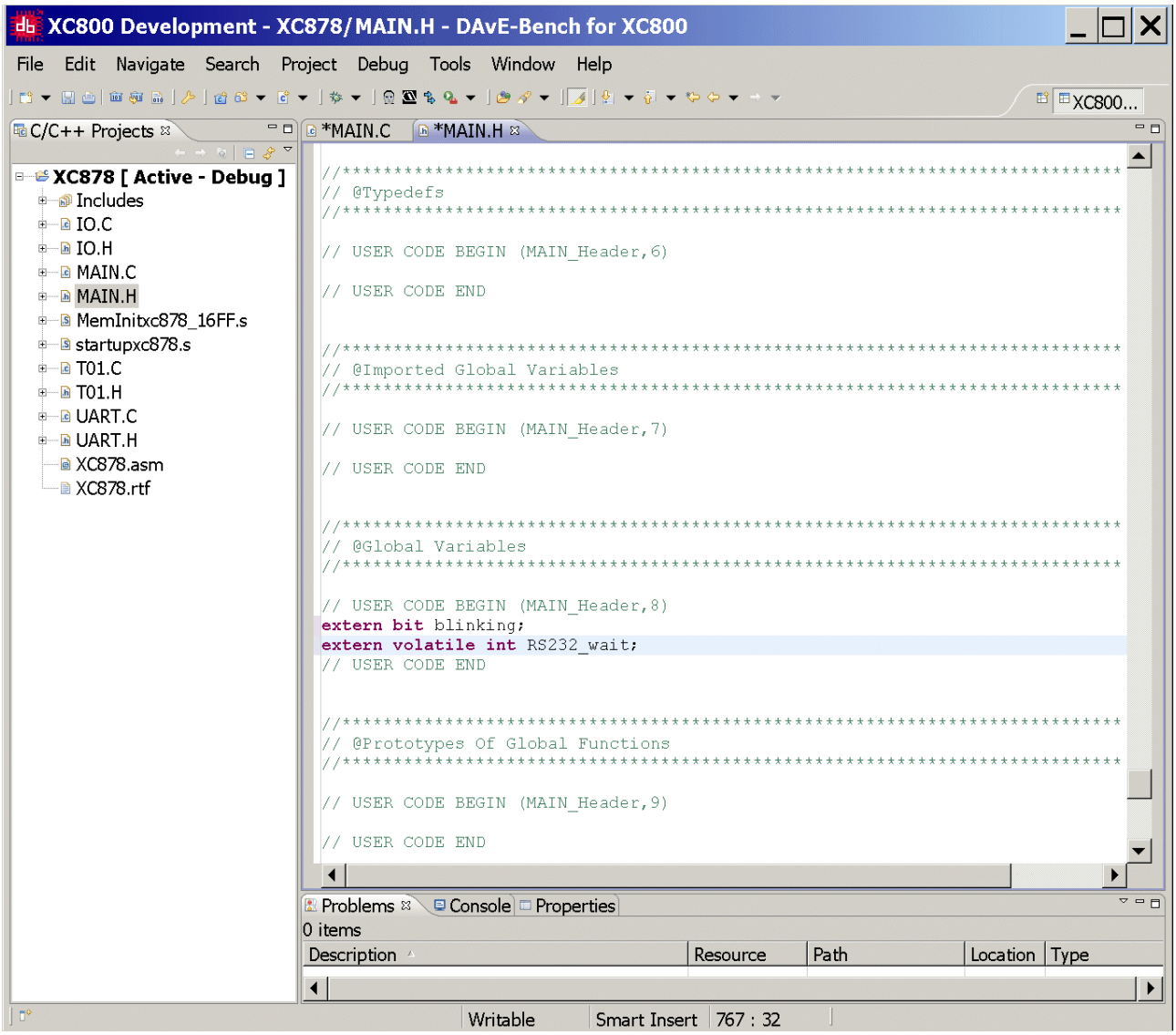
Double click Main.h and insert the following Defines:

```
#define OFF 0
#define ON 1
#define LED_ON 0xFF
#define LED_OFF 0x00
```



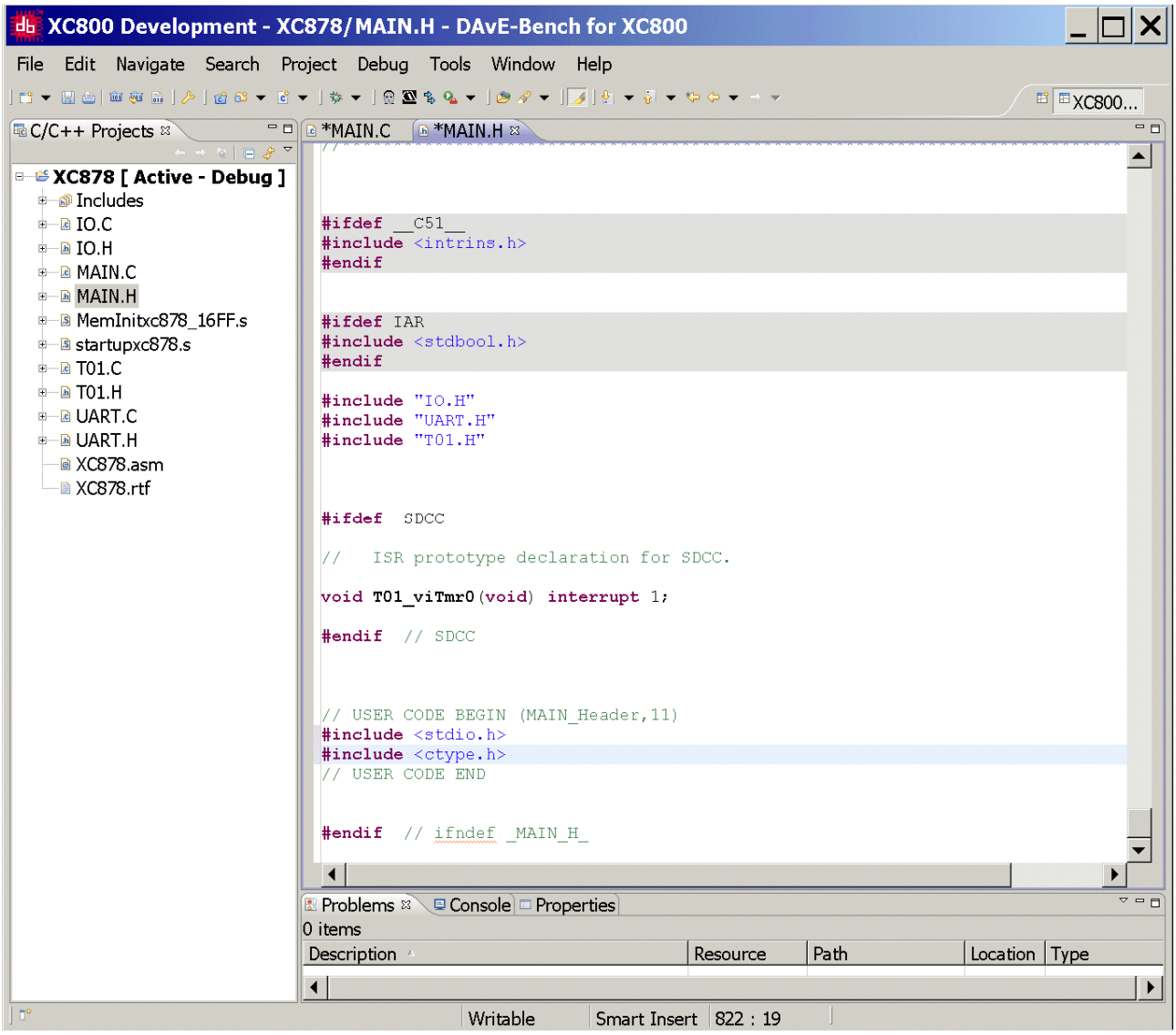
Double click Main.h and insert extern-declarations “Global Variables”:

```
extern bit blinking;
extern volatile int RS232_wait;
```



Double click Main.h and insert include files:

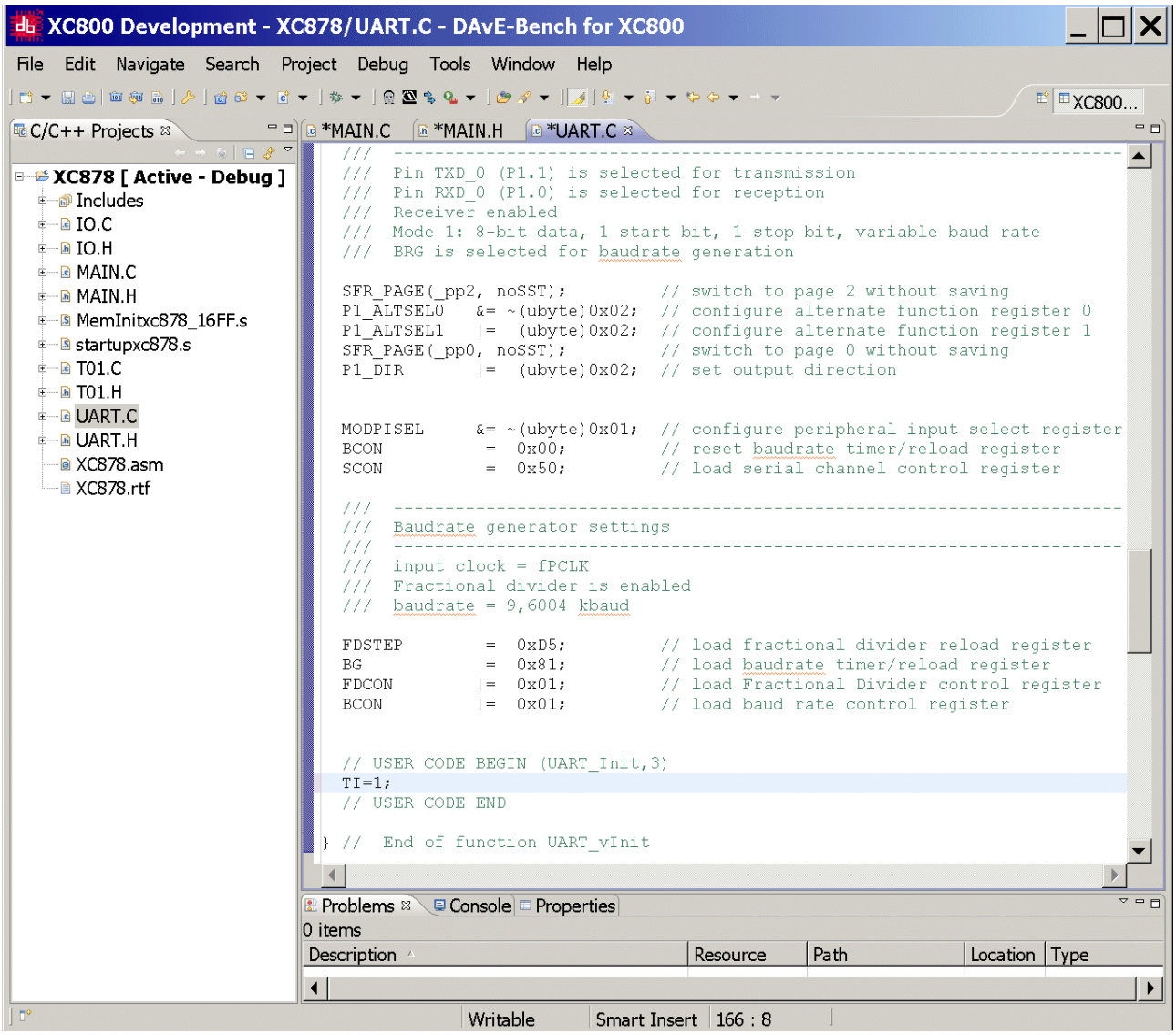
```
#include <stdio.h>
#include <ctype.h>
```



Double click UART.C

Insert code into the UART_vInit function: [to start printf_small()]:

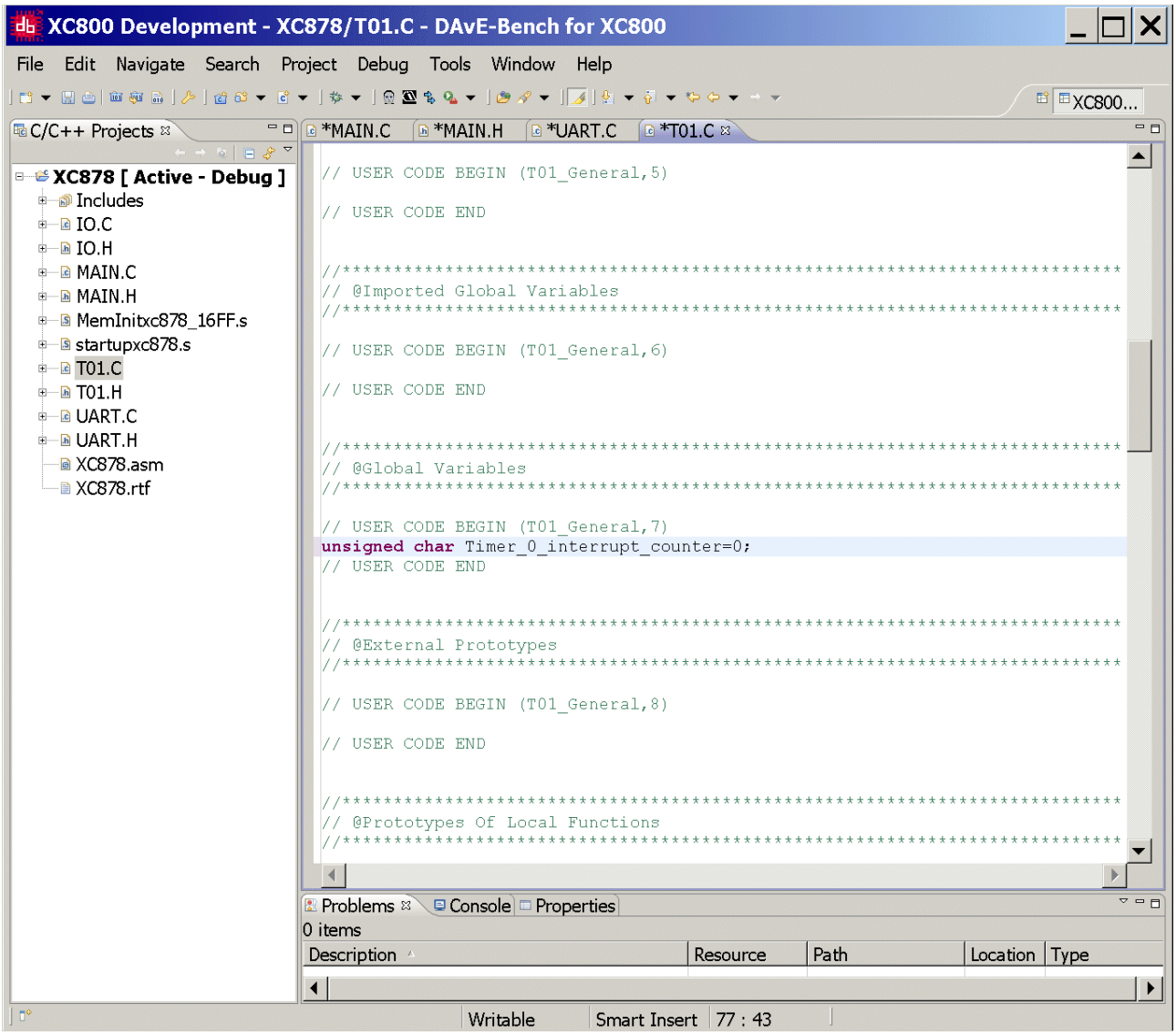
```
TI=1;
```



Double click T01.C

Insert the following global variable:

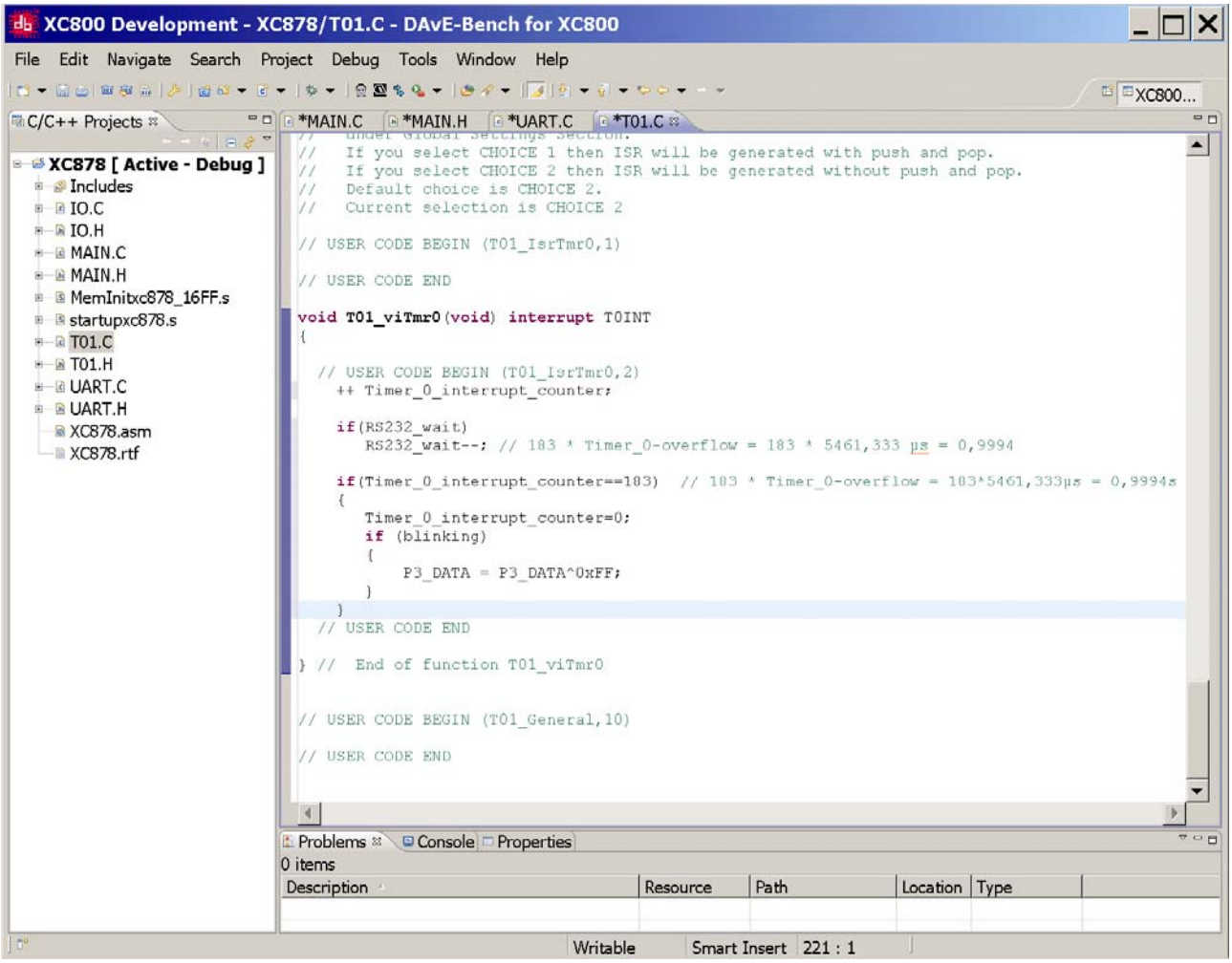
```
unsigned char Timer_0_interrupt_counter=0;
```



Double click T01.C

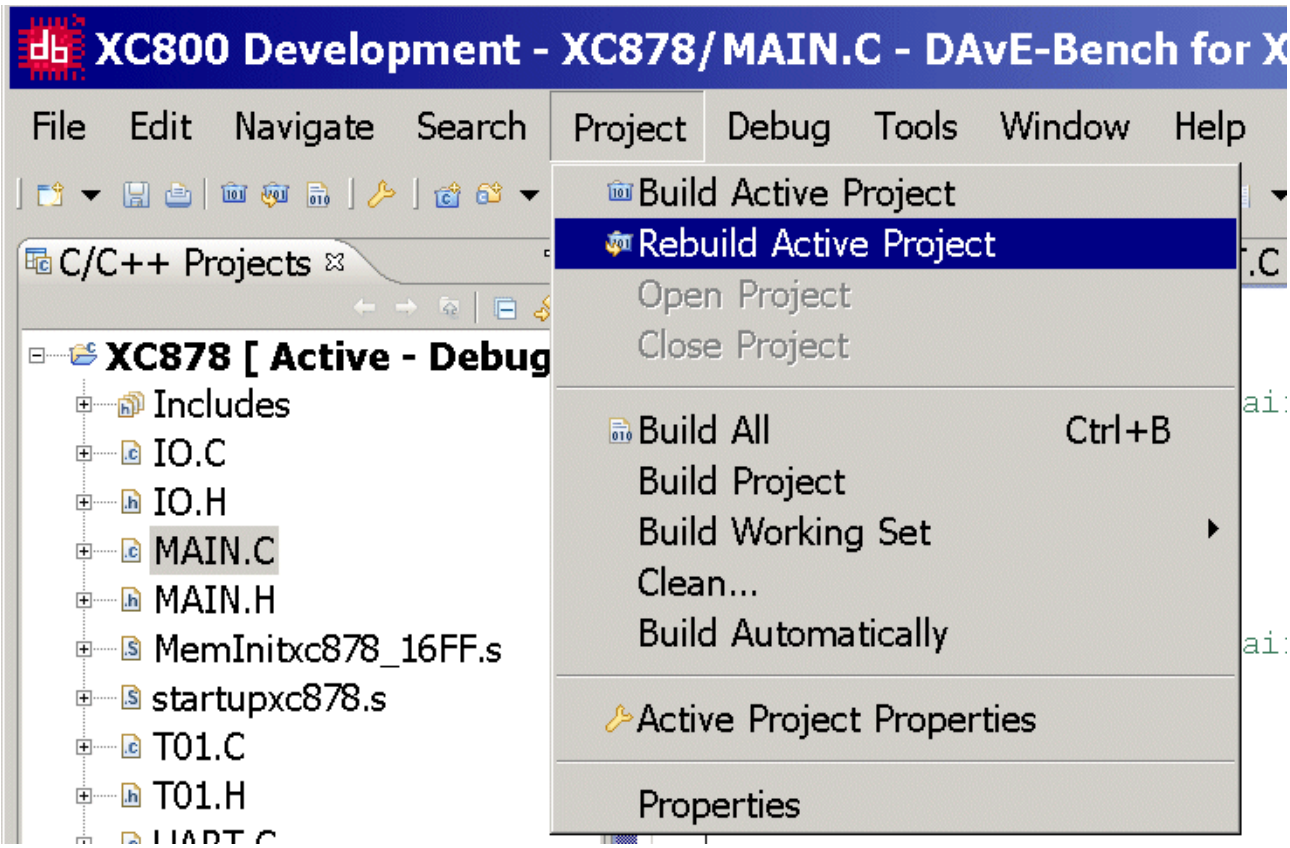
Insert code for T0 interrupt service routine:

```
++ Timer_0_interrupt_counter;

        if(RS232_wait)
                RS232_wait--; // 183 * Timer_0-overflow = 183 * 5461,333 µs = 0,9994

        if(Timer_0_interrupt_counter==183)  // 183 * Timer_0-overflow = 183*5461,333µs = 0,9994s
        {
              Timer_0_interrupt_counter=0;
              if (blinking)
              {
                    P3_DATA = P3_DATA^0xFF;
               }
         }
```

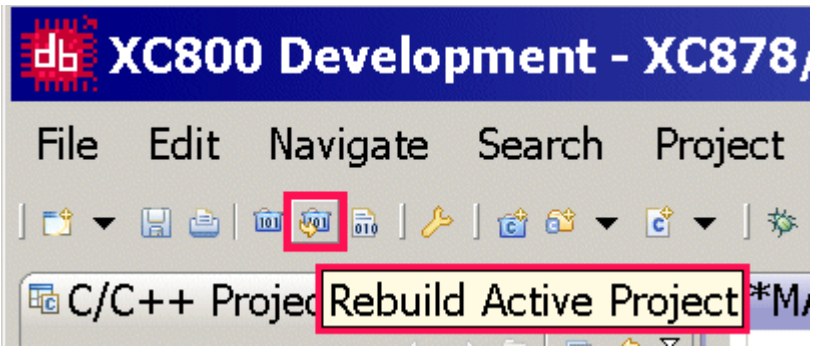


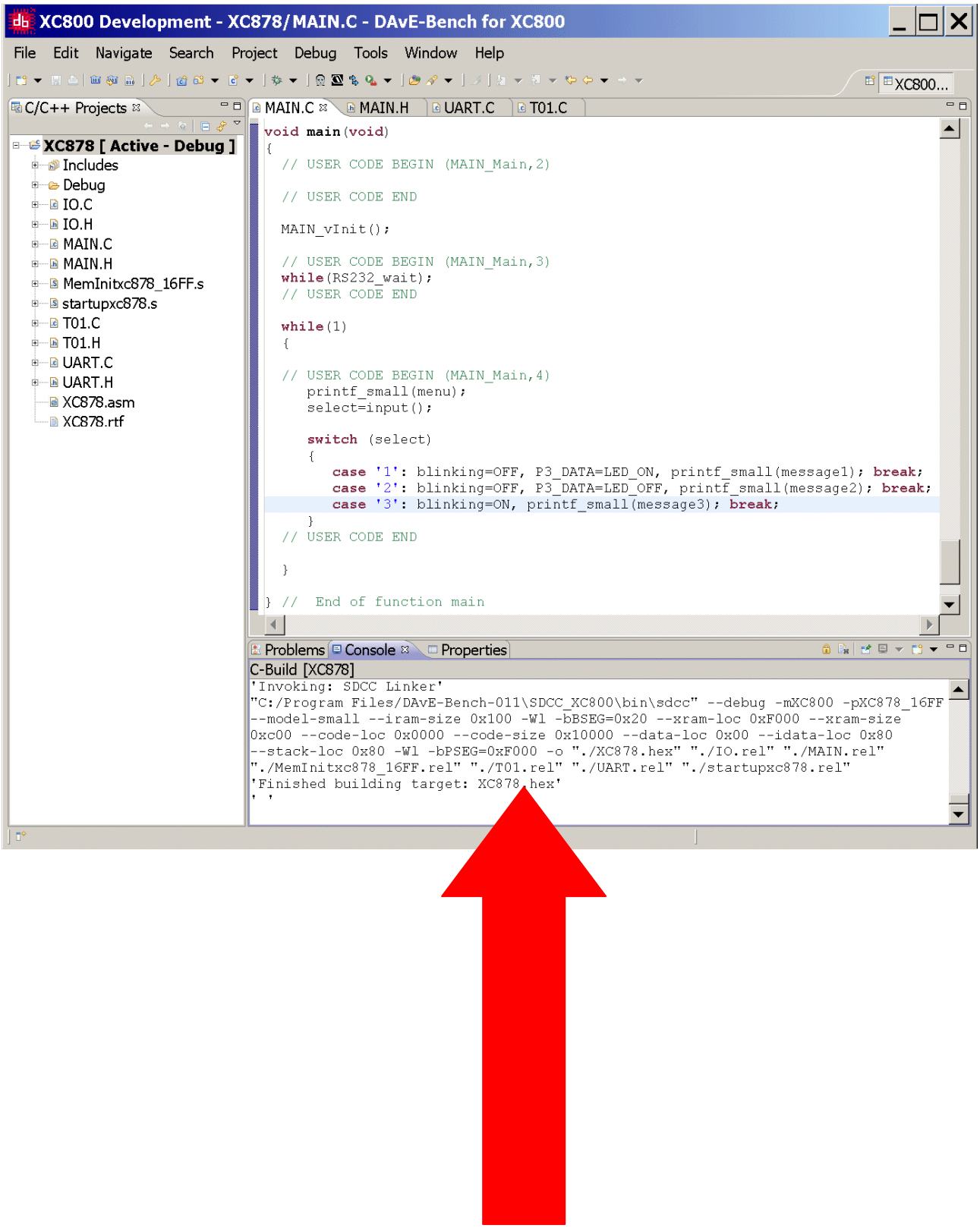
Generate your application program:

Project – Rebuild Active Project



or: click 



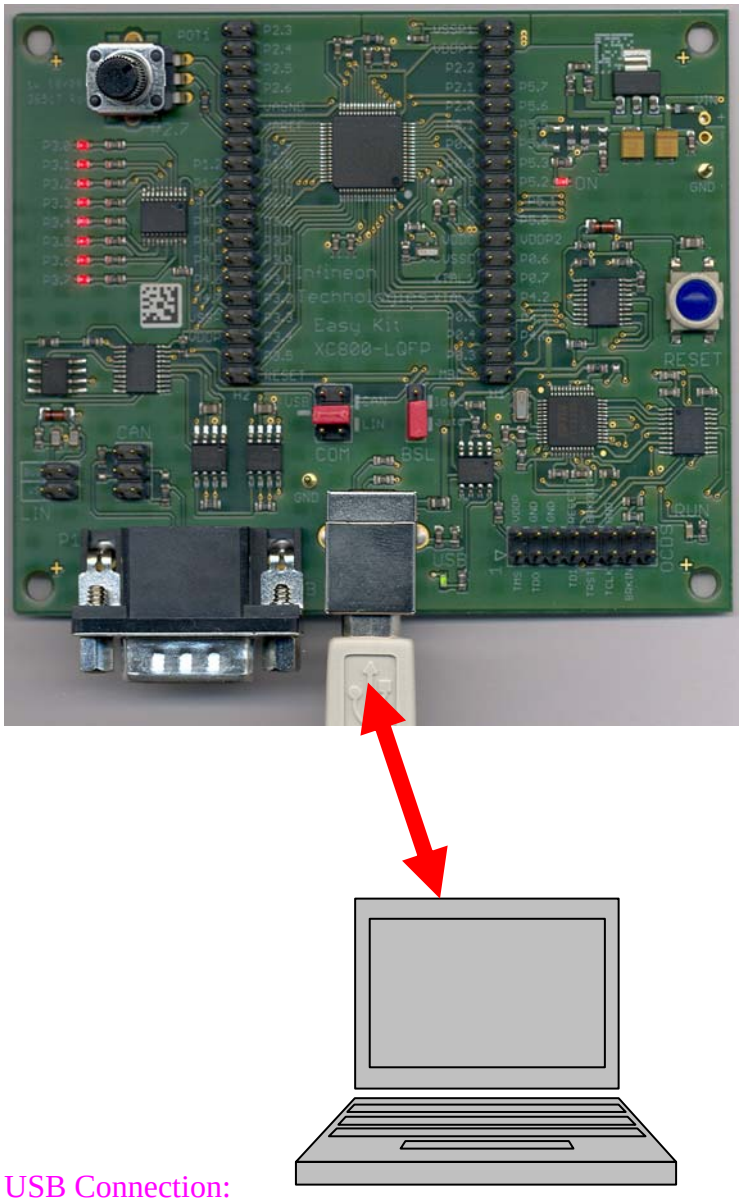

XC800 Development - XC878/MAIN.C - DAvE-Bench for XC800
File Edit Navigate Search Project Debug Tools Window Help
C/C++ Projects
XC878 [Active - Debug]
Includes
Debug
IO.C
IO.H
MAIN.C
MAIN.H
MemInitxc878_16FF.s
startupxc878.s
T01.C
T01.H
UART.C
UART.H
XC878.asm
XC878.rtf
MAIN.C MAIN.H UART.C T01.C
void main(void)
{
// USER CODE BEGIN (MAIN_Main,2)
// USER CODE END
MAIN_vInit();
// USER CODE BEGIN (MAIN_Main,3)
while(RS232_wait);
// USER CODE END
while(1)
{
// USER CODE BEGIN (MAIN_Main,4)
printf_small(menu);
select=input();
switch (select)
{
case '1': blinking=OFF, P3_DATA=LED_ON, printf_small(message1); break;
case '2': blinking=OFF, P3_DATA=LED_OFF, printf_small(message2); break;
case '3': blinking=ON, printf_small(message3); break;
}
// USER CODE END
}
} // End of function main
Problems Console Properties
C-Build [XC878]
'Invoking: SDCC Linker'
"C:/Program Files/DAvE-Bench-011\SDCC_XC800\bin\sdcc" --debug -mXC800 -pXC878_16FF
--model-small --iram-size 0x100 -Wl -bBSEG=0x20 --xram-loc 0xF000 --xram-size
0xc00 --code-loc 0x0000 --code-size 0x10000 --data-loc 0x00 --idata-loc 0x80
--stack-loc 0x80 -Wl -bPSEG=0xF000 -o "./XC878.hex" "./IO.rel" "./MAIN.rel"
"./MemInitxc878_16FF.rel" "./T01.rel" "./UART.rel" "./startupxc878.rel"
'Finished building target: XC878.hex'

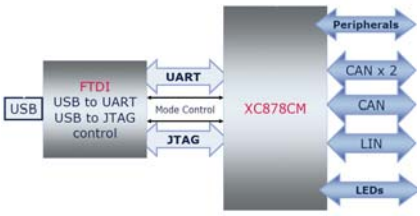
```
**** Build of configuration Debug for project XC878 ****

C:\Program Files\DAvE-Bench-011\SDCC_UTILS\make all
'Building file: ../IO.C'
'Invoking: SDCC Compiler'
"C:/Program Files/DAvE-Bench-011\SDCC_XC800\bin\sdcc" -mXC800 -pXC878_16FF -
-model-small  -I"C:/Program Files/DAvE-Bench-011\SDCC_XC800\include" -
I"C:/Program Files/DAvE-Bench-011\SDCC_XC800\include\xc800" -I"C:/Program
Files/DAvE-Bench-011\SDCC_XC800\include\asm\xc800" --opt-code-size --
nooverlay --noinduction --debug -S  -o "IO.s" "../IO.C"
'Finished building: ../IO.C'
' '
'Building file: IO.s'
'Invoking: SDCC Assembler'
"C:/Program Files/DAvE-Bench-011\SDCC_XC800\bin\as-xc800" -plosgffcx "IO.s"
-O "IO.rel"
'Finished building: IO.s'
' '
'Building file: ../MAIN.C'
'Invoking: SDCC Compiler'
"C:/Program Files/DAvE-Bench-011\SDCC_XC800\bin\sdcc" -mXC800 -pXC878_16FF -
-model-small  -I"C:/Program Files/DAvE-Bench-011\SDCC_XC800\include" -
I"C:/Program Files/DAvE-Bench-011\SDCC_XC800\include\xc800" -I"C:/Program
Files/DAvE-Bench-011\SDCC_XC800\include\asm\xc800" --opt-code-size --
nooverlay --noinduction --debug -S  -o "MAIN.s" "../MAIN.C"
'Finished building: ../MAIN.C'
' '
'Building file: MAIN.s'
'Invoking: SDCC Assembler'
"C:/Program Files/DAvE-Bench-011\SDCC_XC800\bin\as-xc800" -plosgffcx
"MAIN.s" -O "MAIN.rel"
'Finished building: MAIN.s'
' '
'Building file: ../MemInitxc878_16FF.s'
'Invoking: SDCC Assembler'
"C:/Program Files/DAvE-Bench-011\SDCC_XC800\bin\as-xc800" -plosgffcx
"../MemInitxc878_16FF.s" -O "MemInitxc878_16FF.rel"
'Finished building: ../MemInitxc878_16FF.s'
' '
'Building file: ../T01.C'
'Invoking: SDCC Compiler'
"C:/Program Files/DAvE-Bench-011\SDCC_XC800\bin\sdcc" -mXC800 -pXC878_16FF -
-model-small  -I"C:/Program Files/DAvE-Bench-011\SDCC_XC800\include" -
I"C:/Program Files/DAvE-Bench-011\SDCC_XC800\include\xc800" -I"C:/Program
Files/DAvE-Bench-011\SDCC_XC800\include\asm\xc800" --opt-code-size --
nooverlay --noinduction --debug -S  -o "T01.s" "../T01.C"
'Finished building: ../T01.C'
' '
'Building file: T01.s'
'Invoking: SDCC Assembler'
"C:/Program Files/DAvE-Bench-011\SDCC_XC800\bin\as-xc800" -plosgffcx "T01.s"
-O "T01.rel"
```

```
'Finished building: T01.s'
' '
'Building file: ../UART.C'
'Invoking: SDCC Compiler'
"C:/Program Files/DAvE-Bench-011\SDCC_XC800\bin\sdcc" -mXC800 -pXC878_16FF -
-model-small  -I"C:/Program Files/DAvE-Bench-011\SDCC_XC800\include" -
I"C:/Program Files/DAvE-Bench-011\SDCC_XC800\include\xc800" -I"C:/Program
Files/DAvE-Bench-011\SDCC_XC800\include\asm\xc800" --opt-code-size --
nooverlay --noinduction --debug -S  -o "UART.s" "../UART.C"
'Finished building: ../UART.C'
' '
'Building file: UART.s'
'Invoking: SDCC Assembler'
"C:/Program Files/DAvE-Bench-011\SDCC_XC800\bin\as-xc800" -plosgffcx
"UART.s" -O "UART.rel"
MOV dir(0x82),dir(0x99) found at 1638 of UART.s
'Finished building: UART.s'
' '
'Building file: ../startupxc878.s'
'Invoking: SDCC Assembler'
"C:/Program Files/DAvE-Bench-011\SDCC_XC800\bin\as-xc800" -plosgffcx
"../startupxc878.s" -O "startupxc878.rel"
'Finished building: ../startupxc878.s'
' '
'Building target: XC878.hex'
'Invoking: SDCC Linker'
"C:/Program Files/DAvE-Bench-011\SDCC_XC800\bin\sdcc" --debug -mXC800 -
pXC878_16FF --model-small --iram-size 0x100 -Wl -bBSEG=0x20 --xram-loc
0xF000 --xram-size 0xc00 --code-loc 0x0000 --code-size 0x10000 --data-loc
0x00 --idata-loc 0x80 --stack-loc 0x80 -Wl -bPSEG=0xF000 -o "./XC878.hex"
"./IO.rel" "./MAIN.rel" "./MemInitxc878_16FF.rel" "./T01.rel" "./UART.rel"
"./startupxc878.rel"
'Finished building target: XC878.hex'
                                        ' '
```

5.) Using the debugger (DAvE Bench):

Make sure that the XC878 Easy Kit is still connected to the host computer:





USB Connection:
.) used for: UART communication (the UART/RS232/serial interface is available via USB as a virtual COM port of the second USB channel of the FTDI FT2232 Dual USB to UART/JTAG interface).
.) used for: On-Chip-Flash-Programming and Debugging (first USB channel of the FTDI FT2232 Dual USB to UART/JTAG interface).
.) the USB connection works also as the power supply.

U-SPY:

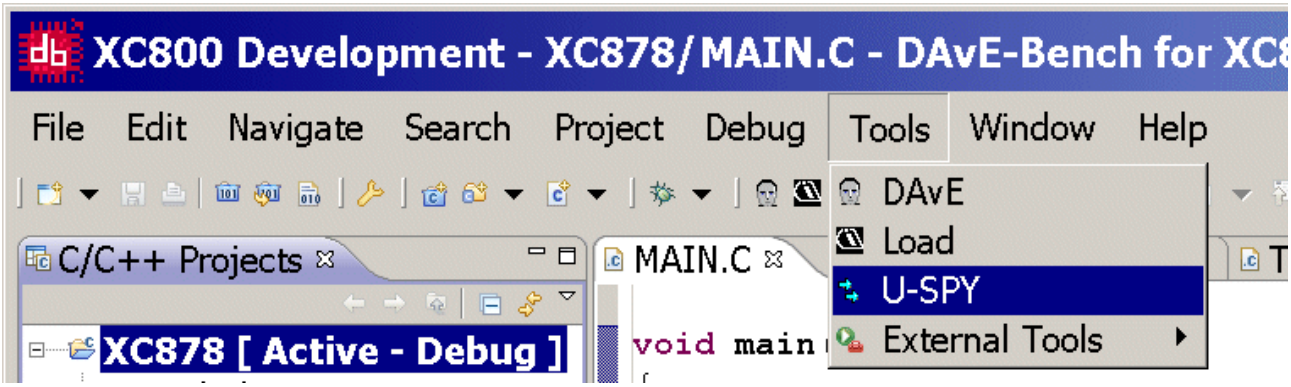
Note:
Now we need a terminal program which is able to handle our virtual COM port (COM12)!
As an example of “any terminal program” we are going to use U-SPY.

U-SPY can be found either on the XC878 Easy Kit CD (USPY_install.exe)

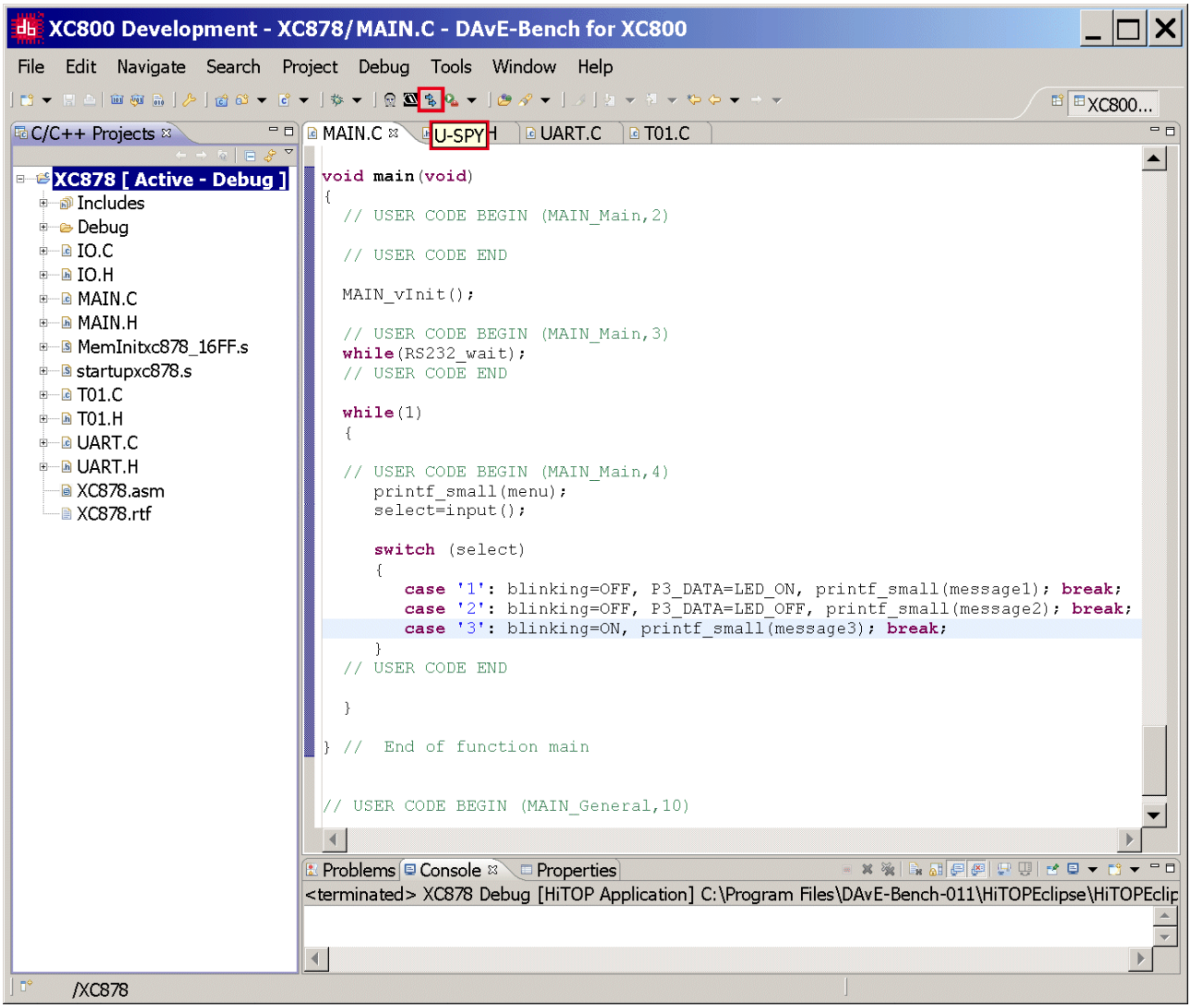
Do not install U-SPY from this CD!

or in DAvE-Bench (already installed!):

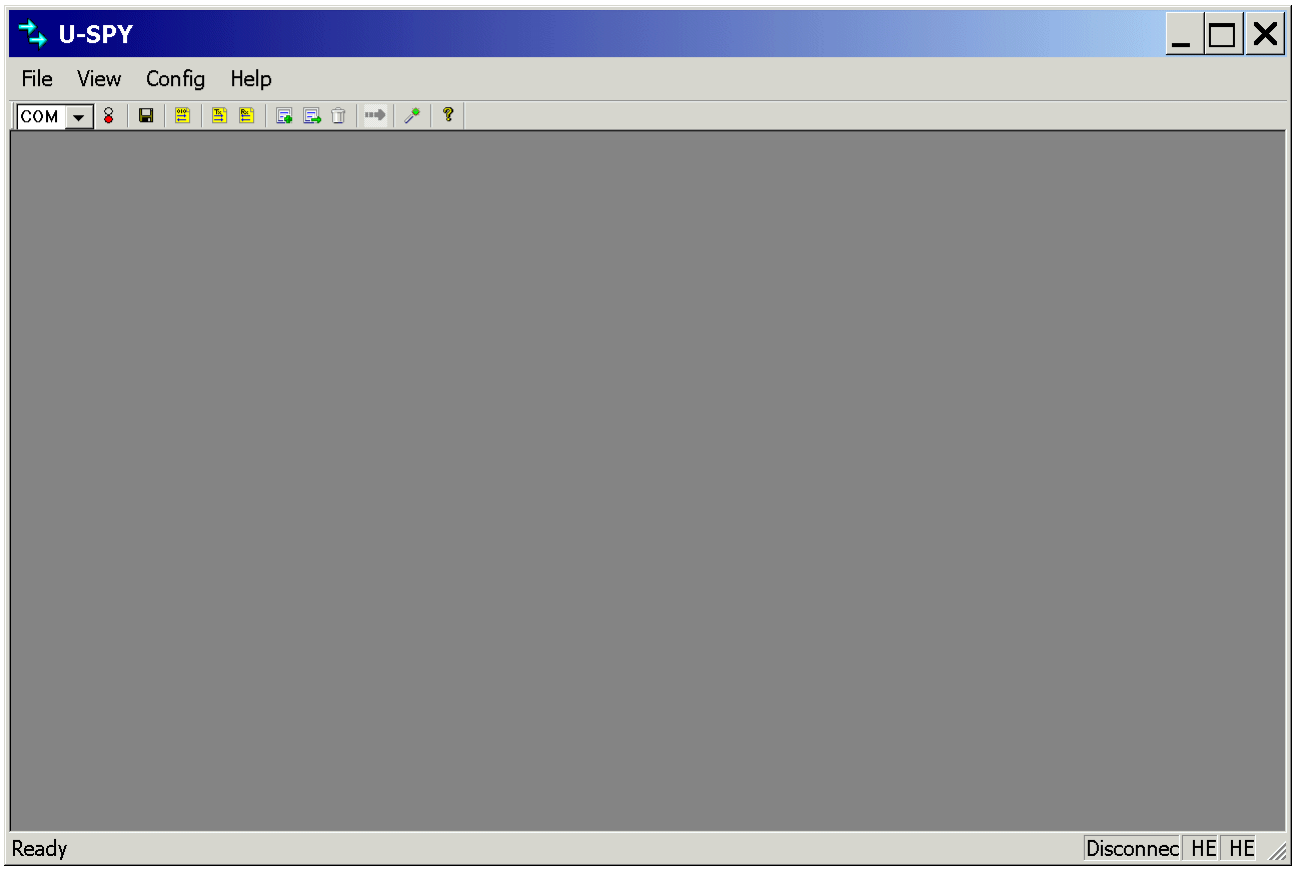
Tools – U-SPY:



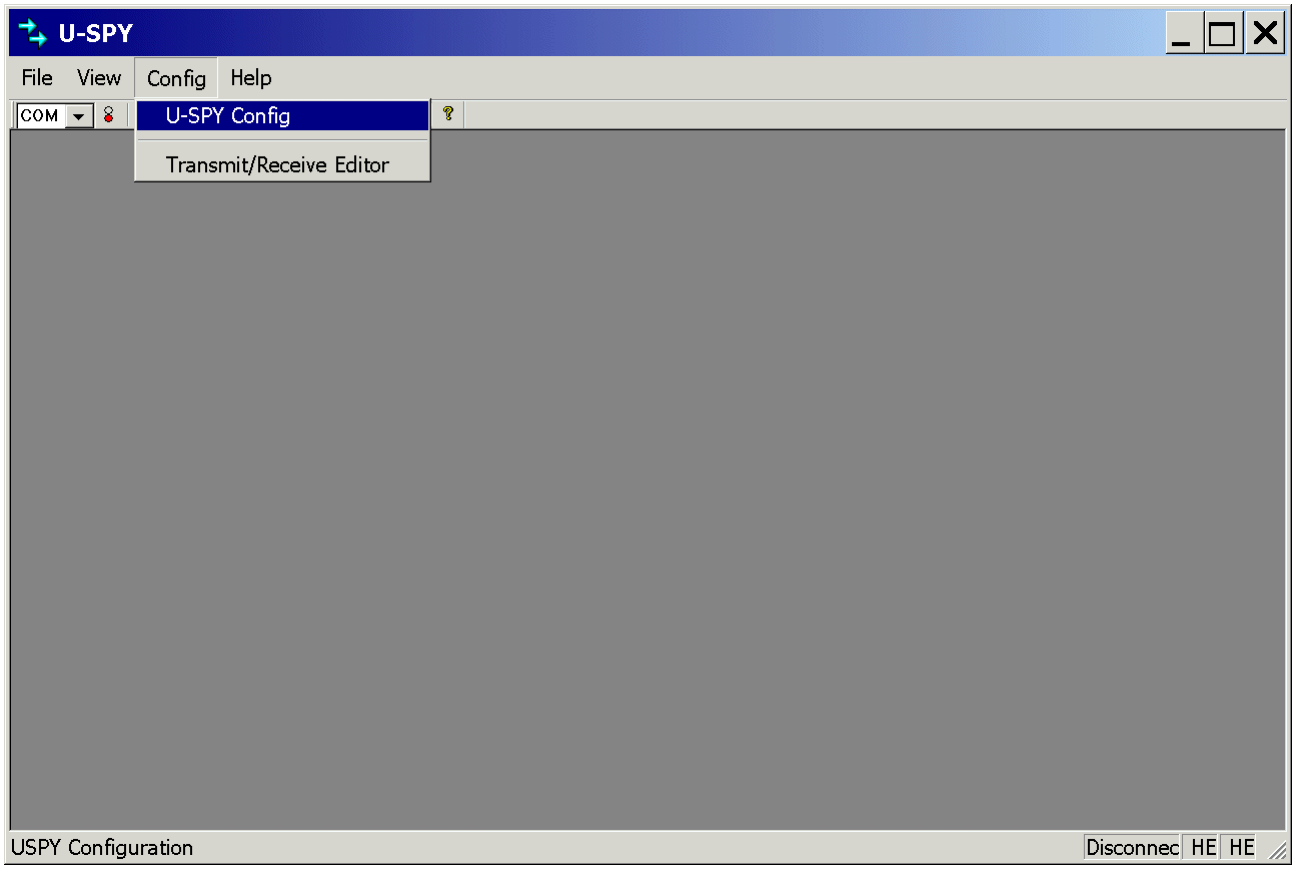
or: click 



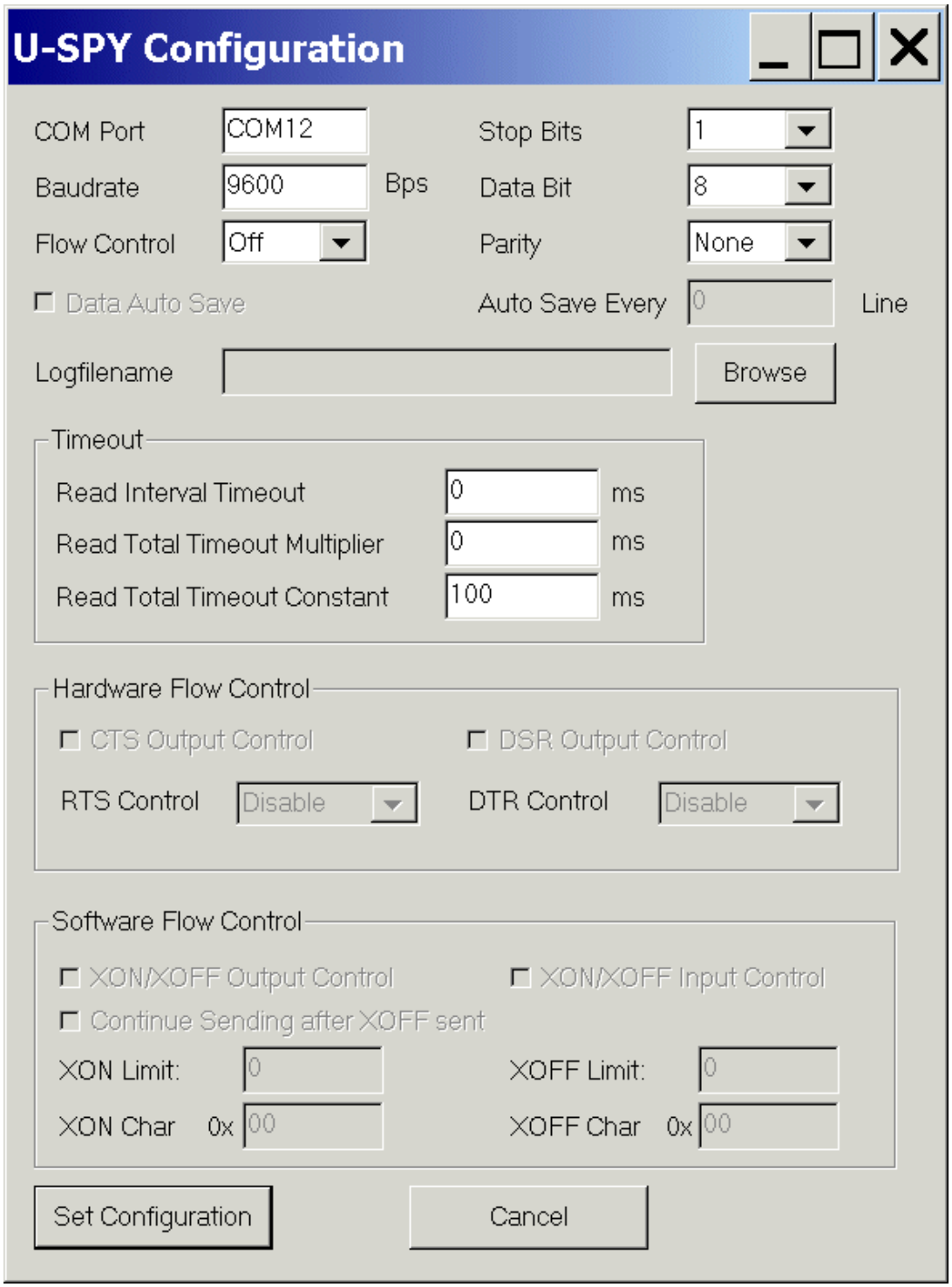
Now, start U-SPY: click 



Config – U-SPY Config

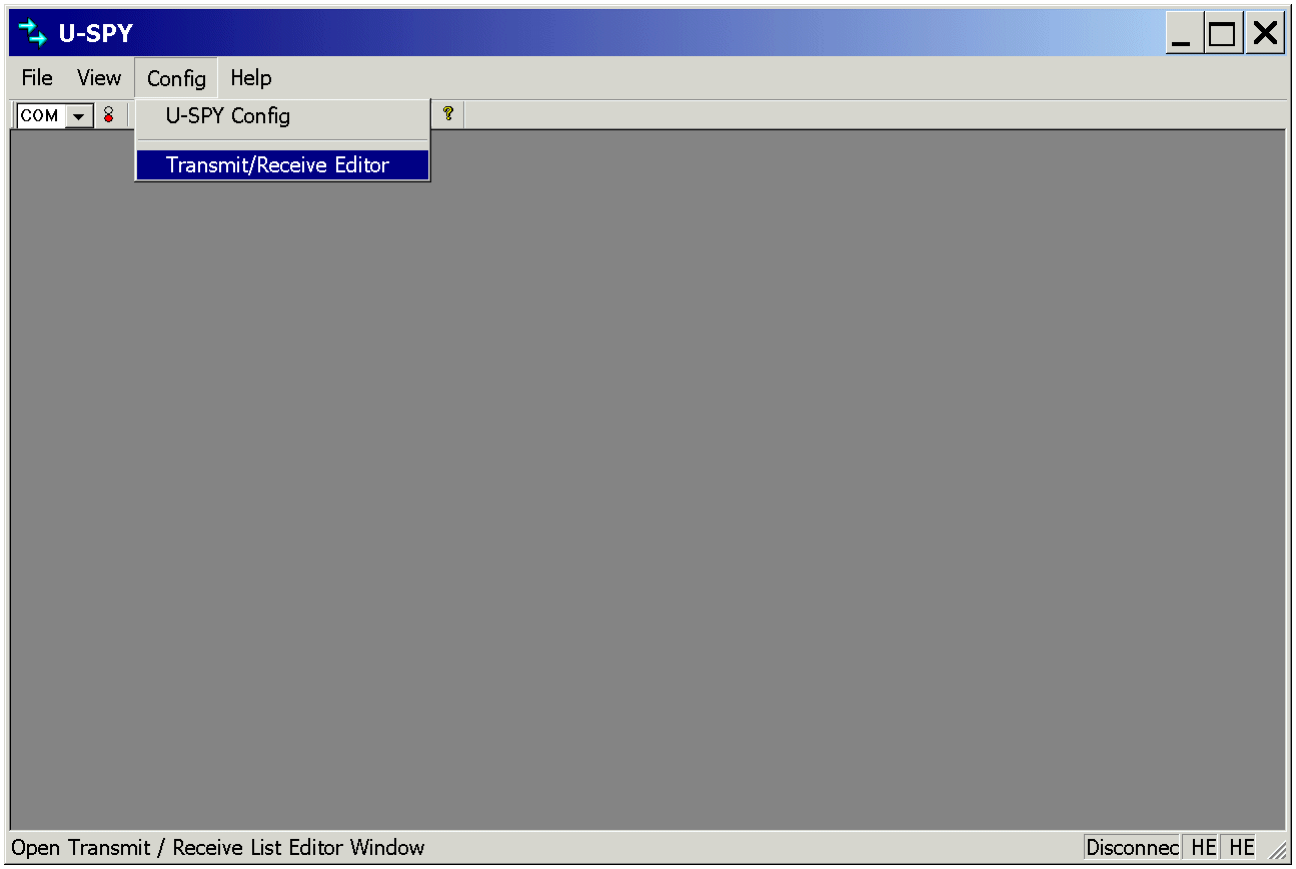


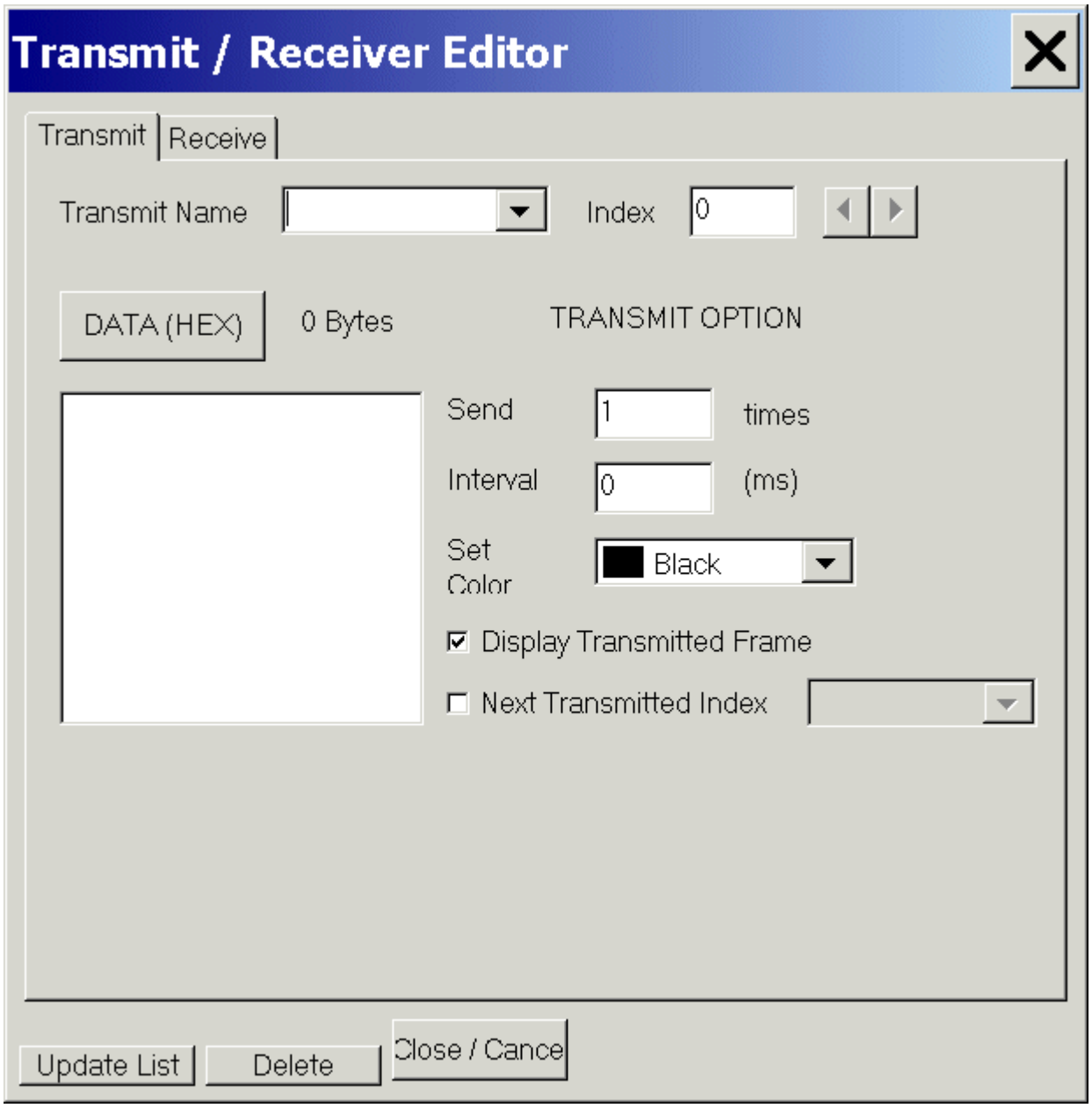
COM Port check/insert COM12
Baudrate check/insert 9600
Flow Control check/insert Off
Stop Bits check/insert 1
Data Bit check/insert 8
Parity check/insert None



Click Set Configuration

Config – Transmit/Receive Editor



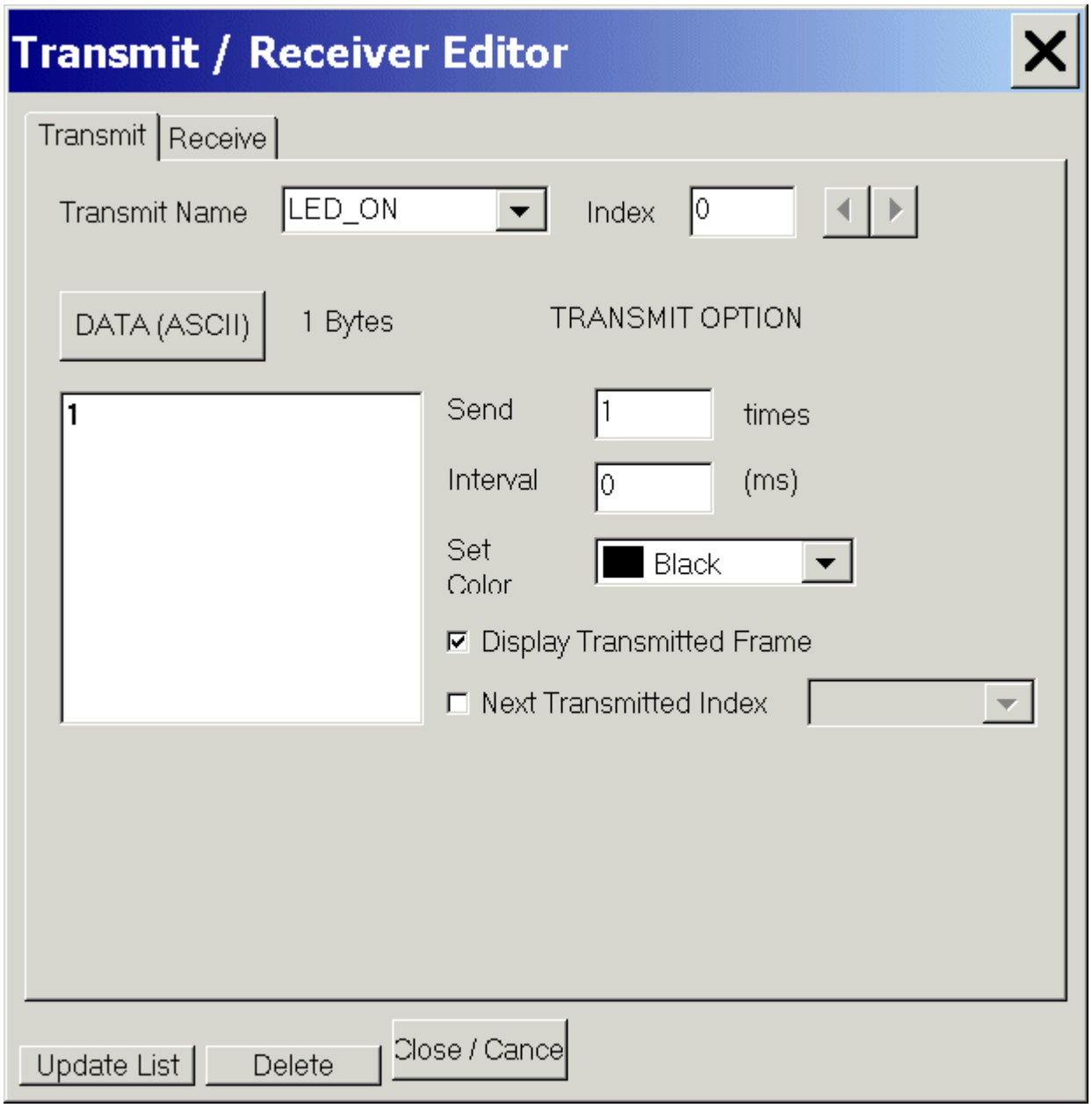

Transmit / Receiver Editor
Transmit
Receive
Transmit Name
Index
0
DATA (HEX)
0 Bytes
TRANSMIT OPTION
Send
1
times
Interval
0
(ms)
Set Color
Black
Display Transmitted Frame
Next Transmitted Index
Update List
Delete
Close / Cance

With the Transmit/Receive Editor configure the Transmit Window so that it appears like the Transmit windows below:

Transmit: Transmit Name: insert LED_ON

Click DATA (HEX) to change to: DATA (ASCII)

Insert 1:

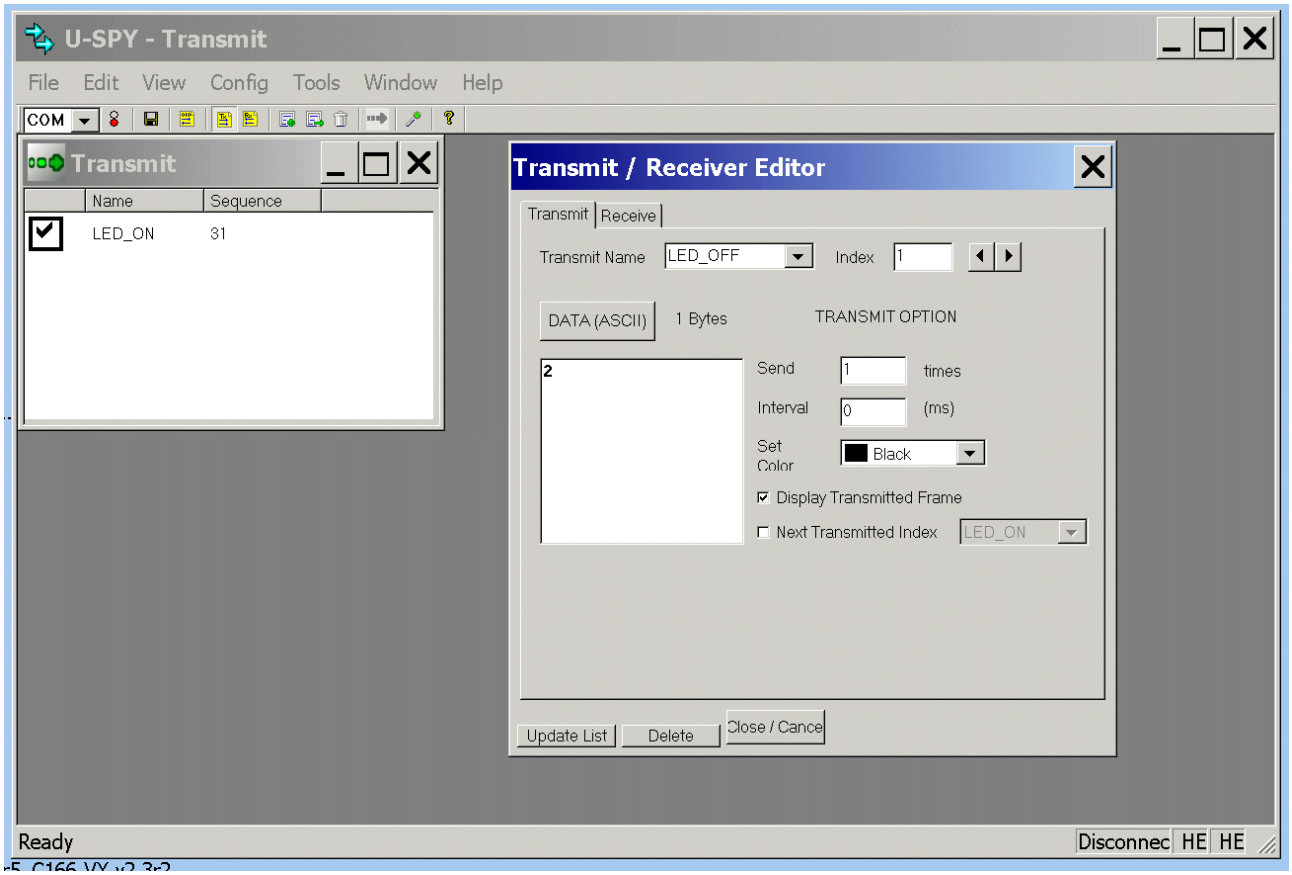


Click Update List

Transmit: Transmit Name: insert LED_OFF

Click DATA (HEX) to change to: DATA (ASCII)

Insert 2:

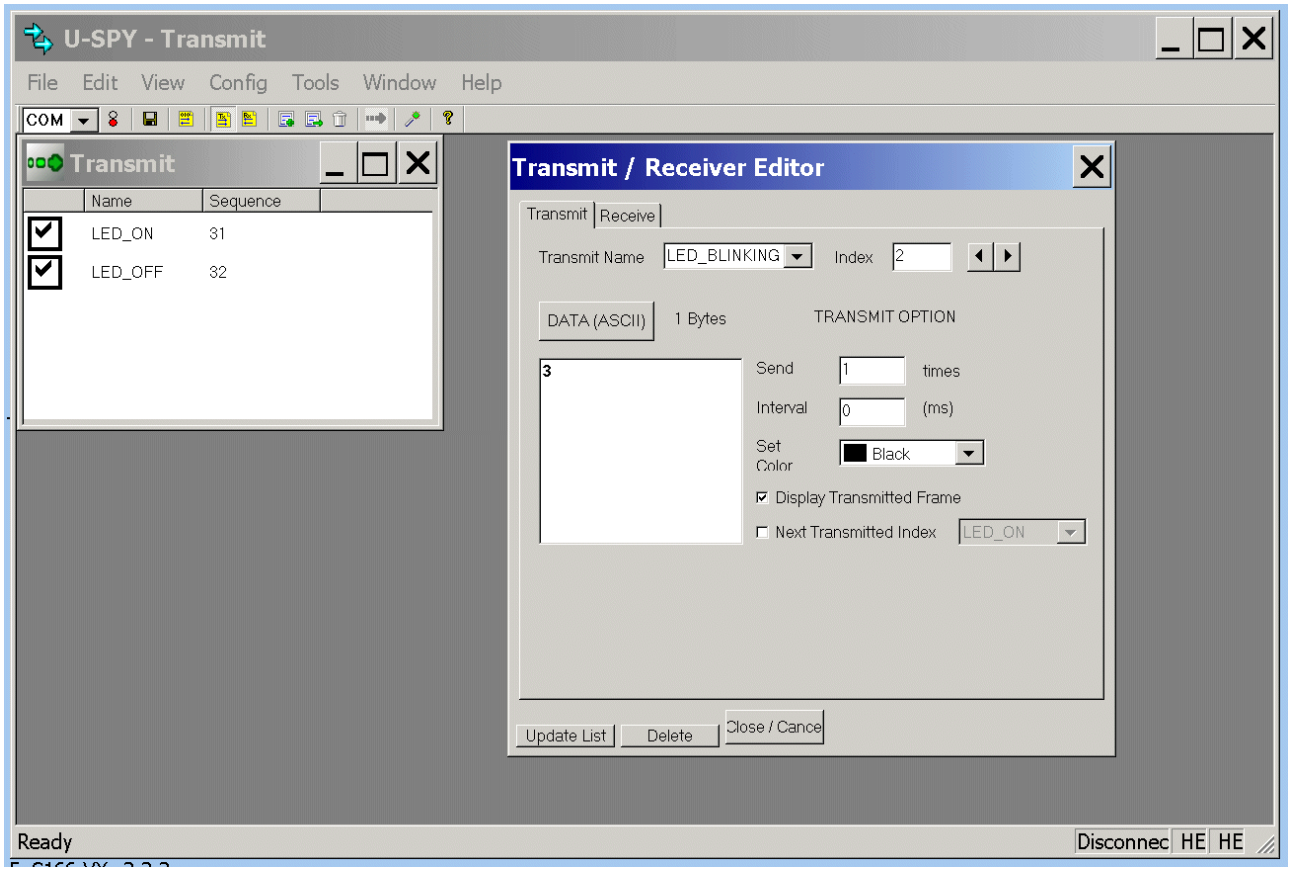


Click Update List

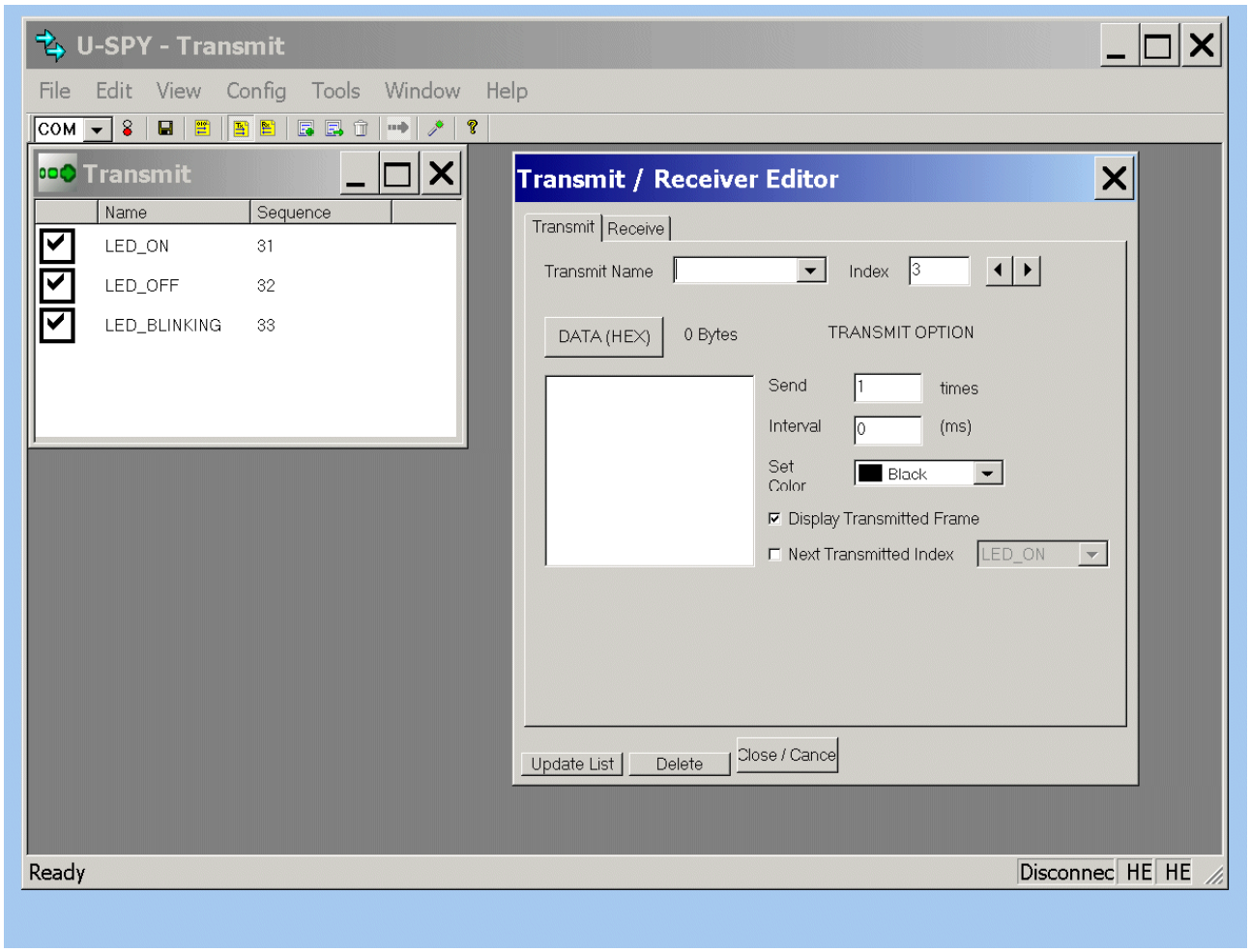
Transmit: Transmit Name: insert LED_BLINKING

Click DATA (HEX) to change to: DATA (ASCII)

Insert 3:

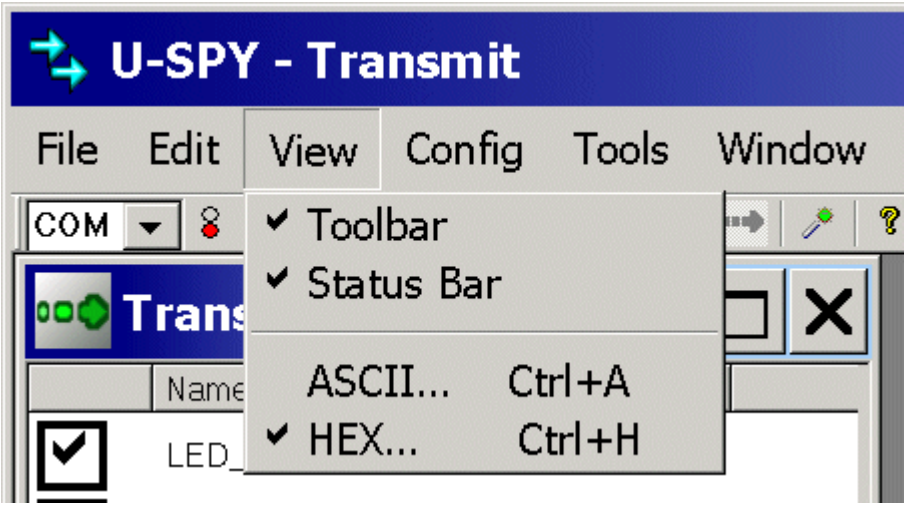


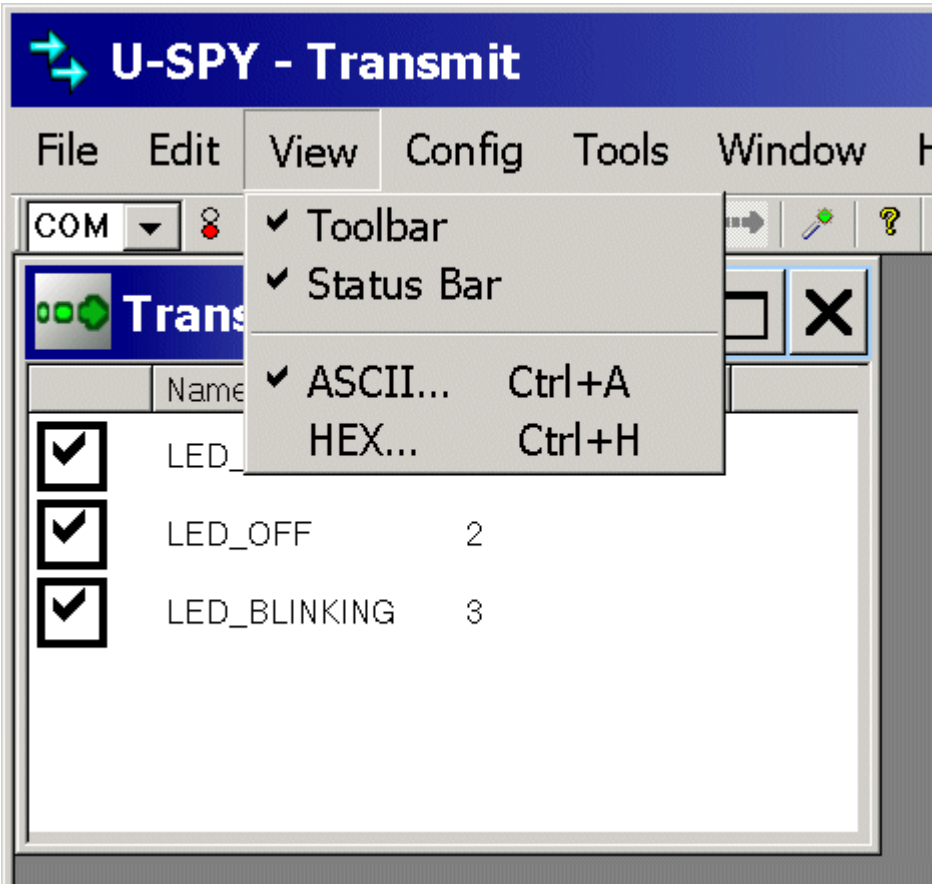
Click Update List


U-SPY - Transmit
File Edit View Config Tools Window Help
COM
Transmit
Name Sequence
LED_ON 31
LED_OFF 32
LED_BLINKING 33
Transmit / Receiver Editor
Transmit Receive
Transmit Name
Index 3
DATA (HEX)
0 Bytes
TRANSMIT OPTION
Send 1 times
Interval 0 (ms)
Set Color Black
Display Transmitted Frame
Next Transmitted Index LED_ON
Update List
Delete
Close / Cance
Ready
Disconnec HE HE

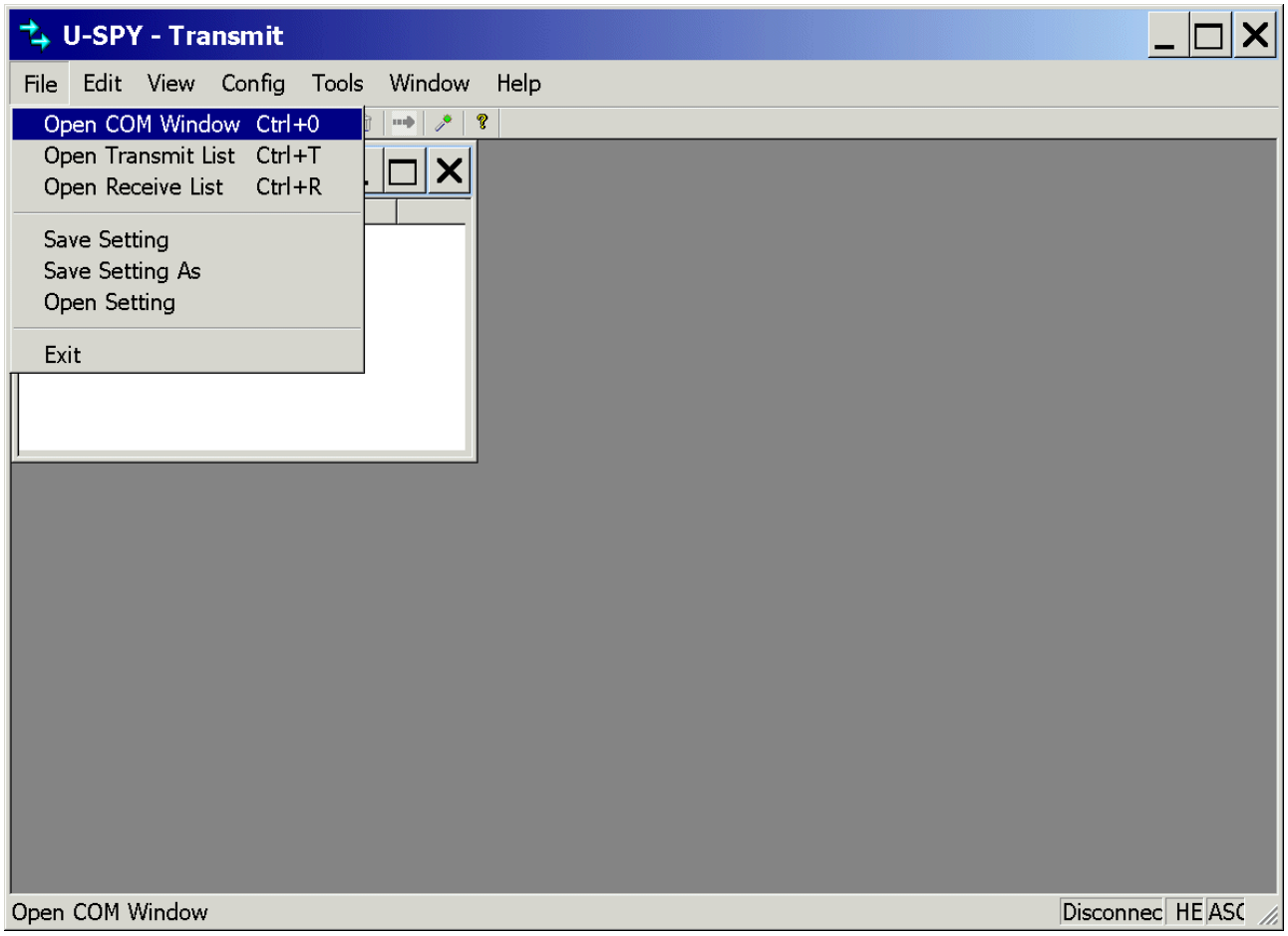
Click 
Close / Cance

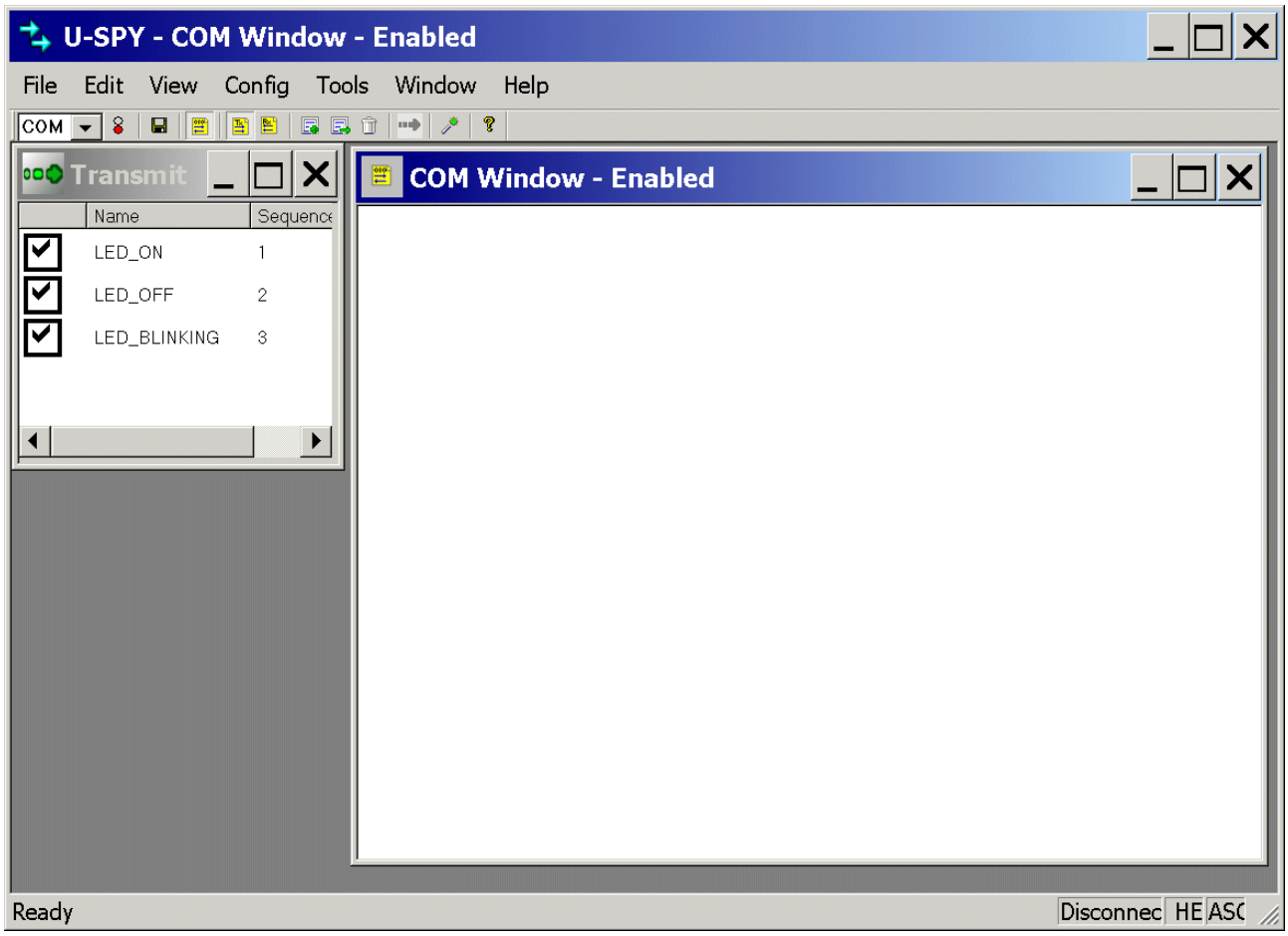
View – select ASCII…



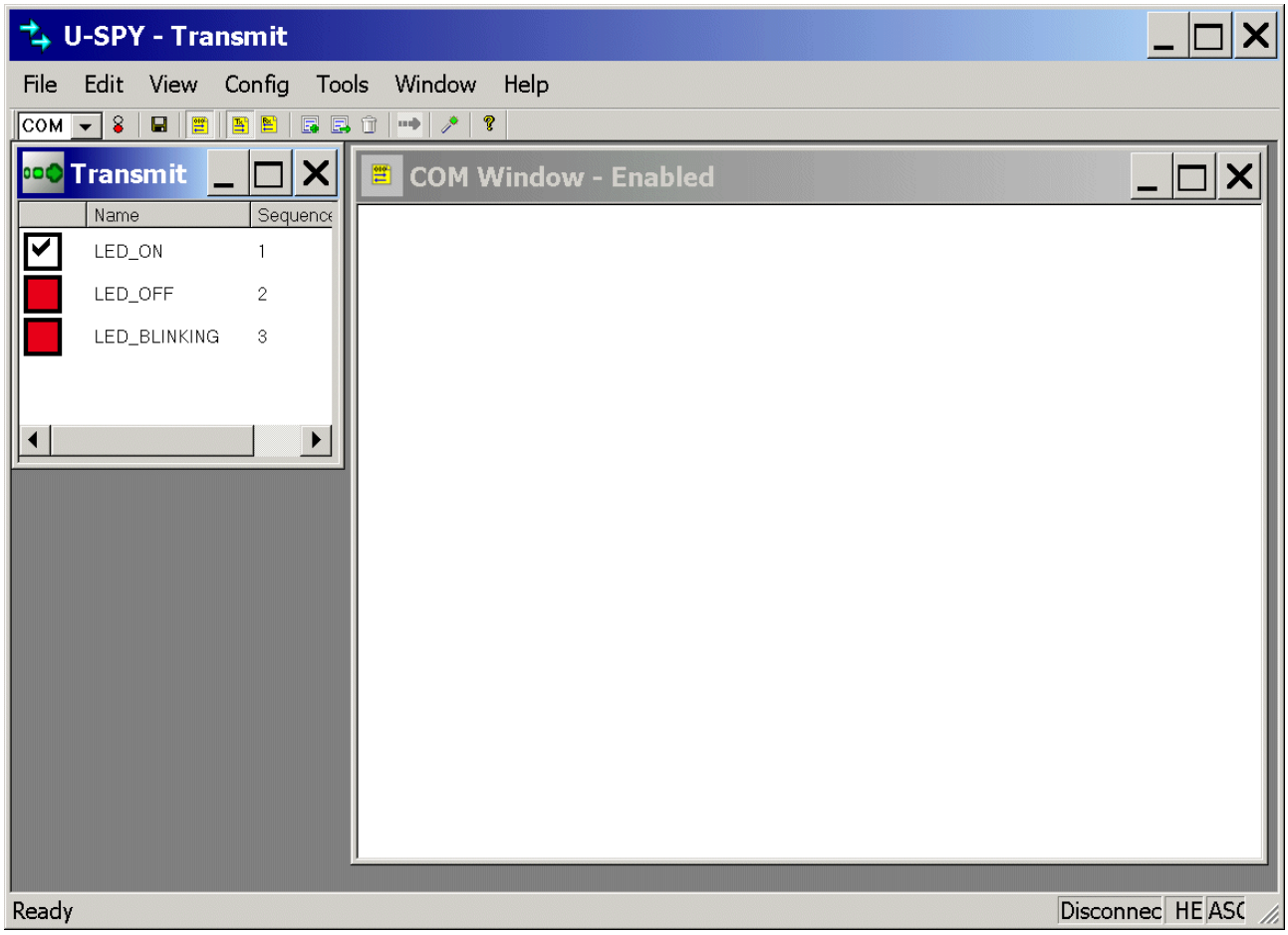


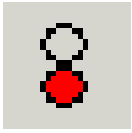
File – Open COM Window

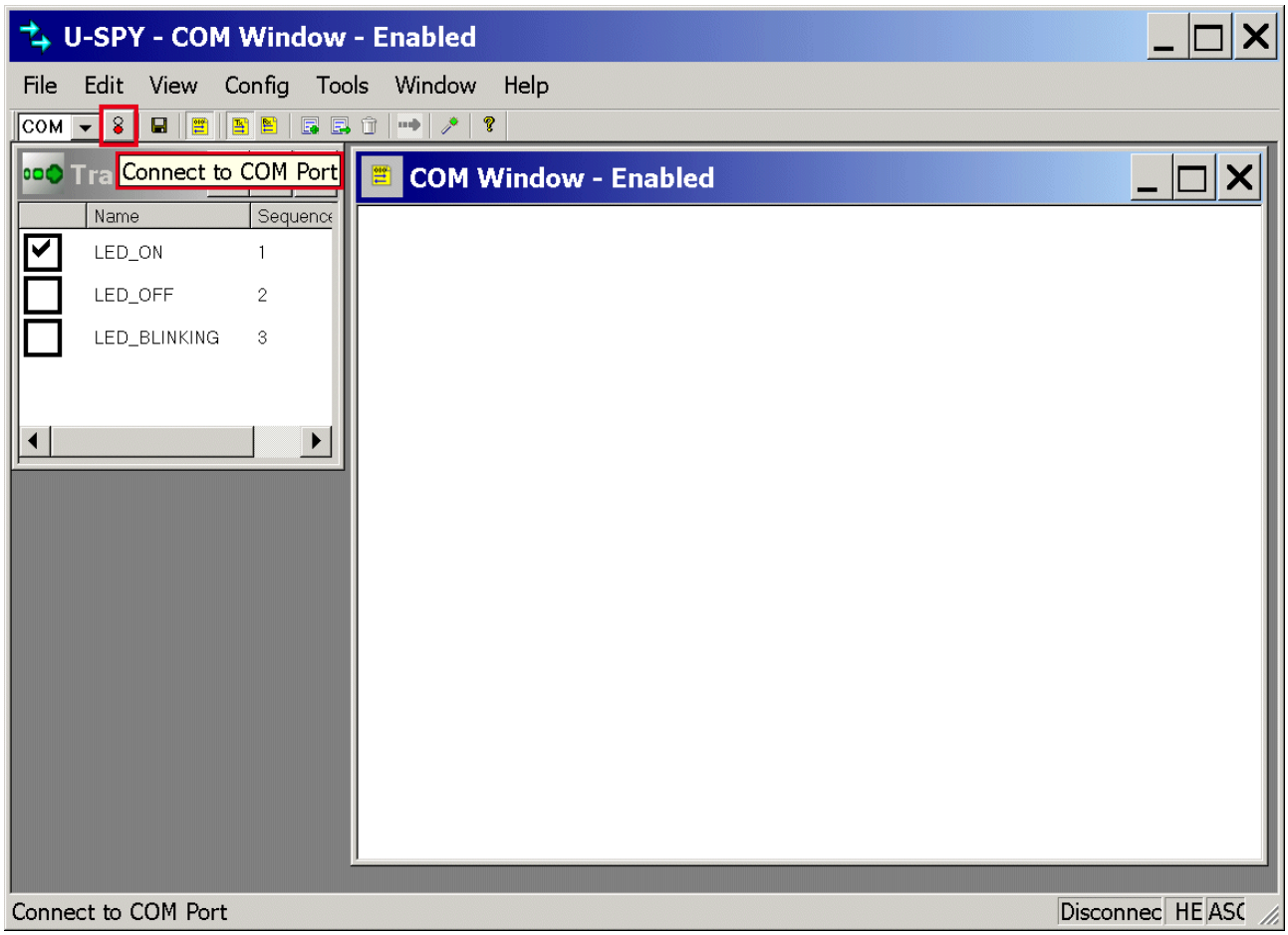



U-SPY - COM Window - Enabled
File
Edit
View
Config
Tools
Window
Help
COM
Transmit
Name
Sequence
LED_ON
1
LED_OFF
2
LED_BLINKING
3
COM Window - Enabled
Ready
Disconnec
HE
ASC

Click ☑ LED_OFF to untick/unselect ☐ LED_OFF
Click ☑ LED_BLINKING to untick/unselect ☐ LED_BLINKING



Click  :

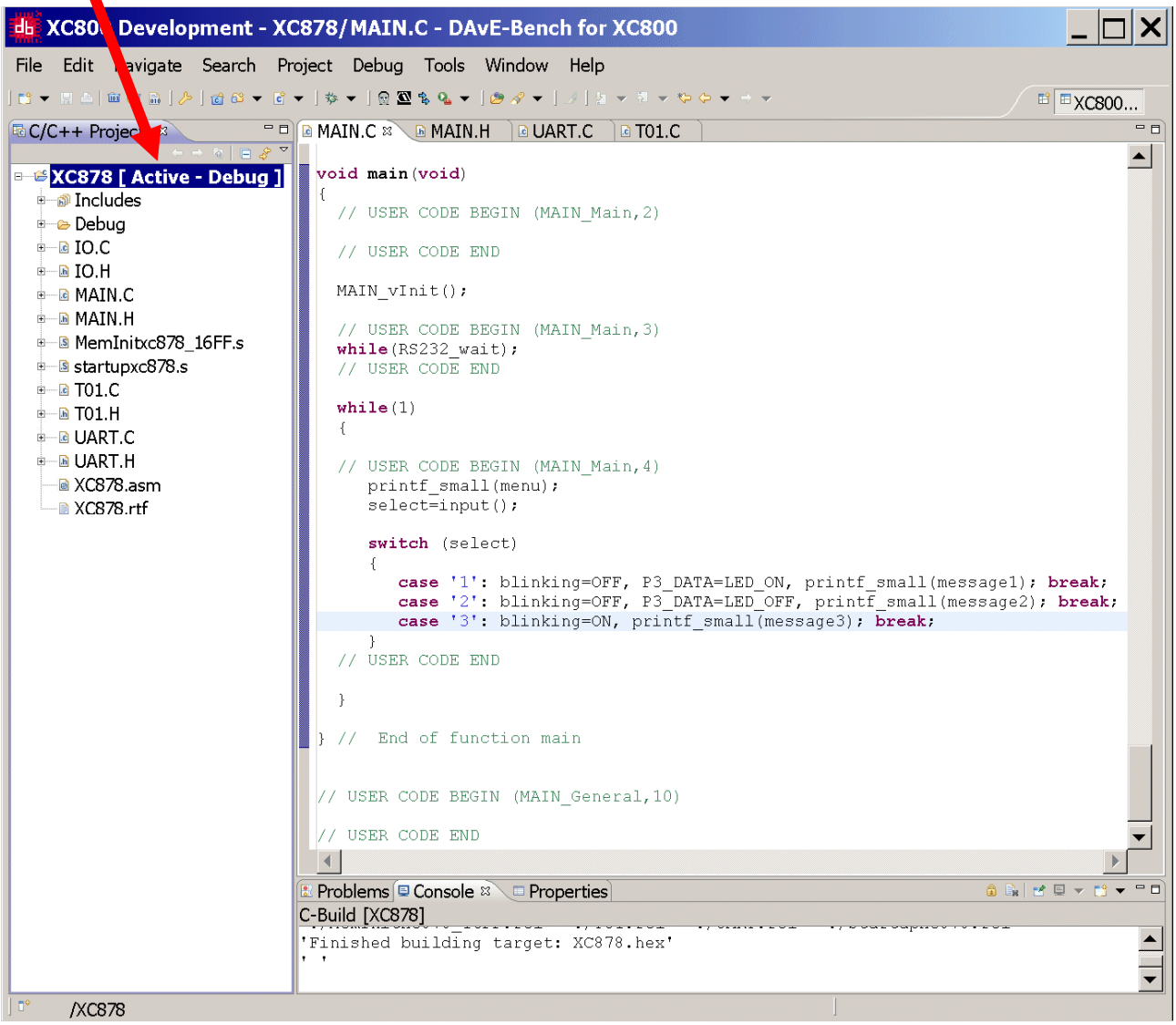


Note:

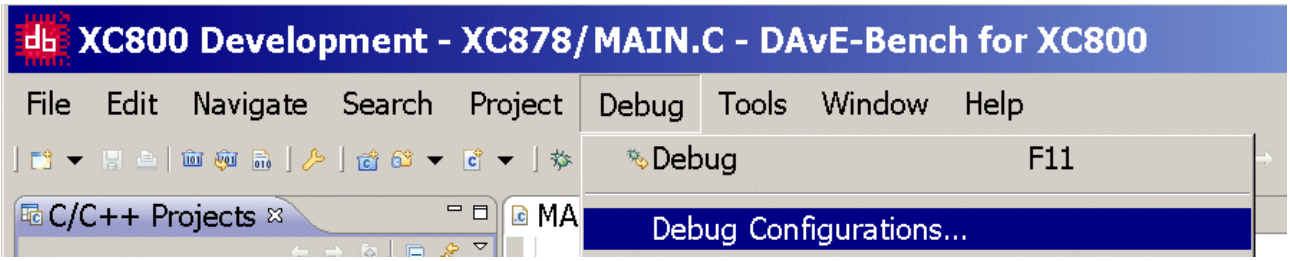
 : U-SPY is now ready for serial communication! 

Go back to DAvE Bench and configure the debugger:

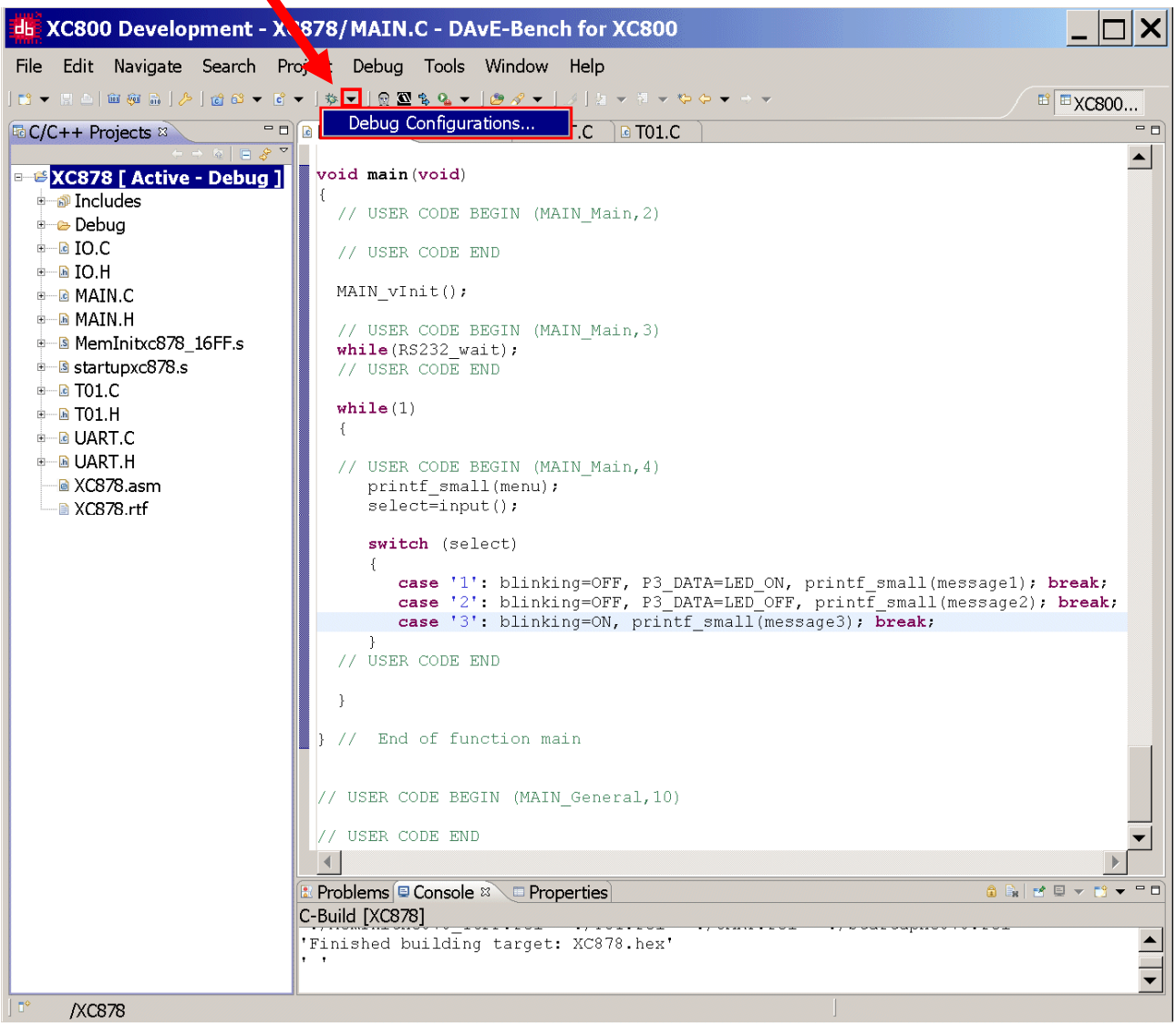
Click on **XC888 [Active - Debug]**:



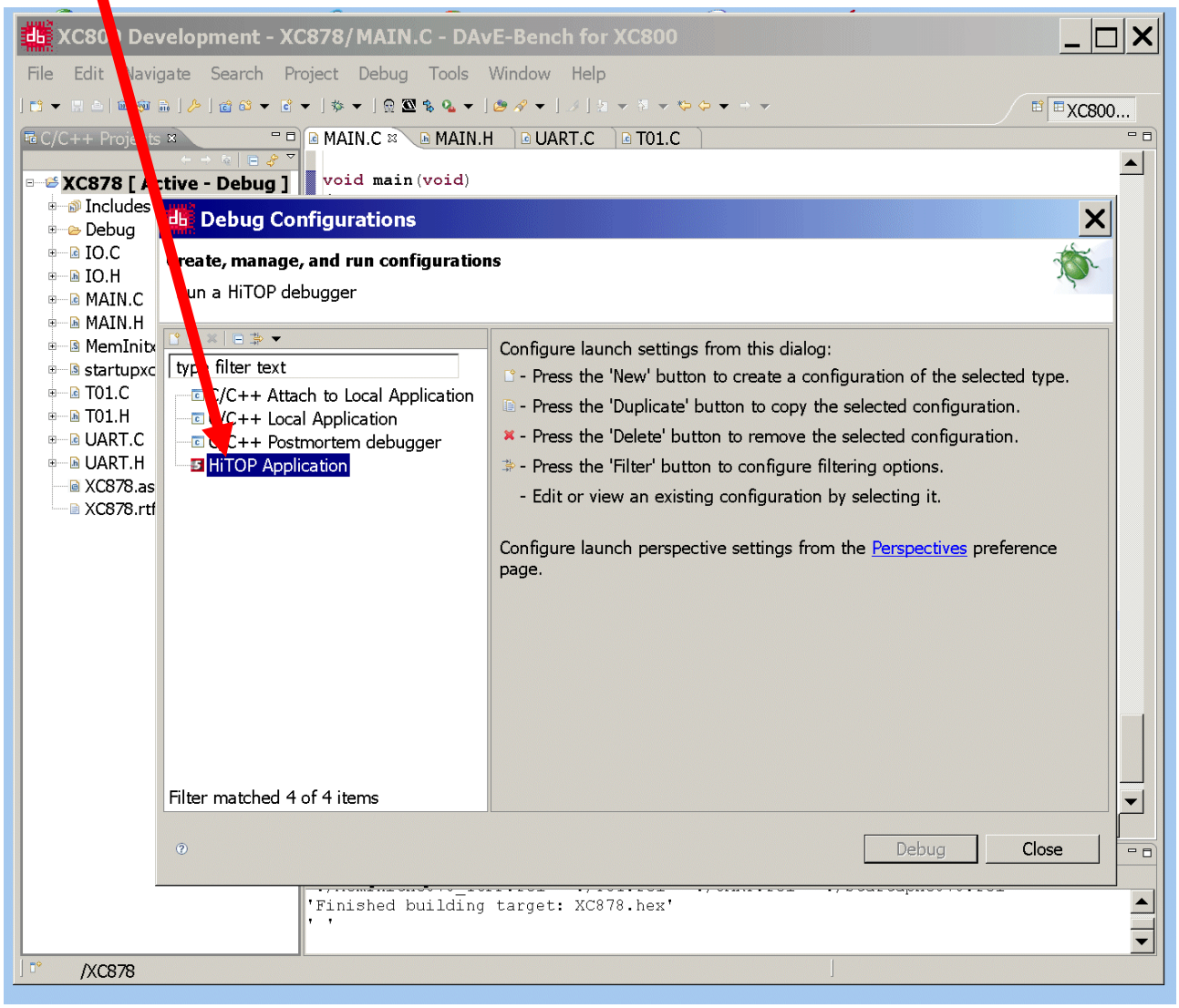
Debug – Debug Configurations…



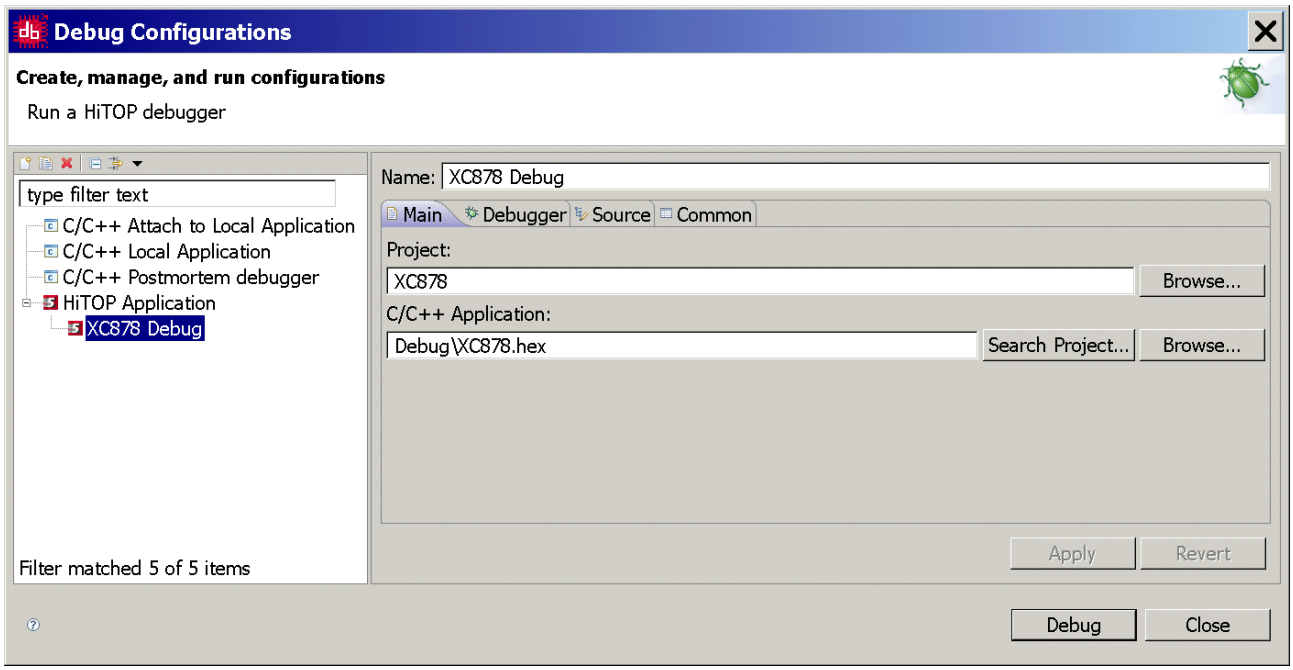
or click Debug Configurations…

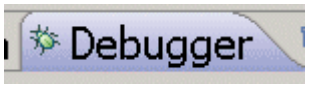


Double click: HiTOP Application HiTOP Application:

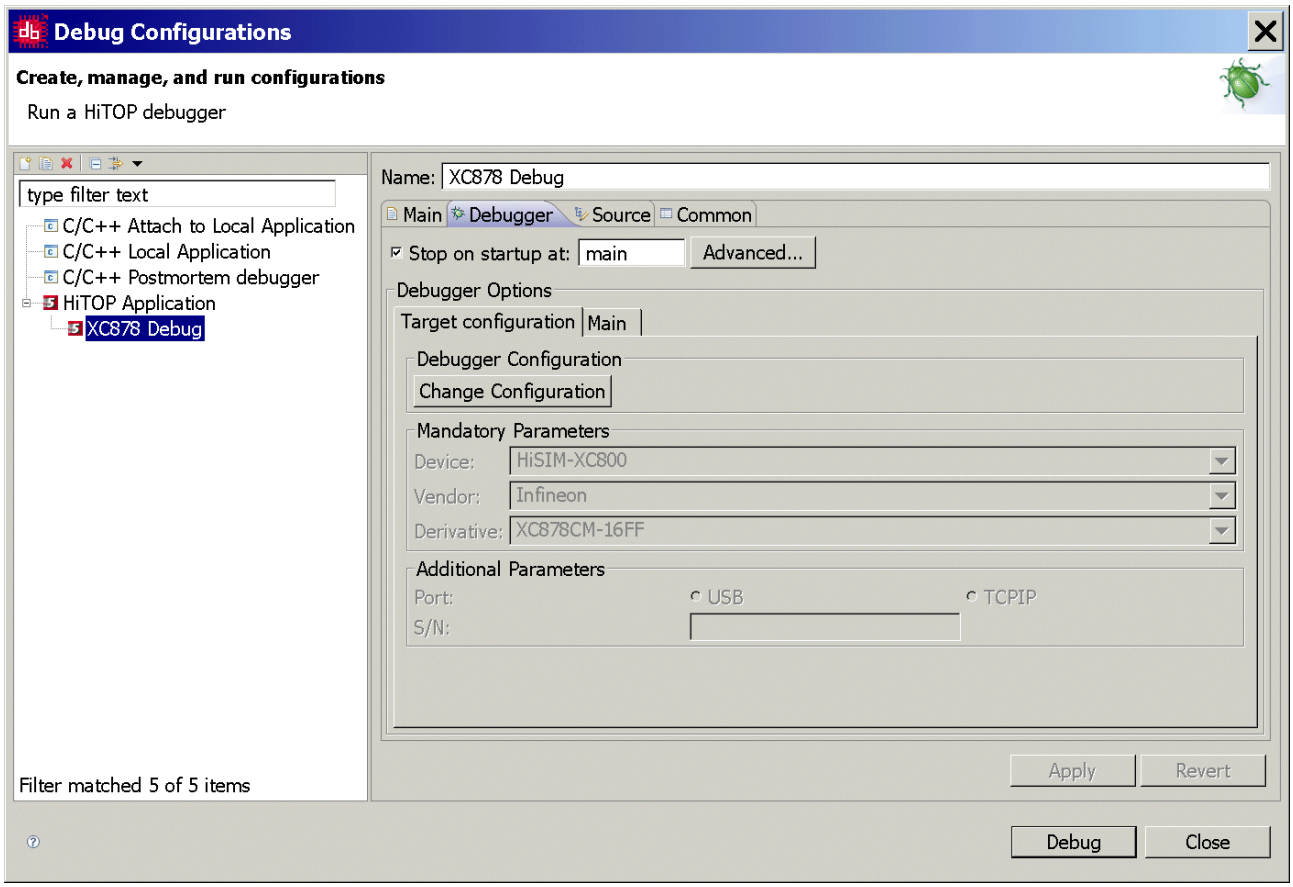



Main
Main:

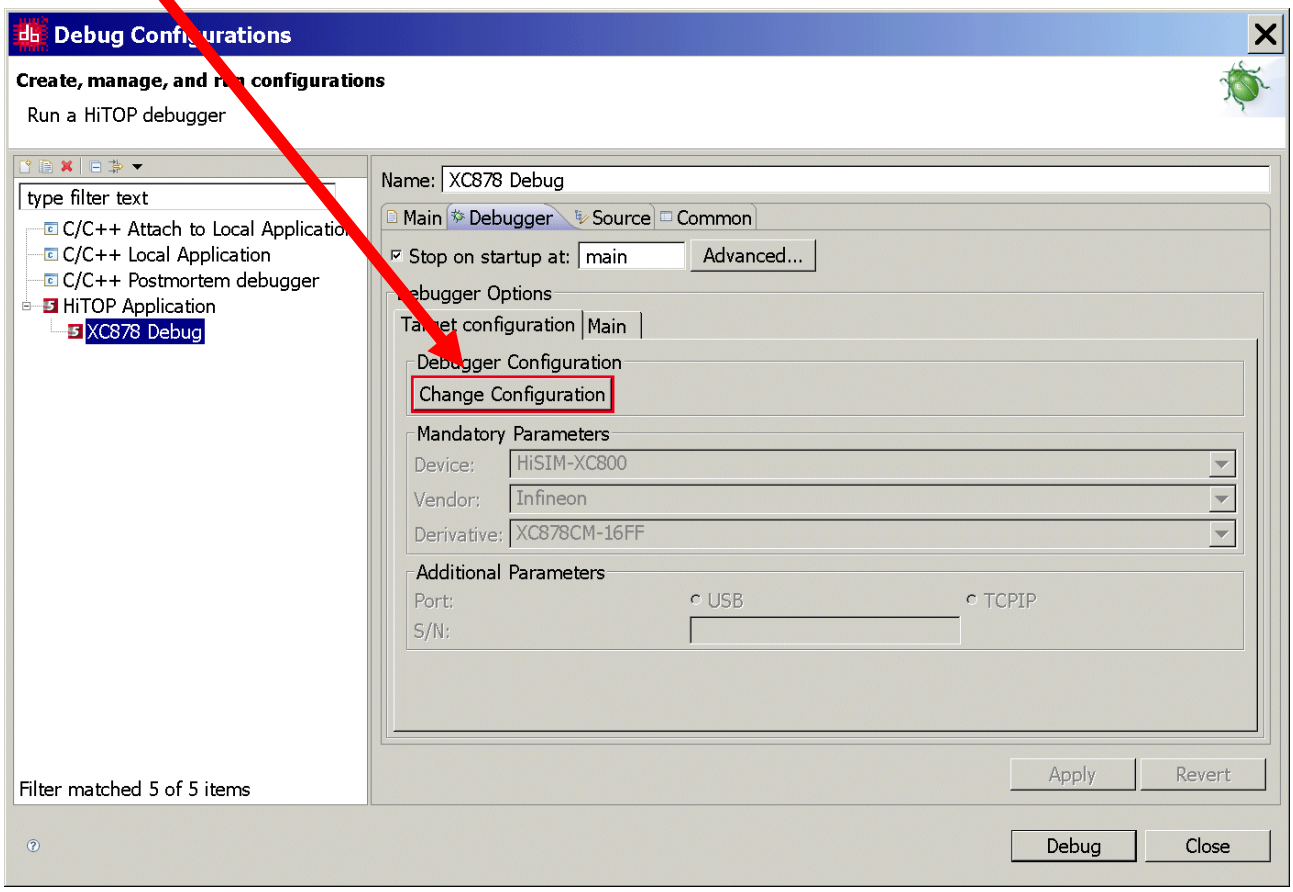

Debug Configurations
Create, manage, and run configurations
Run a HiTOP debugger
type filter text
C/C++ Attach to Local Application
C/C++ Local Application
C/C++ Postmortem debugger
HiTOP Application
XC878 Debug
Filter matched 5 of 5 items
Name: XC878 Debug
Main
Debugger
Source
Common
Project:
XC878
Browse...
C/C++ Application:
Debug\XC878.hex
Search Project...
Browse...
Apply
Revert
Debug
Close



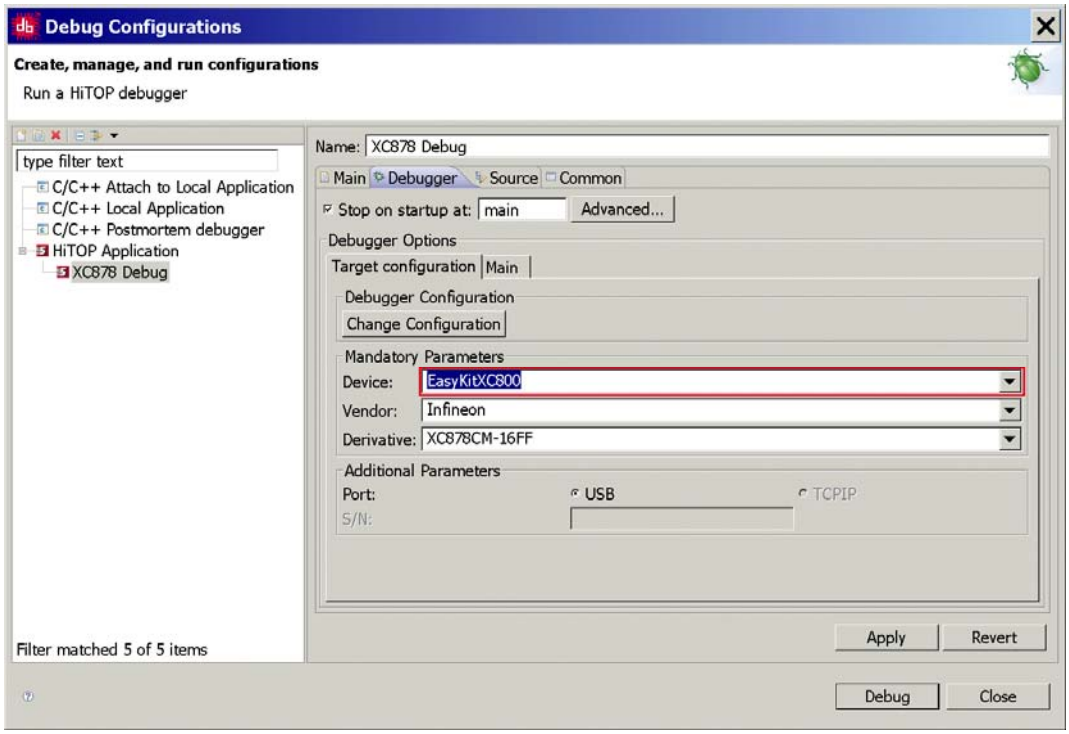
Debugger, Target configuration:

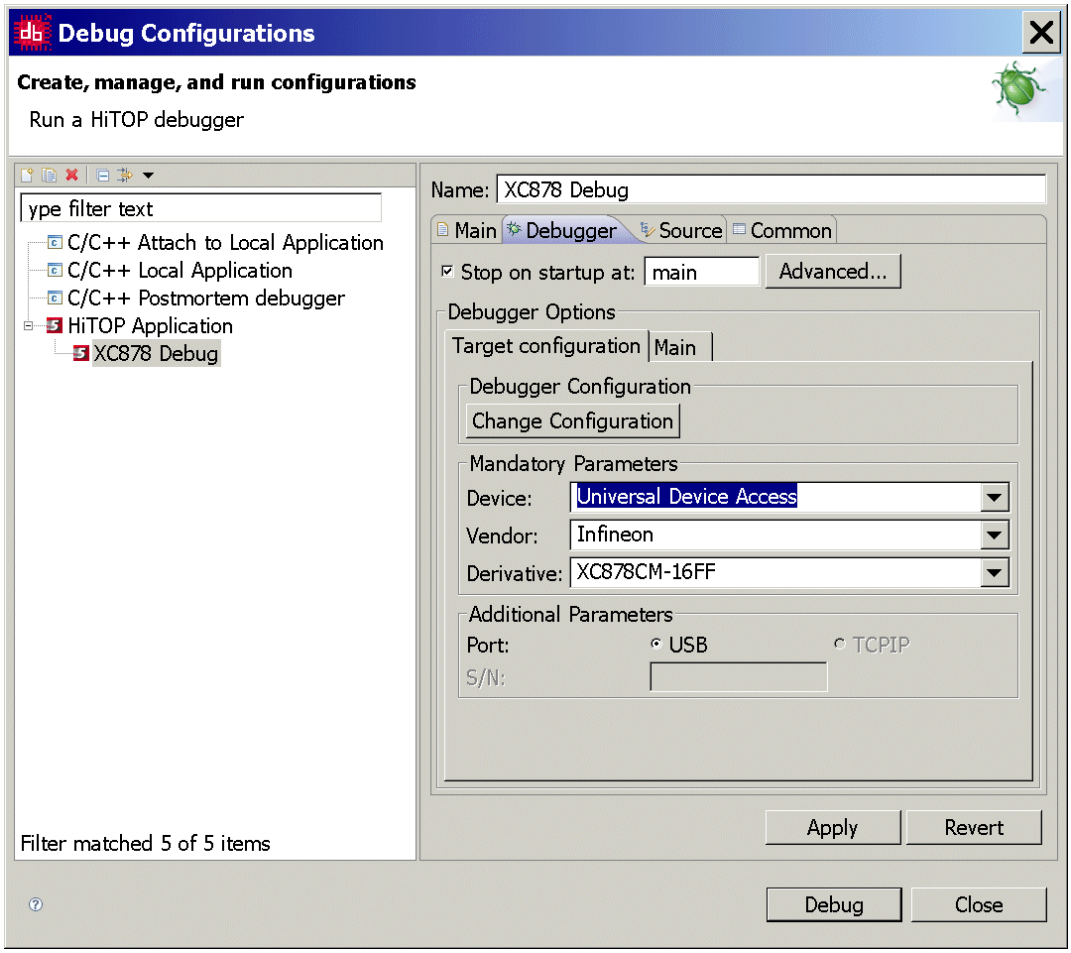


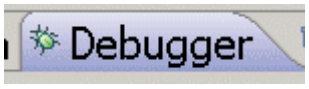
Click: Change Configuration Change Configuration:



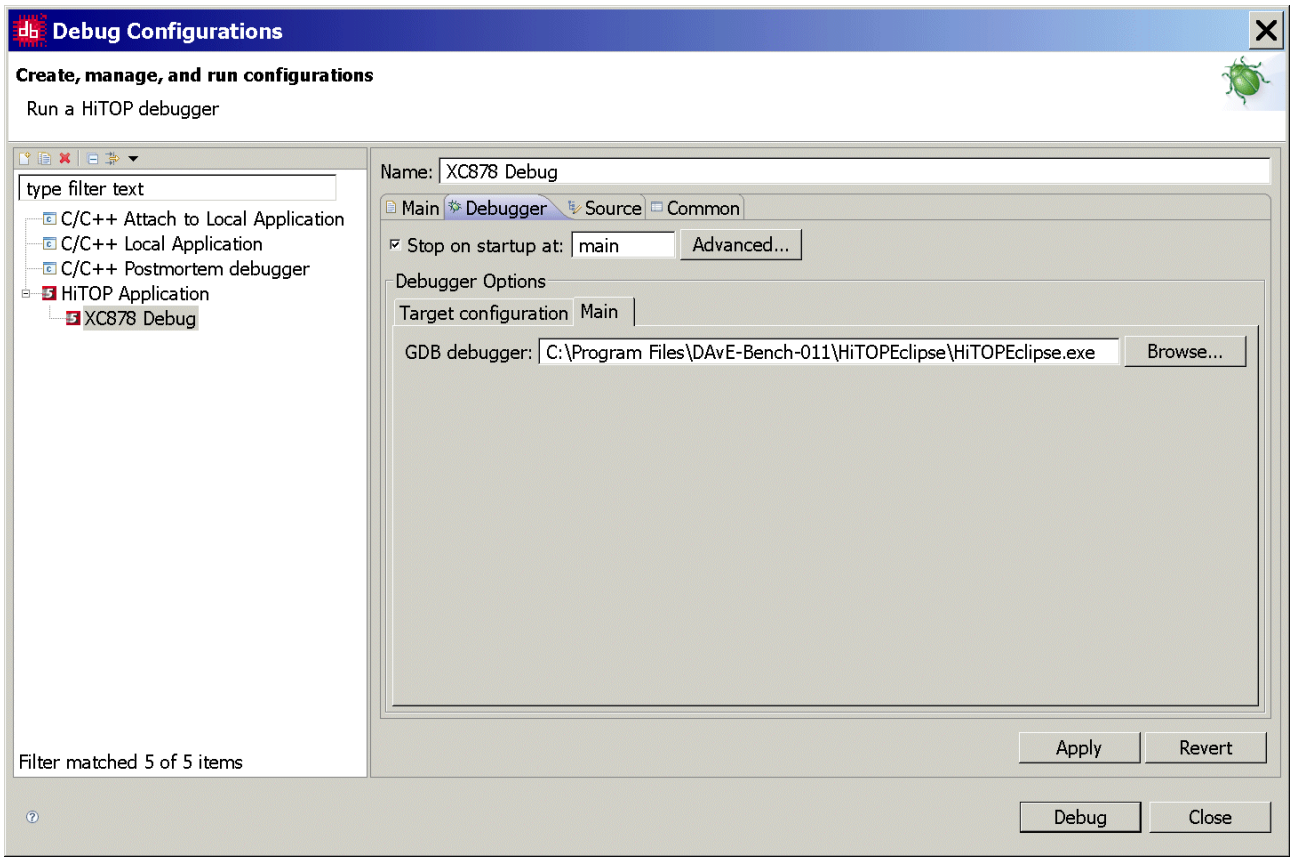
Mandatory Parameters: Device: select EasyKitXC800 or Universal Device Acess



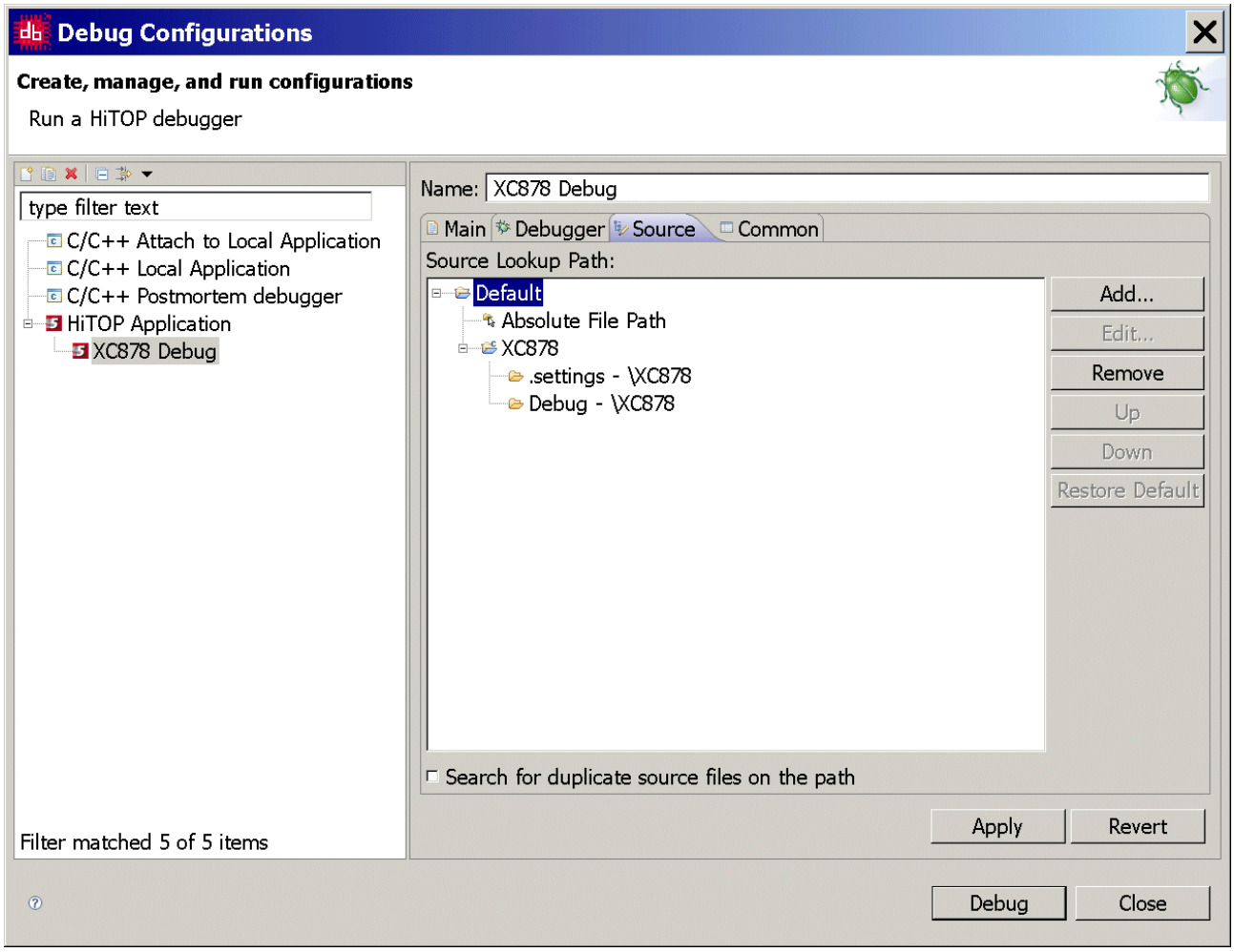




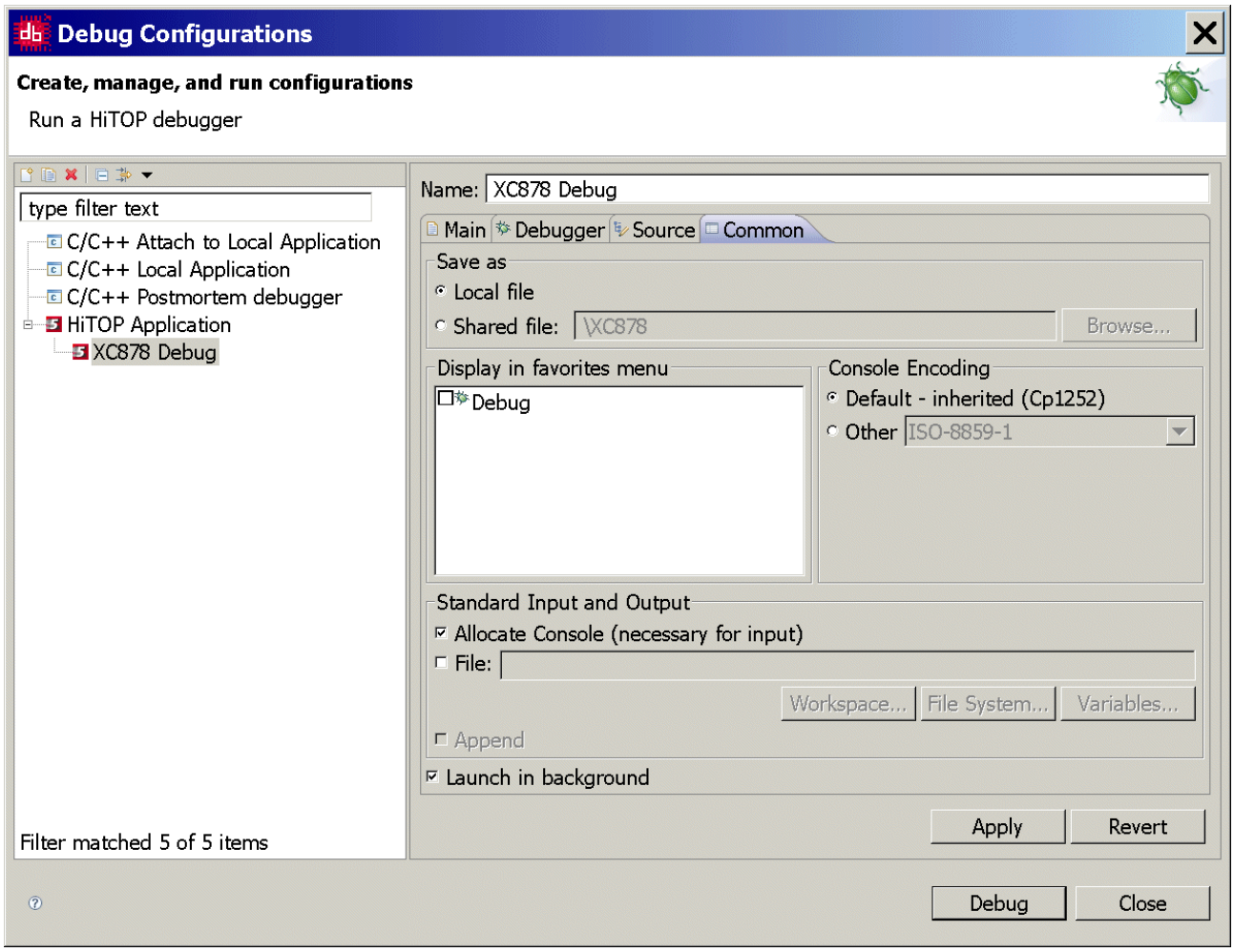
Debugger, Main: (do nothing)



Source: (do nothing)



Common: (do nothing)

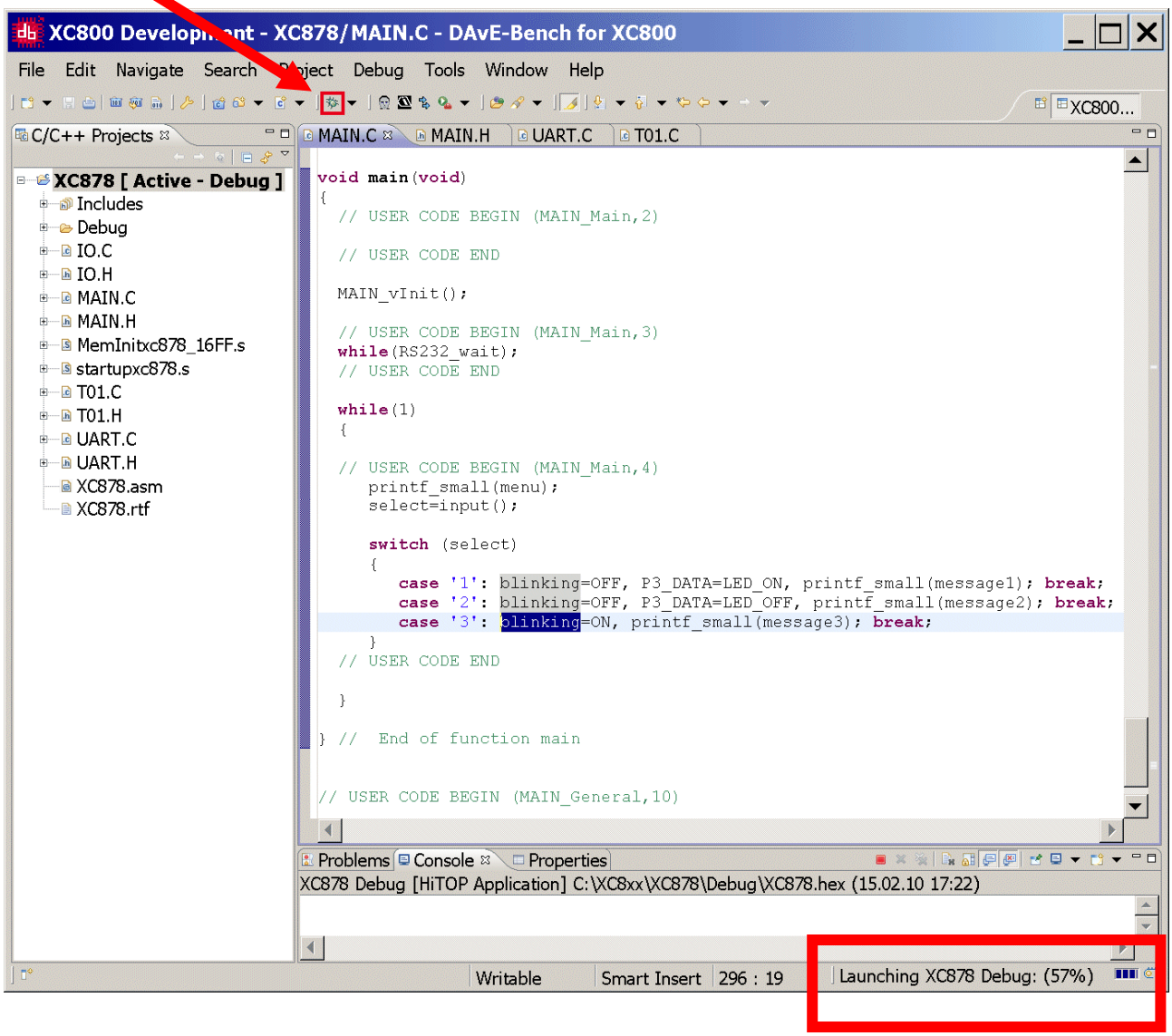


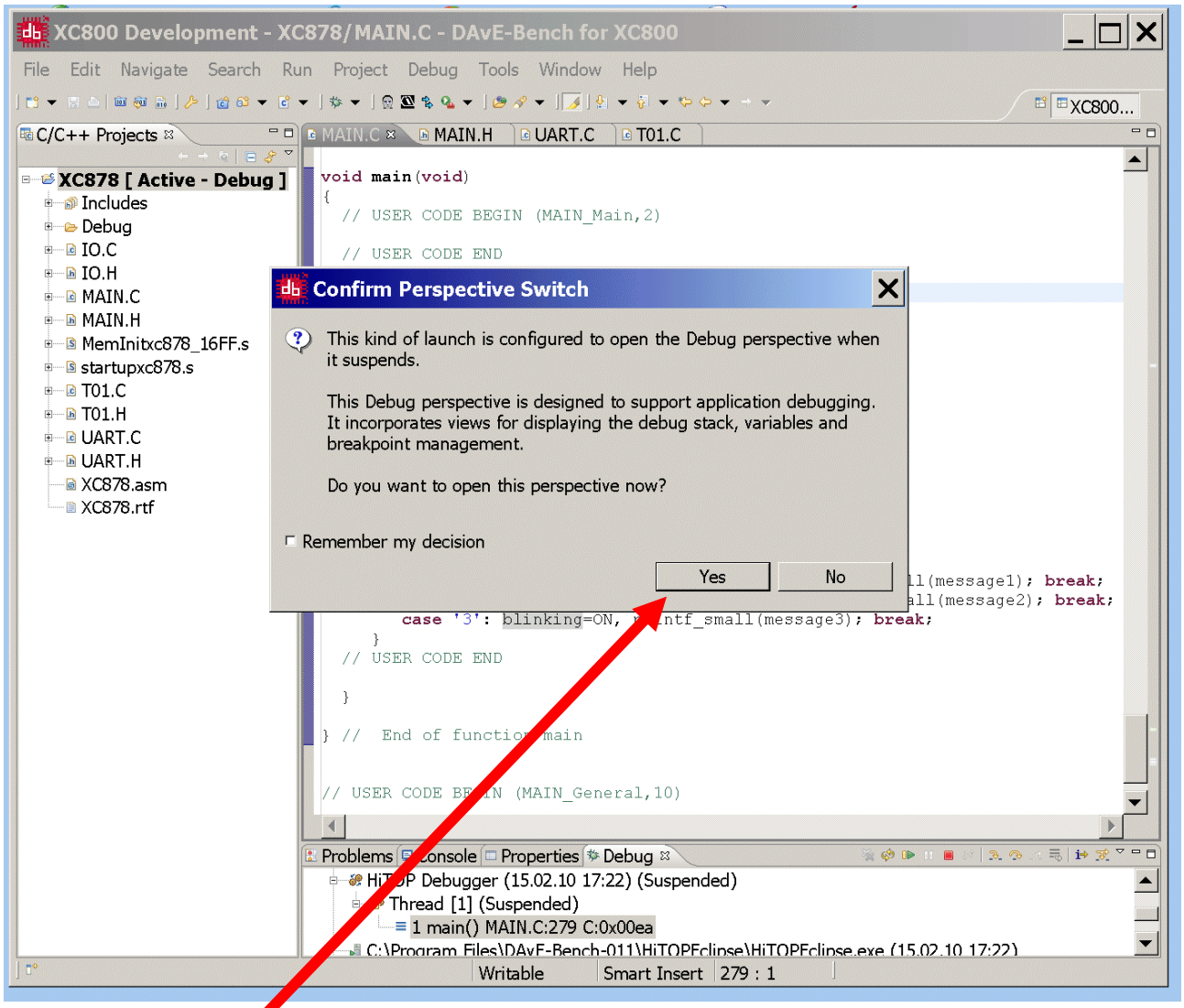
Apply

Close

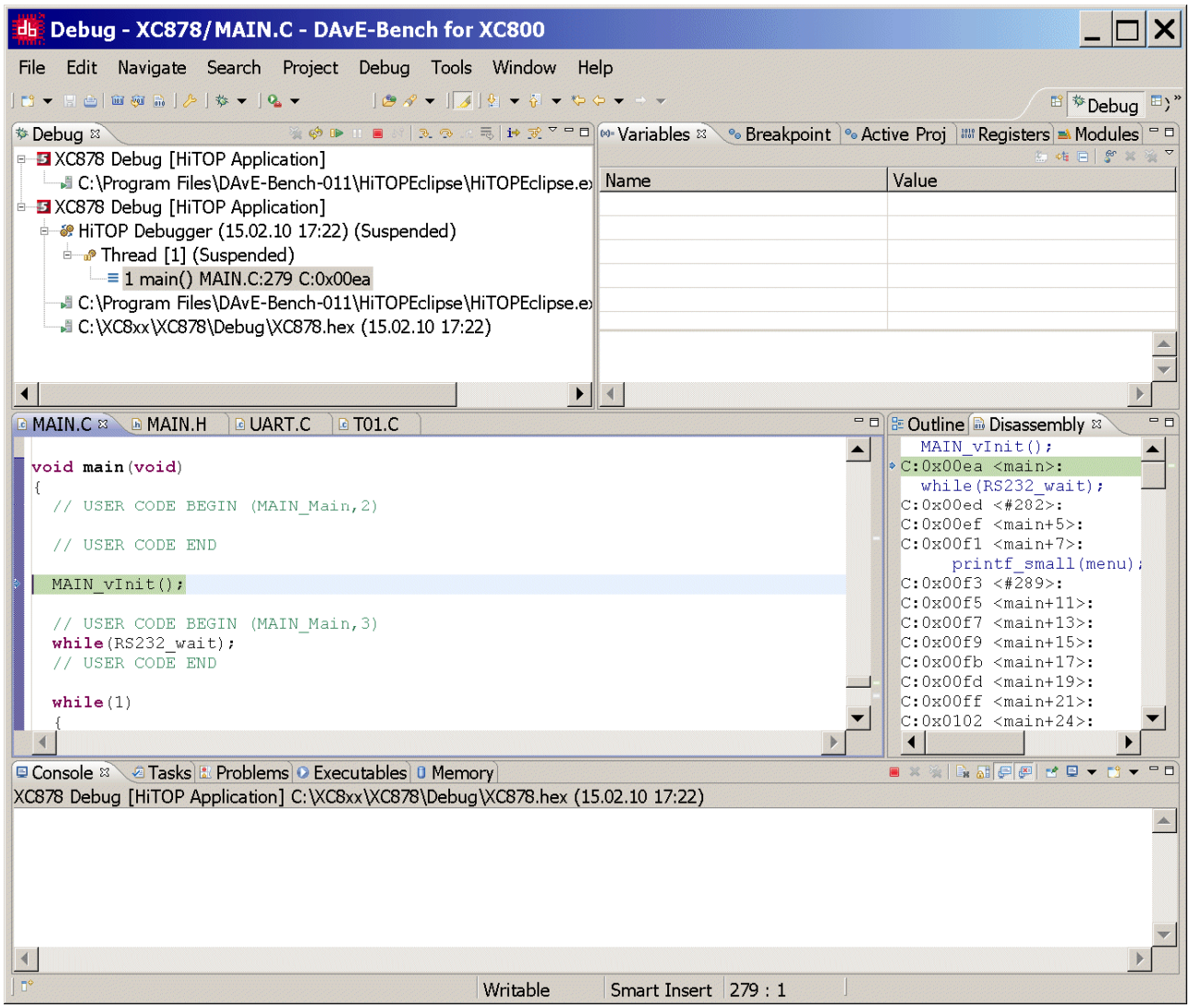
Start/Launch the debugger:

Click  :

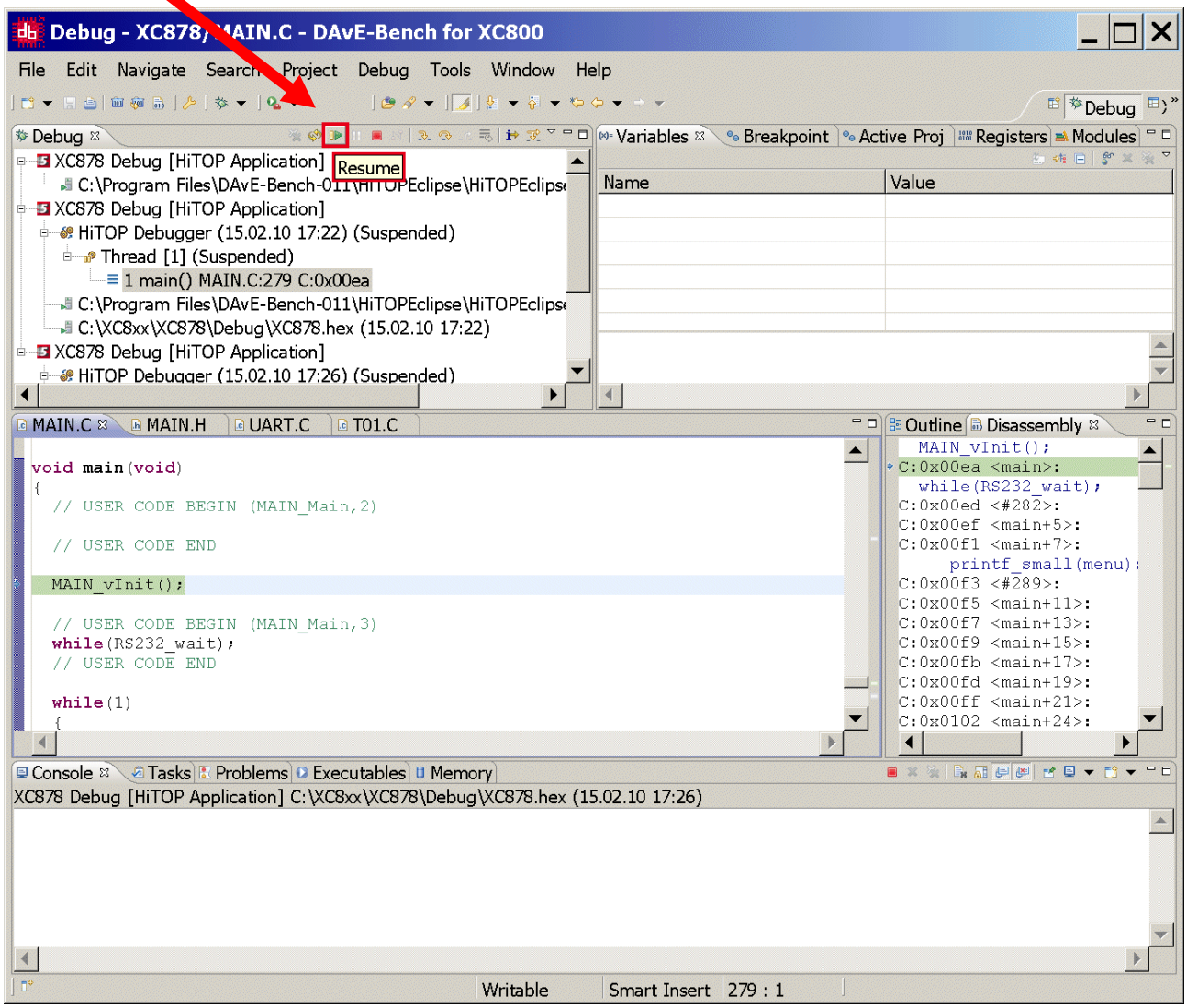



XC800 Development - XC878/MAIN.C - DAvE-Bench for XC800
File Edit Navigate Search Run Project Debug Tools Window Help
C/C++ Projects
XC878 [Active - Debug]
Includes
Debug
IO.C
IO.H
MAIN.C
MAIN.H
MemInitxc878_16FF.s
startupxc878.s
T01.C
T01.H
UART.C
UART.H
XC878.asm
XC878.rtf
MAIN.C
MAIN.H
UART.C
T01.C
void main(void)
{
// USER CODE BEGIN (MAIN_Main,2)
// USER CODE END
Confirm Perspective Switch
This kind of launch is configured to open the Debug perspective when it suspends.
This Debug perspective is designed to support application debugging. It incorporates views for displaying the debug stack, variables and breakpoint management.
Do you want to open this perspective now?
Remember my decision
Yes
No
// USER CODE END
// USER CODE BEGIN (MAIN_General,10)
Problems
Console
Properties
Debug
Thread [1] (Suspended)
1 main() MAIN.C:279 C:0x00ea
Writable
Smart Insert
279 : 1

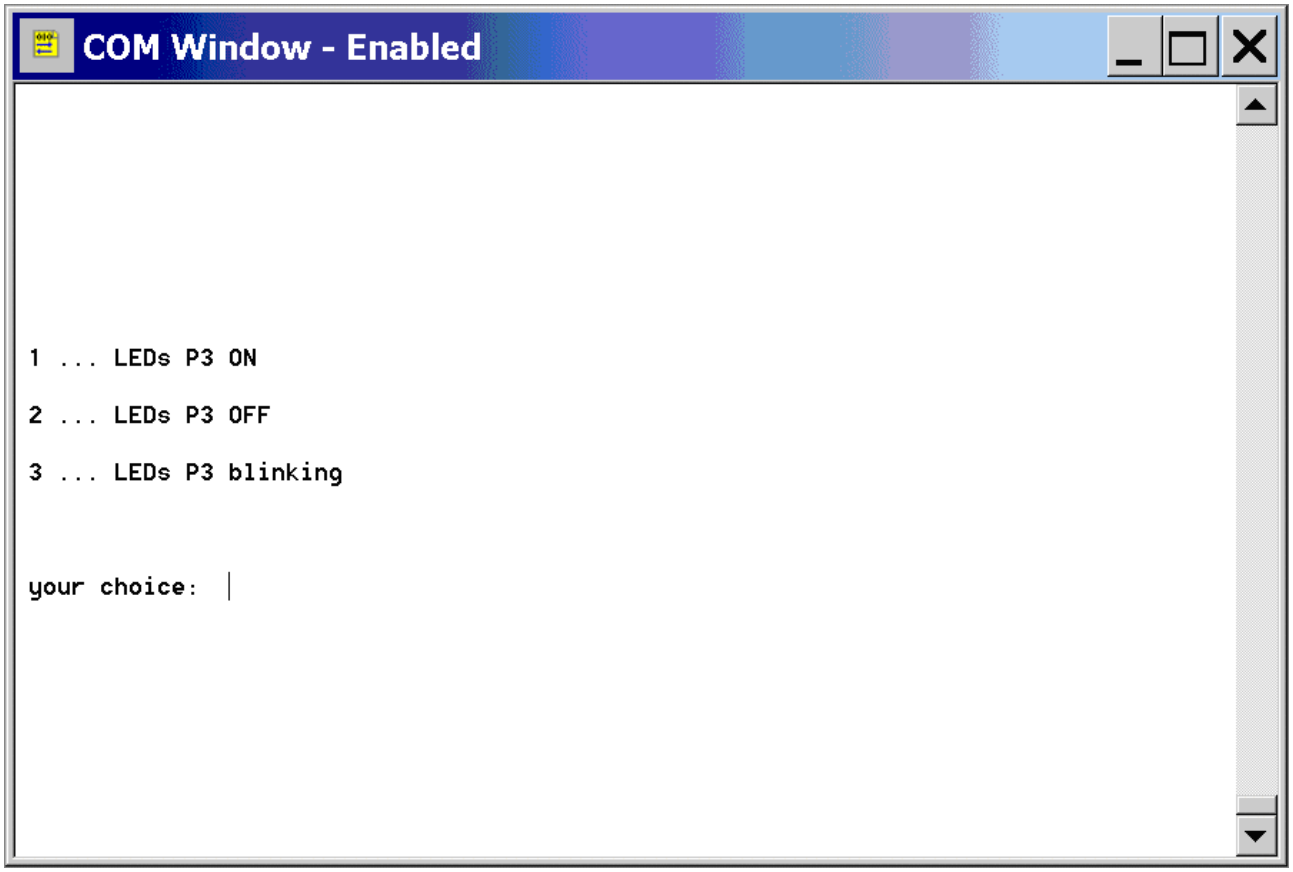
Click Yes

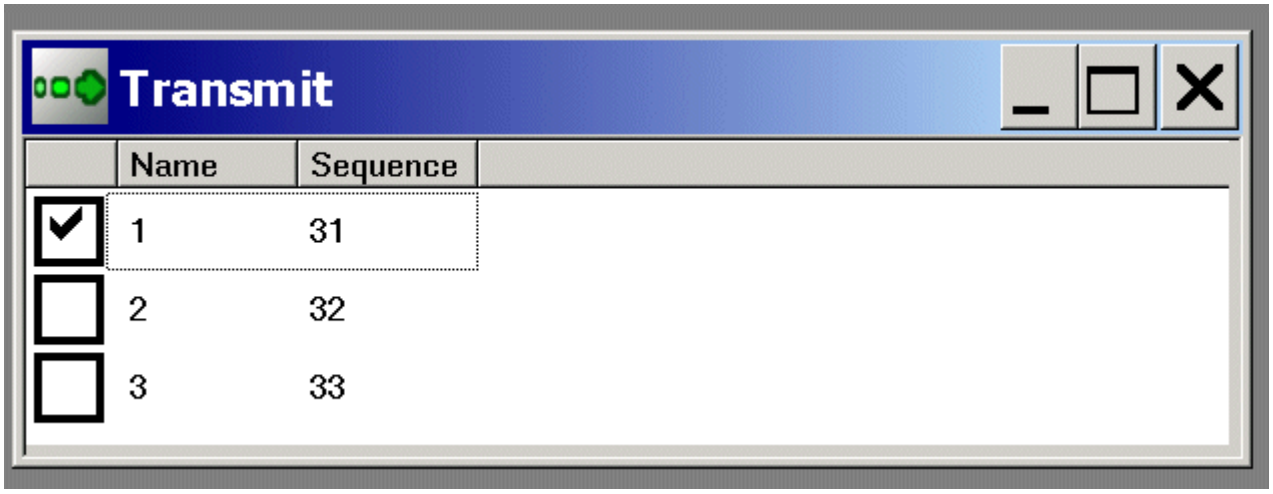

Debug - XC878/MAIN.C - DAvE-Bench for XC800
File Edit Navigate Search Project Debug Tools Window Help
Debug
XC878 Debug [HiTOP Application]
C:\Program Files\DAvE-Bench-011\HiTOPEclipse\HiTOPEclipse.e
XC878 Debug [HiTOP Application]
HiTOP Debugger (15.02.10 17:22) (Suspended)
Thread [1] (Suspended)
1 main() MAIN.C:279 C:0x00ea
C:\Program Files\DAvE-Bench-011\HiTOPEclipse\HiTOPEclipse.e
C:\XC8xx\XC878\Debug\XC878.hex (15.02.10 17:22)
Variables Breakpoint Active Proj Registers Modules
Name Value
MAIN.C MAIN.H UART.C T01.C
void main(void)
{
// USER CODE BEGIN (MAIN_Main,2)
// USER CODE END
MAIN_vInit();
// USER CODE BEGIN (MAIN_Main,3)
while(RS232_wait);
// USER CODE END
while(1)
{
Outline Disassembly
MAIN_vInit();
C:0x00ea <main>:
while(RS232_wait);
C:0x00ed <#282>:
C:0x00ef <main+5>:
C:0x00f1 <main+7>:
printf_small(menu);
C:0x00f3 <#289>:
C:0x00f5 <main+11>:
C:0x00f7 <main+13>:
C:0x00f9 <main+15>:
C:0x00fb <main+17>:
C:0x00fd <main+19>:
C:0x00ff <main+21>:
C:0x0102 <main+24>:
Console Tasks Problems Executables Memory
XC878 Debug [HiTOP Application] C:\XC8xx\XC878\Debug\XC878.hex (15.02.10 17:22)
Writable Smart Insert 279 : 1

Click: Resume

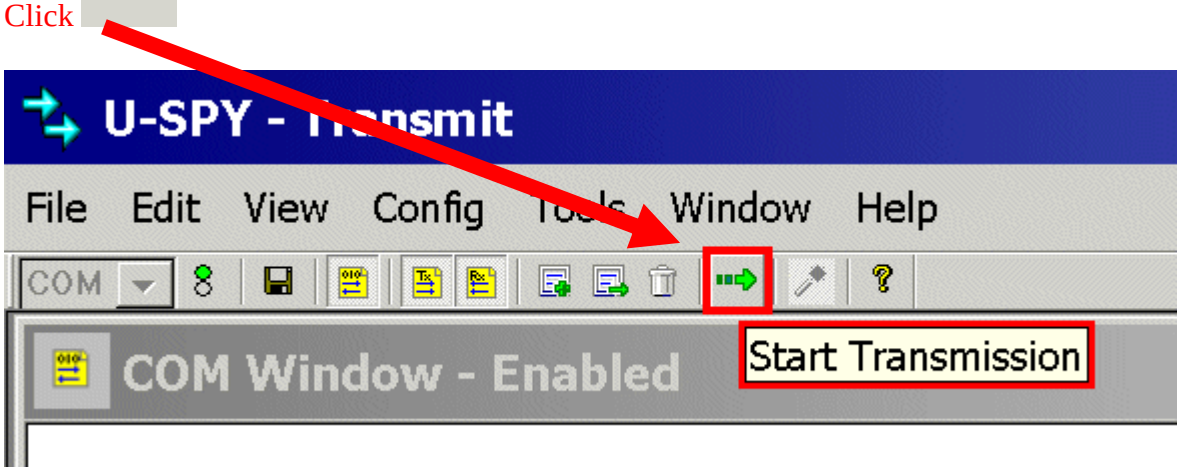


Go back to U-SPY and see the result:

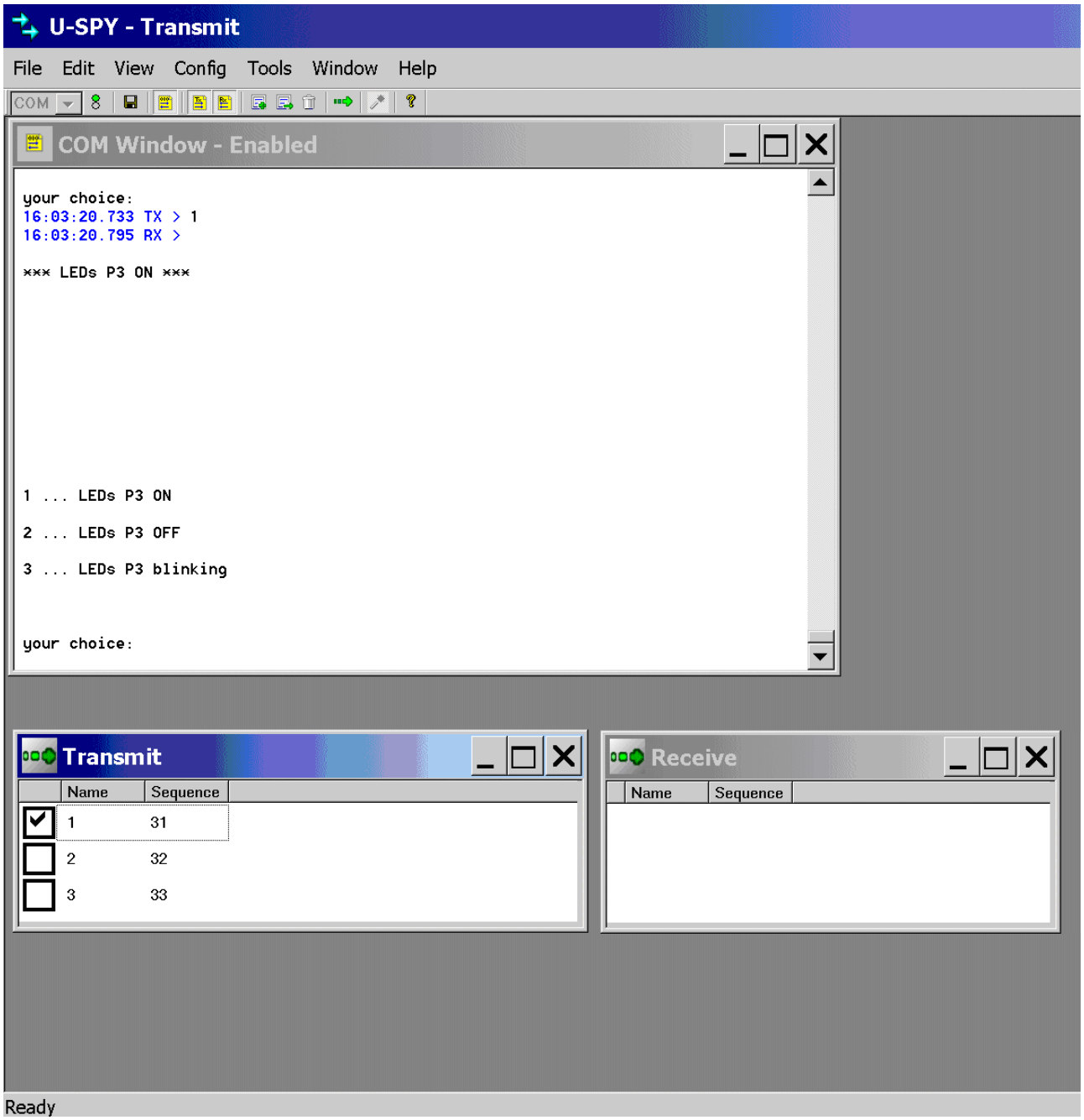




Click 



See the result:



And also check the result on your XC878 Easy Kit:

Conclusion:

In this step-by-step book you have learned how to use the XC878 Easy Kit together with the DAvE-Bench tool chain.

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

You can connect either a part of - or your entire application to the XC878 Easy Kit.

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

Have fun and enjoy working with the XC878 Easy Kit!

Note:

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

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

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

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

There are even more programming examples using the same style available [e.g. ADC-examples, CAPCOM6-examples (e.g. BLDC-Motor, playing music), Simulator-examples, C++ examples] based on these step-by-step books.

6.) Feedback (XC878 Easy Kit, DAvE-Bench): Your opinion, suggestions and/or criticisms

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

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

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

Your suggestions:

http://www.infineon.com

Published by Infineon Technologies AG