DELGEN

X-Graph and Dynamic C

User's Manual
Version 1.3 – July 22, 2010

Printed in Luxembourg.
Document Nr: XGRAPHDC-001

Content

Figures

Tables

1 Welcome

1.1 Introduction

Learn how to install and configure the Rabbit Dynamic C Compiler for use with the X-Graph modules.

Find out about the included X-Graph specific libraries and required patches to the existing Dynamic C libraries. Learn how to use the Dynamic C sample programs with the X-Graph module. Browse the list of new X-Graph functions.

Before installing this software you should read and agree with the software license agreement found on the last page of this documentation

Refer to the 'xgConsole Users Manual' if you want to operate the X-Graph module WITHOUT a C compiler.

Also read the X-Graph Module Users Manuals. These include information on the X-Graph software modules, FAT, Firmware Upgrade, GUI, DMX512, etc...

1.2 How This Book Is Organized

You can find following chapters in it:

Chapter 1 contains a view on all the information in this book.

Chapter 2 explains on how to install Dynamic C and include X-Graph support.

How to configure a target configuration is explained in **Chapter 3**.

Want to run sample programs, check **Chapter 4**.

In **chapter 5** you find all information on the programming cable needed to get Dynamic C working with the X-Graph modules.

Chapter 6 explains how to start a new project with an X-Graph module.

Chapter 7 contains detailed information on all X-Graph library functions.

1.3 More Questions

If you have questions while using your X-Graph module, check first if the information is available in this book. If you cannot find the answer check the information and forum on the X-graph website (www.x-graph.be). Finally you can also contact your local distributor or the X-Graph technical support by e-mail (techsup@x-graph.be).

This manual includes all available documentation on the X-Graph module. It is strongly advised to download and read documentation on the Rabbit processor, and the OP7200 operating console available from the Rabbit Semiconductor (www.rabbit.com) website. This manual is complimentary to the documentation found on these websites.

2 Installing Dynamic C with X-Graph Support

Dynamic C is a C-like programming language available from Rabbit Semiconductor (www.rabbit.com). It allows fast and effortless development of Rabbit applications and includes a wealth of software samples on all kind of subjects.
ANSI-C compatibility is gradually added from version 10.60 and higher.

2.1 Installing Dynamic C and X-Graph Libraries

Before you can start using the X-Graph module you should install Dynamic C, install the X-Graph libraries and upgrade some Dynamic C libraries.

2.1.1 Installing Dynamic C

Before you start using the X-Graph module, install Dynamic C as described in the Dynamic C documentation. Then read carefully all documentation included with Dynamic C.

For Rabbit 3000 based products you must install Dynamic C 9.x.
For Rabbit 4000 based X-Graph products it is advised to install Dynamic C 10.x. Most X-Graph modules also work with Dynamic C 9.x.

This manual does not copy any of the information available in the Dynamic C documentation. We encourage you to download the latest release of this documentation from the Rabbit Semiconductor website (www.rabbit.com) and regularly check for updates of these docs.

2.1.2 Installing X-Graph Libraries

The next step is installing the X-Graph libraries. You should download the latest release of the libraries from the X-Graph website (www.x-graph.be). Also regularly check for updates of these libraries. The X-Graph libraries and all future upgrades are available, free-of-charge, for all X-Graph customers. The libraries are copyrighted and the source code cannot be distributed, see the copyright notice in this manual for more information.

Just unzip the file to your Dynamic C root directory.

All files in the LIB directory should be copied to the Dynamic C LIB directory, SAMPLES in the Dynamic C SAMPLES directory, etc …

Only for DynamicC 10.x, the files in the LIB directory must be copied in the LIB\Rabbit4000 directory.

The zip file includes files for all the members of the growing X-Graph family. Although not all files in the zip are required to operate your X-Graph module, we encourage you to install all the files.

WARNING: Remember to install the files for each Dynamic C upgrade you install on your PC.

2.2 Upgrading Dynamic C Standard Libraries

To integrate X-Graph module support in the Dynamic C environment some small changes need to be done to the standard Dynamic C files/IDE. These changes do not influence any projects you would have with standard RCM modules.

WARNING: Remember to do these changes for each Dynamic C upgrade you install.

2.2.1 Include X-Graph BIOS Support

As the Dynamic C BIOS is tailored during each compile to the connected hardware, it needs to know about the X-Graph modules. All information is stored in the "xg_board.lib" file. The file needs to be loaded by the BIOS during each compile. This is easily accomplished by adding a single line to the bios C file.

- Open the BIOS\RABBITBIOS.C file
- Find the #use "BOARDTYPES.LIB" line (located very near the top of the file)
- Add the following line: #use "XG_BOARD.LIB" AFTER #use "BOARDTYPES.LIB"
- Save the file

- Open the LIB\BOARDTYPES.LIB file (only for Dynamic C 9.x)
- Find the line:
  #error "Compile mode 'Code and BIOS in flash, run in RAM' not ... board."
  (note on some DC versions #error might be #fatal)
- Remark the line:
  //#error "Compile mode 'Code and BIOS in flash, run in RAM' not ... board."
- Save the file

Dynamic C includes a basic project management system. With this system you can define a specific bios file for each of your projects. If you prefer, you can use this system for your X-Graph projects and keep the original file as-is.

The XG_BOARD.LIB library adds macro's which on some Dynamic C versions might generate warnings during compilation. You can just neglect these warnings.
If you want to remove the warnings, just double click the warning in the 'Compiler Messages' window. The correct file will load. Then add a #ifndef … #endif around the macro generating the warning.
For example:

- #ifndef CLOCK_DOUBLED
- #define CLOCK_DOUBLED 1
- #endif

Refer to chapter 6 to find out how to start a new application with this adapted BIOS.

2.2.2 Dynamic C 10.x PILOT.BIN

Dynamic C version 10.50 and higher uses an updated debugging protocol speed, which makes the debugger fail on some computers with some X-Graph modules.
The BIOS directory contains the PILOT_DC10_XG.BIN file. This is a recompiled version of PILOT.BIN which solves this problem.

If you have any debugging timeouts while using for example the XG4101 on your computer, copy the PILOT_DC10_XG.BIN file to the BIOS directory of your Dynamic C 10.x install directory. Then rename the original pilot.bin to pilotorg.bin. Finally rename the PILOT_DC10_XG.BIN file to PILOT.BIN.

2.2.3 Dynamic C 10.46 STDBIOS.C / PILOT.BIN

Only for Dynamic C 10.46, the following change must be made in STDBIOS.C (line 1007 and following):

```
; *** CS0WE0
clr hl                                  ; Also mb2 flag initial value
ex      de', hl
clr hl
ld      iy, .test_cs0we1
jr      .do_test

.test_cs0we1:
ld      hl, 0x0404
ld      iy, .test_cs1we1
jr      .do_test

.test_cs1we1:
#if (XG4000_SERIES)
ld      hl, 0x0505
ld      iy, .test_cs2we1
jr      .do_test
#endif
```

Do the changes in **bold**.

Also the PILOT.BIN file included with Dynamic C 10.46 contains a bug which prevents it from working with a high-speed 8-bit memory module, as the XG4101.
Do the changes of section 2.2.2 to solve this bug.

3 Target Configuration

Dynamic C contains a list of all known Rabbit hardware modules. The X-Graph modules are not included in this standard list. So, you will need to install the X-Graph configuration files, BEFORE using Dynamic C with any X-Graph module.

The RTI directory contains all the required RTI files. These are Dynamic C configuration files which need to be installed once. You do not need to install all RTI files, just the one which correspond with the X-Graph modules you are using. For the standard XG5000 module, you only need the XG5000.RTI file. If you own an XG5001 module (XG5000 with bb-SRAM installed), you should use the XG5001.RTI file.
For custom designed X-Graph boards, you will need other RTI files.

IMPORTANT: For some modules there are two RTI files, one for Dynamic C 9.x and another for Dynamic C 10.x. You must select the RTI file matching your Dyanmic C version.

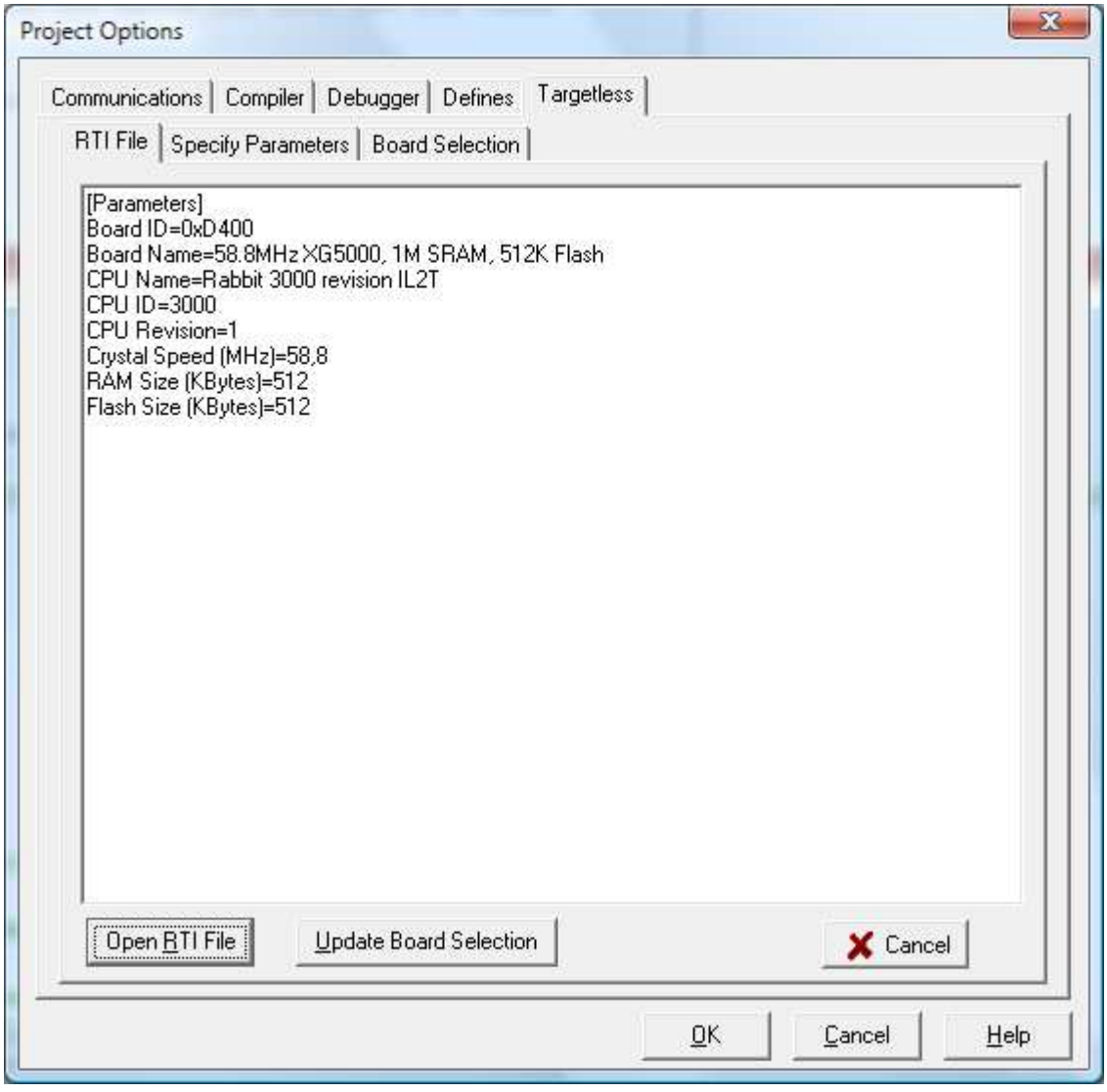

Figure 1: Dynamic C RTI File

Installation procedure:

- Start Dynamic C
- Select in the 'Options' menu the 'Project Options' item
- Select the 'Targetless' tab
- Then the 'RTI File' sub-tab
- Press the 'Open RTI File' button and use the file open box to select the correct RTI file
- Press the 'Update Board Selection' button

IMPORTANT FOR DYNAMIC C 9.X ONLY:
A bug in the RTI file read functionality of some Dynamic C 9.x versions sometime causes the CPU revision not be set correctly.
Select the 'Board Selection' tab and find the X-Graph board you just added.
The 'CPU (revision shown on chip):' field should indicate for all X-Graph products 'Rabbit 3000 revision IL2T/...'. If it doesn't indicate this, select it manually.

IMPORTANT FOR DYNAMIC C 9.X ONLY:
When you use an XG4x00 series X-Graph module with Dynamic C 9.x, the selected cpu must also be 'Rabbit 3000 revision IL2T/...'. Do not select the 'Rabbit 4000', even if it is available in the dropdown box. The Rabbit 4000 processor installed on the XG4x00 series X-Graph modules, runs in Rabbit 3000 compatibility mode if Dynamic C 9.x is used. There is no speed penalty.

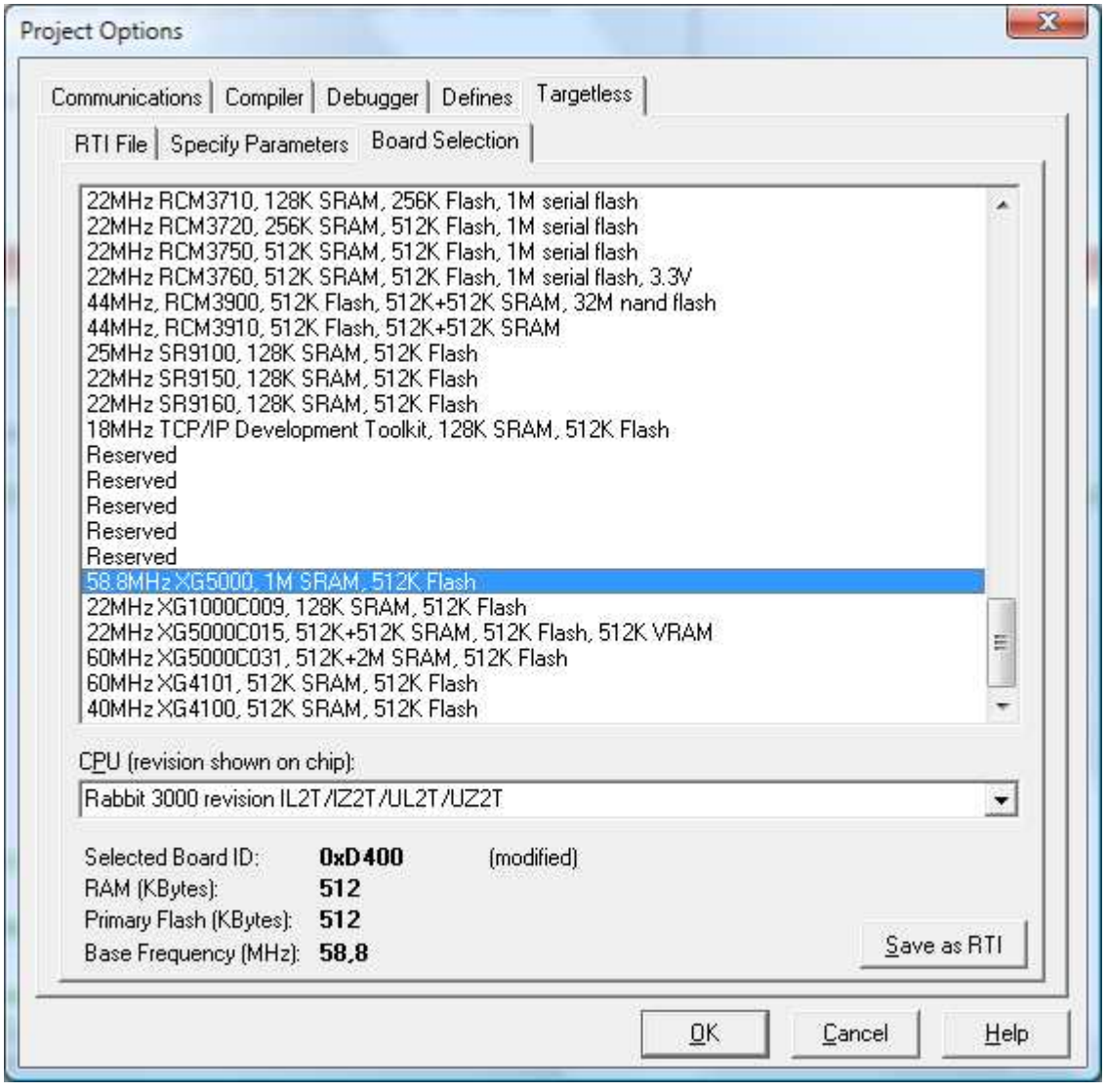

Figure 2: Dynamic C Board Selection

Note: You do not need to reinstall the RTI file if you install a new Dynamic C version. The RTI data is stored in the registry and is accessed by all Dynamic C versions.

4 Dynamic C Sample Programs

Because the X-Graph module are very compatible with the existing RCM modules, most Dynamic C sample programs work on the X-Graph modules. The OP7200 samples will, in most cases, fail to work on any X-Graph module. These Dynamic C samples often include references to the OP7200 hardware. The hardware abstraction is poorly done in the samples, which makes them fail on the X-Graph modules.

5 Programming Cable

WARNING: DO NOT insert the USB cable in the X-Graph module until you've downloaded and installed the proper drivers.

Dynamic C requires a programming cable to be connected to the target device. It uses the Rabbit serial Port A for communication with the PC.

X-Graph modules can have three different programminginterfaces.

Some X-Graph modules have a USB-to-Serial converter integrated, for example on the XG5000. Also an X-Graph USB plug-in board is available for some X-Graph modules. With such an USB port installed you just need to connect a USB cable to start debugging with Dynamic C. No need for a special programming cable.

Other X-Graph modules have a RS232C level shifter installed and use a standard RS232C 2x5 pin header. A standard null-modem cable and 2x5 pin to dSub9 convertor cable are needed for debugging. You can skip section 5.2 and following in this chapter when using this programming system.

Your X-Graph module might also include a Rabbit programming header. On those modules you will need a Rabbit programming cable. You can skip this complete chapter if your X-Graph module has this type of programming header.

Check your modules X-Graph Users Manual to verify which programming interface is available.

5.1 Programming Enable Jumper

Most X-Graph modules require a jumper to be mounted to switch the X-Graph module in programming mode. Check the X-Graph Users Manual of your hardware for the correct location of this jumper. For example on the XG5000, place a jumper on pins 7 and 9 of J21.

When you mount this jumper, SMODE0 and SMODE1 are pulled high which causes the Rabbit to switch to programming mode.

All you need to start debugging is connect the correct programming cable and install the jumper. To stop the programming mode just remove the jumper, the USB cable can be kept connected.

NOTE: some X-Graph modules have an automatic jumper detector and do not require the mounting of the jumper. For example the OP7300.

5.2 USB Programming Cable

5.2.1 USB Driver

The X-Graph modules with an USB chip use the FTDI FT232R. To use the Dynamic C environment for emulation, you need to install the USB-to-RS232C driver. This driver emulates a serial port on your PC.

Windows Vista and Windows 7 both include the FTDI USB driver.

For other Windows version a Windows driver is included with the X-Graph library set. You can find it in the 'Utilities' directory of your Dynamic C root directory. Or you can always find the latest drivers at the www.ftdichip.com website. Follow the instructions on the FTDI website. You need to select 'Drivers' in the menu and then select the 'VCP' driver. Now you can select the driver matching your OS and download the driver files.

IMPORTANT READ THIS BEFORE INSTALLING THE DRIVER:
If you want to use a higher download speed and/or want to use a production programming environment, first read the remaining sections in this chapter **BEFORE** installing the driver.

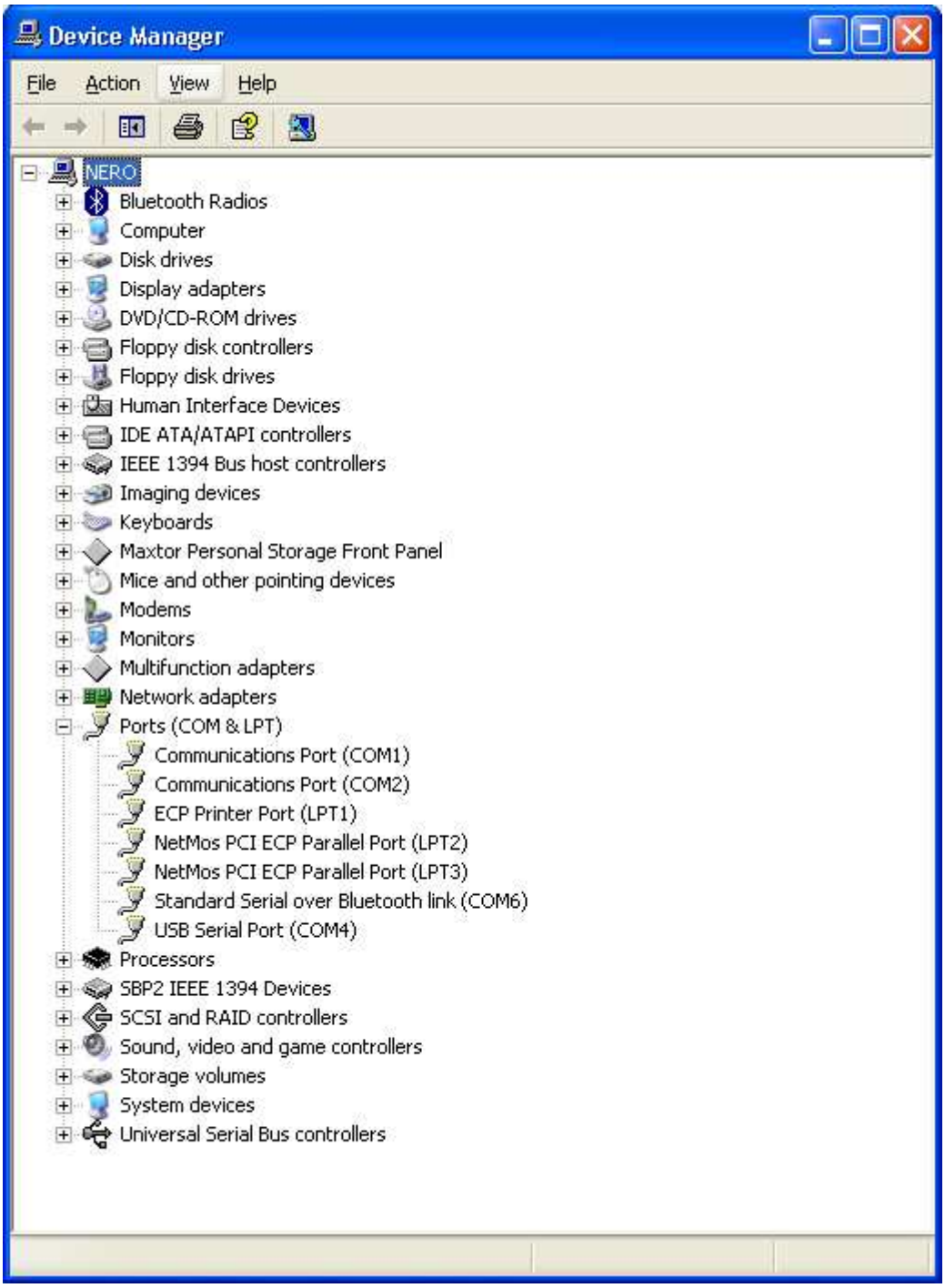

Figure 3: USB to Serial Driver in Windows XP Device Manager

To install the driver, just insert a USB cable in the X-Graph. Although some X-Graph modules do not to be powered, we advise you to first apply power to the module. The driver is installed automatically. Make sure to point Windows to the directory with the correct INF files, if requested.

You can use either the FTDIPORT.INF or FTDIPORT_XGRAPH.INF files located in the 'Utilities' directory of your Dynamic C root directory. Never use the FTDIPORT_BUS.INF file as-is, it needs changes specific for each PC setup (see the following sections).

When using FTDIPORT_XGRAPH.INF Windows will complain about an unsigned driver. This is due to the manual changes. If you don't want to use unsigned drivers, just use the standard FTDIPORT.INF file and forget about faster downloads.

A COM port number is automatically assigned. Make sure this number is lower then 9 (Dynamic C requirement). If it isn't, rearrange the port numbers until the number is lower then 9.
You can find the COM port number in the Windows Device Manager.

- Right click on 'My Computer'
- Select 'Properties'
- Select the 'Hardware' tab (on Windows Vista and earlier only)
- Then start the 'Device Manager'
- Find the 'Ports (COM & LPT) item and click on the + sign
- There should be a line 'USB Serial Port (COMx)
- Remember the COM number (x)

To get the maximum possible speed with the FTDI USB adaptor a change needs to be done in the configuration settings. Double click on the 'USB Serial Port (COMx)' line, then select the Port Settings tab.

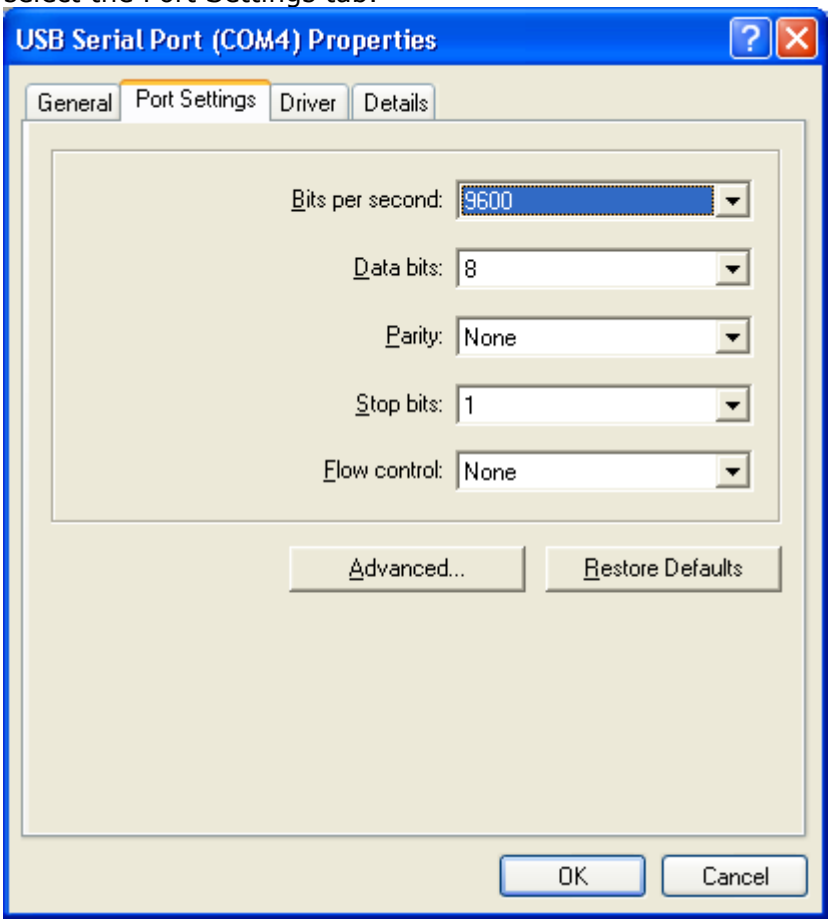

Figure 4: USB to Serial Driver Properties

Press the 'Advanced' button and change the Latency Timer from 16 to 1 msec.

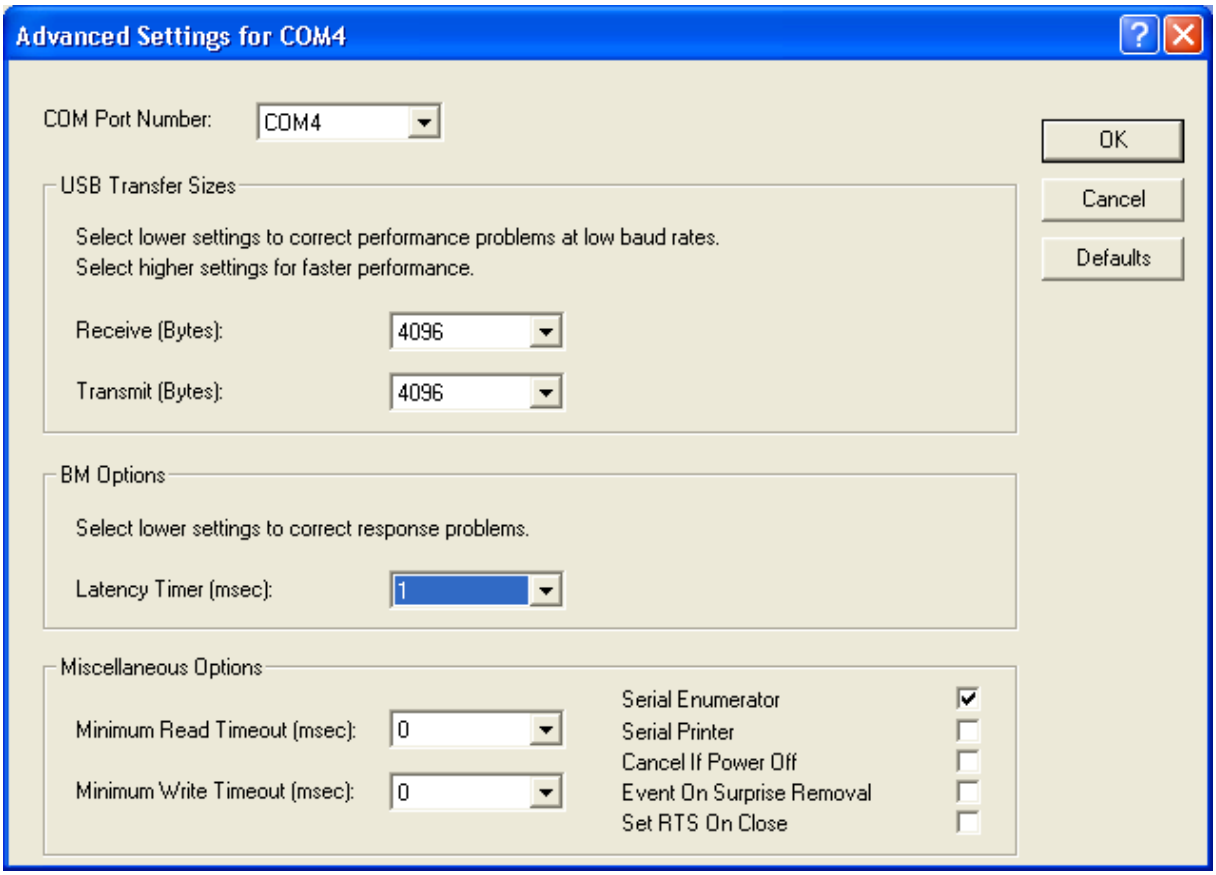

Figure 5: USB to Serial Driver Advanced Properties

5.3 Max. Download Speed of Dynamic C

The maximum speed of the FTDI USB-to-Serial port is 3MBaud. Dynamic C limits the communication speed to 460kBaud. By making some changes to the configuration files of both Dynamic C and the FTDI driver a max. download speed of 920kBaud can be achieved. If you would like to install this feature, read the this section **BEFORE** installing the the FTDI driver.

If you don't care about the increased download speed, skip this section and jump to the next section about production programming environments.

Many programmers are annoyed by the slow download speed of Dynamic C.
What protocol is Dynamic C using to make things so slow?

- First the the bootloader is send at 2400 baud (Sending Initial Loader)
- Then the Pilot BIOS is send (Sending Pilot BIOS) at 57600 baud. Once this pilot is downloaded and started it negotiates a maximum communication baudrate with Dynamic C (see paragraph 6.3). The max. baudrate is limited to 460kbaud.
- Then Dynamic C starts compiling the program. By installing the libraries and source files on a fast disk, this process will speed up. Also using a fast PC will help.
- Finally the program is downloaded at the negotiated baudrate
- As a last step Dynamic C starts the debugging environment using the 'Debug Baud Rate' (see paragraph 6.3). Generally it's better to select a slow baud rate for

this, to limit timeouts during debugging. Slower baudrates do not effect the download speed on the X-Graph module.

The longest wait time is located in step 4. The X-Graph USB interface can handle speeds up to 3MBaud, but some other parameters need to be considered.

- The USB chip base clock is 3MHz and it only has a limited number of dividers.
- The Rabbit 3000 on a XG5000 has a 58.8MHz clock speed. Its base clock for serial ports is 1.8475MHz, which is also the max. allowed baudrate. On a 60MHz XG4x00 the base clock slightly increases to 1.875MHz.

The highest common baudrate of these two chips is 923750 (Rabbit @ 58.8MHz) and 923077 (USB).
If it would be possible to select a 3MHz divider in the Rabbit 3000, it would just be possible to download at 3MHz. But to do that the cpu needs to be clocked at 48MHz (an option on the XG5000) and the serial port pre-divider needs to be set to 1 (not available in the pilot bios).

Because the Dynamic C IDE has no option to select this baudrate (see paragraph 6.3) a trick can be used, as described in the AN232B-05 application note on the FTDI website. This trick makes the FTDI chip generate a different baudrate from the one selected by the application. By just selecting 230400 baud in the Dynamic C Options-Communications window, you tell Dynamic C to use this baudrate for downloading. Dynamic C then sets this baudrate via the FTDI Windows driver. Only we can configure this driver, when it receives a command to set the baudrate to 230400, to actually set it to 923077 baud.

A similar trick needs to be done with the Pilot BIOS. When it gets a command from Dynamic C to switch it's baudrate to 230400 baud, we'll need to change the pilot bios to actually set the baudrate to 923750 baud.

5.3.1 FTDI INF Changes

Note: the following section is included for customers whom which to change the INF file manually after downloading it from the FTDI website and **BEFORE** installing the driver. The 'Utilities' directory (found in the Dynamic C root directory) includes an already changed INF file: FTDIPORT_XGRAPH.INF. Below changes are already done in this file.

Unzip all the files of the FTDI driver in a directory. Then open the FTDIPORT.INF file and find the following section:

Note: there are two historical versions of FTDIPORT.INF in which the 'ConfigData' section is slightly different. Have a look at the following two examples:

```
[FtdiPort232.NT.HW.AddReg]
HKR,,"UpperFilters",0x00010000,"serenum"
HKR,,"ConfigData",1,01,00,3F,3F,10,27,88,13,C4,09,E2,04,71,02,38,41,9c,80,4E,C0,34,
00,1A,00,03,80,06,40,03,80,00,00,d0,80
HKR,,"MinReadTimeout",0x00010001,0
HKR,,"MinWriteTimeout",0x00010001,0
HKR,,"LatencyTimer",0x00010001,1
```

```
[FtdiPort232.NT.HW.AddReg]
HKR,,"UpperFilters",0x00010000,"serenum"
HKR,,"ConfigData",1,11,00,3F,3F,10,27,00,00,88,13,00,00,C4,09,00,00,E2,04,00,00,71,
02,00,00,38,41,00,00,9C,80,00,00,4E,C0,00,00,34,00,00,00,1A,00,00,00,03,80,00,0
6,40,00,00,03,80,00,00,00,00,00,00,D0,80,00,00
HKR,,"MinReadTimeout",0x00010001,0
HKR,,"MinWriteTimeout",0x00010001,0
HKR,,"LatencyTimer",0x00010001,1
```

Make the 3 (three) changes indicated in **bold** -> don't forget the LatencyTimer change.

IMPORTANT: By doing this change, the driver will never be able to use a 230.400 baud rate. This will be changed to 923.077 for all Windows applications using the driver.

IMPORTANT for Windows x64:
Windows x64 only allows you to install signed drivers. By doing above change the driver is not signed anymore and Windows x64 will refuse to use the driver.
You can force Windows x64 to use unsigned drivers by pressing F8 when Windows is booting. Then select in the Advanced Boot Options, 'Disable Driver Signature Enforcement'.
Or you can do the necessary changes in the registry, see next section.

5.3.2 FTDI Registry Changes

If you already installed the driver before doing above changes, you can either reinstall the driver or make above changes directly in the Windows registry.

Start regedit and search for the following key:
HKEY_LOCAL_MACHINE
SYSTEM
CurrentControlSet
Enum
FTDIBUS
VID_0403+PID_6001+…. (there might be multiple entries, do the changes to all)
0000
Device Parameters
ConfigData

In the ConfigData entry, make the same changes as you should do in the INF file (see paragraph 5.3.1).

5.3.3 Change PILOT.BIN

Find the PILOT.BIN and COLDLOAD.BIN files, located in the BIOS directory of your Dynamic C installation. First make a copy of these files. We suggest you make a directory BACKUP and copy the file to this directory. If you would ever want to reuse the original files, you only need to copy the files from the BACKUP directory.

A batch file is available in the BIOS directory to create and restore the backup files:

- dc_make_backups.bat creates the BACKUP directory and copies the files. This batch file also creates an additional set of backups in the BIOS directory (dc_pilot.bin and dc_coldload.bin).
- dc.bat restores the original files

The BIOS directory includes already changed PILOT.BIN and COLDLOAD.BIN files, adapted for 920kBaud (920kbaud_pilot.bin, xg4x01_coldload.bin). Note that these files are based on the original Dynamic C 9.50 pilot.bin file. Earlier or future versions of Dynamic C might not work with this adaptation.

Several batch files are available in the BIOS directory to change the coldloader files easily:

- xg5000_920.bat -> make changes needed for all X-Graph modules except the XG4x01 modules.
- xg4x01_920.bat -> make changes needed for all 60MHz XG4x01 modules
- dc.bat -> restore default files

If the included files do not work with your Dynamic C version follow below instructions to make the necessary PILOT.BIN changes manually.

Open the PILOT.BIN file in a hexeditor. A good freeware hex editor DELGEN recommends can be downloaded from www.hexedit.com. A copy is also included with the X-Graph libraries. You can find it in the 'Utilities' directory of your DynamicC root directory. The program needs to be installed. Note that this program is freeware for non-commercial use only. It can be used for a 30-day trial period for commercial use.

Search in the PILOT.BIN for the following sequence, make the one change indicated in bold and save the file.

00 E1 00 00 03 00 C2 01 00 06 00 84 03 00 **0C**
00 E1 00 00 03 00 C2 01 00 06 00 84 03 00 **30**

That's it. By doing this you reduced the download time of a typical 250kByte program to 10 seconds, just by using a USB connection on your X-Graph module.

Make sure you select 230400 baud (see paragraph 6.3) in the Project Options-Communications window of Dynamic C. This will trick the FTDI driver to use 920kbaud.

The download time can be reduced further and the flash-wear can be illiminated by using the X-Graph Fast-SRAM download method. Refer to the 'xgBIOS Users Manual' for more information.

5.4 Multiple Converter Problem

If you only use a single X-Graph module you can skip this section.

A common problem occurs in a manufacturing environment when it is required to test or program several X-Graph modules. Each module has an FTDI chip with a unique serial number and will reinstall the driver. This will also install a new COM port for each X-Graph module which also requires a reconfiguration of the Dynamic C option parameters. Further if the system limit of 256 COM ports is reached, the only way to continue testing/programming is to free up COM ports by uninstalling and then reinstalling the driver.

The problem occurs because each X-Graph appears to the system as a unique device. Each device is identified by its serial number and the serial number is used by the VCP driver when it creates a COM port. If the serial number was not used to create the COM port, then it would be possible to setup an environment where every X-Graph installed on the same COM port. Therefore, only one COM port would be needed to test and/or program multiple X-Graphs.

The idea is to allocate a COM port using information derived from the cable's physical location on the USB bus instead of the serial number. If the cables for different X-Graphs are always plugged into the same USB port, then they will appear to the system as the same COM port.

The FTDIBUS.INF file needs to be changed to get this behavior. The FTDI driver directory in the Utilities directory (Dynamic C root directory) contains an example of an adapted FTDIBUS.INF file, i.e. FTDIBUS_XGRAPH.INF. This adapted INF file works only with a preset USB port, you will need to read below section to make changes to this file compatible with your PC setup.

5.4.1 Find the LocationID with USBView

Location Ids can be obtained using the utility USBView that is available from FTDI (www.ftdichip.com, select "Resources' the 'Utilities'). To get the Location ID for a specific USB port, run USBView and plug a device into the port that is required.
IMPORTANT: do NOT plug the X-Graph into this port. Use some other USB device.

From the menu select Options->LocationIDs. Press F5 to refresh the display and the port numbers are replaced with Location Ids of the form LocXY where X and Y are hexadecimal digits. XY is the Location ID that you need to make changes to the FTDIBUS.INF file.

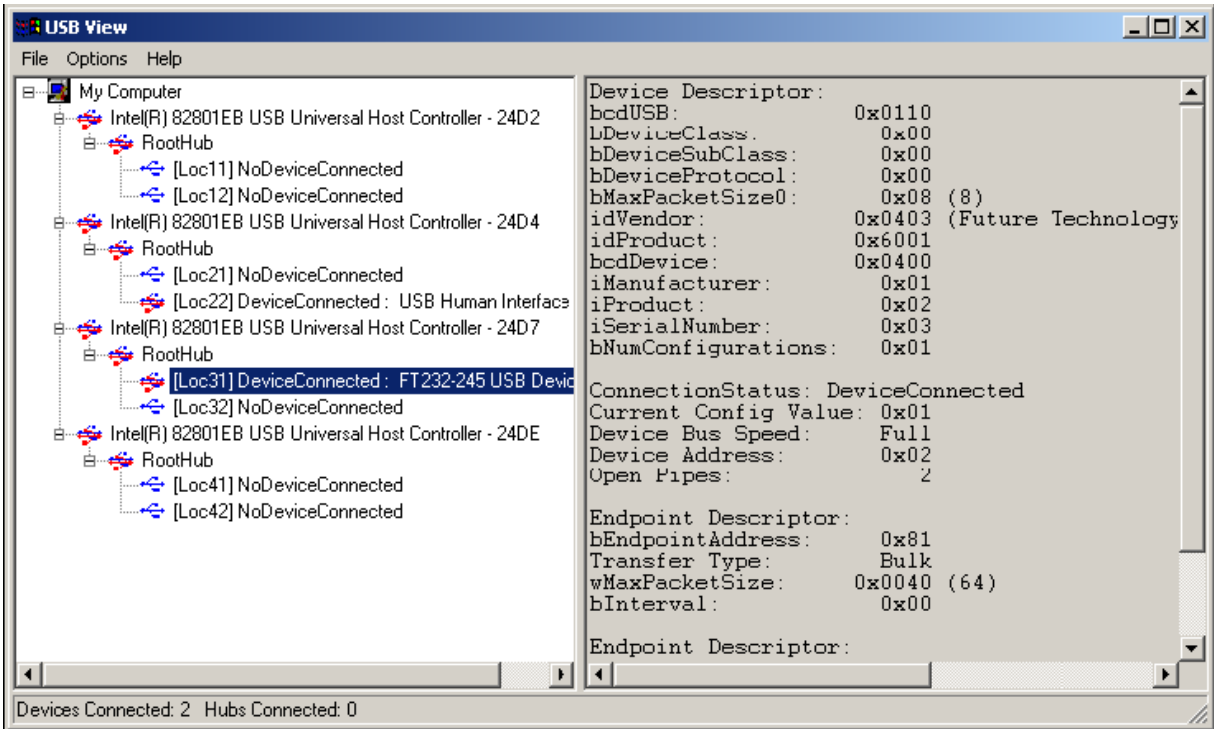

Figure 6: USBView

5.4.2 Implementation

Open the FTDIBUS.INF file and add the following lines:

[FtdiBus.NT.AddService]
AddReg = FtdiBus.NT.AddService.AddReg

[FtdiBus.NT.AddService.AddReg]
HKR,Parameters,"LocIds",41,00,00,00,00

A Location ID is a 32-bit unsigned integer that represents the location of the device in the USB tree. See the USBView example above. The "FT232/245" device is located at Location ID "31".
This 31 represents host controller 3 and port 1. Only, in then Locids value, host controller number need to be increased by one. That's why above setting is 41.

5.5 USB DLL Driver

To increase communication speed with the Rabbit microprocessor, a dedicated application can also use the FTDI DLL driver. This can increase the communication speed up to 3MBaud by using the correct clocks/dividers on the X-Graph module.

DELGEN does not offer specific drivers. The latest driver can always be downloaded from the FTDI website.

Note that this driver can not be used for debugging only for application protocols.

5.6 X-Graph Firmware Updater

Most X-Graph modules are delivered with the 'X-Graph Firmware Upgrader' utility in Ethernet mode (xgFUP Users Manual) installed in Flash. Using this utility, you can easily install your own firmware in the modules Flash through the Ethernet interface. There is no need for the RS232C based, slow, programming interface in a production environment.

6 How to start with a new application with Dynamic C?

6.1 Create a New Project

- Create a project directory (example: C:\Projects\GraphicLCD)
- From your Dynamic C directory copy the following file to this directory:
  - SAMPLES\X-GRAPH\XGRAPH_JUMPSTART.C
- Rename the XGRAPH_JUMPSTART.C file to a name with a useful meaning (example: CSTN_QVGA.C)
- Start Dynamic C
- Select File – Project – Create
- Choose a project name and save it to the project directory

6.2 Select a X-Graph Target

- Select Options – Project Options
- Then the 'Targetless' tab
- Then the 'Board Selection' tab
- Now select the correct X-Graph Module

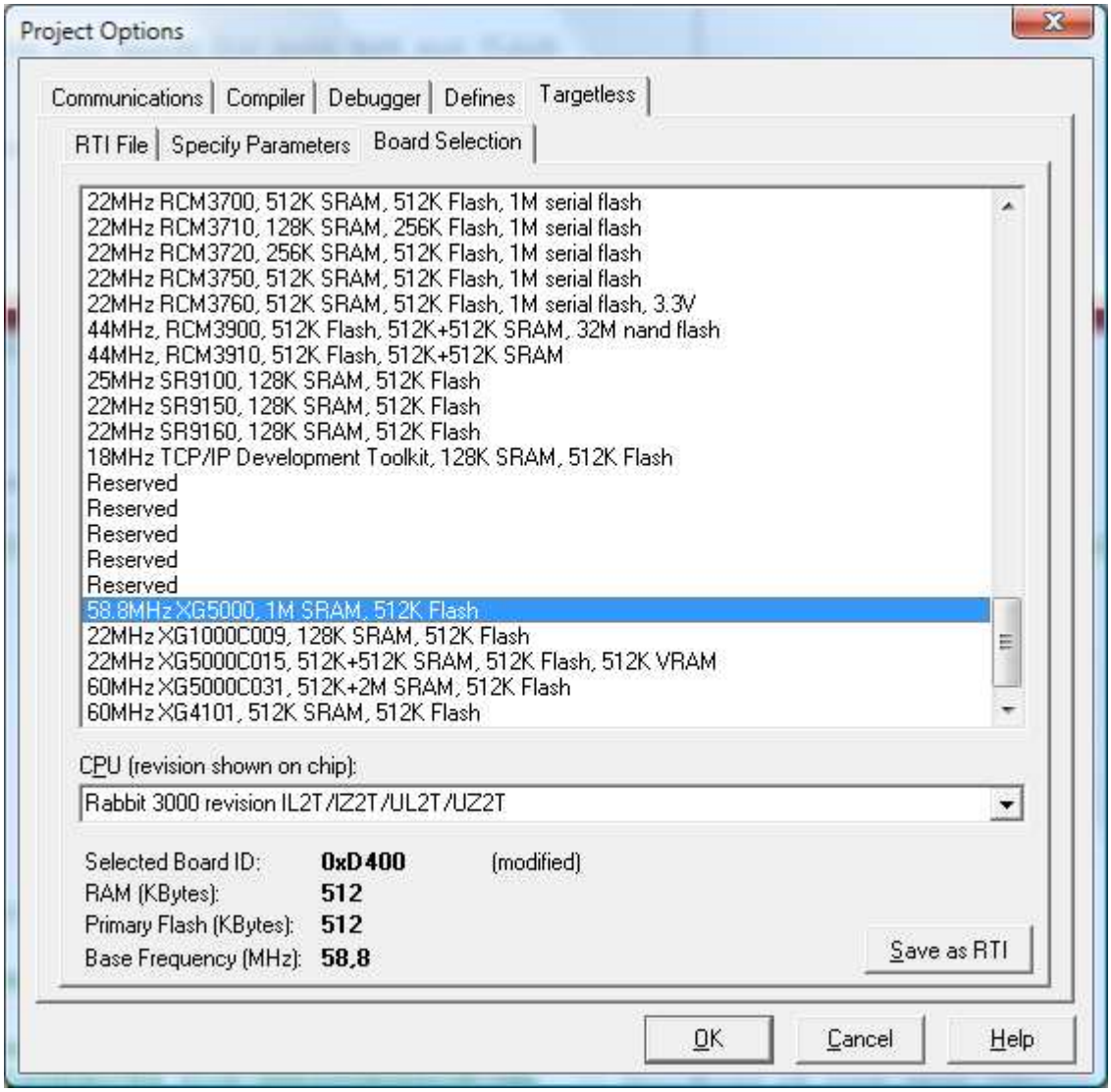

Figure 7: Dynamic C Select Target

6.3 Configure the Communication Parameters

- Select Options – Project Options
- Then the 'Communications' tab
- Connection Type: Use Serial Connection
- Debug Baud Rate: 57600 (or 115200)
- Max. Download Baud Rate: 460800 (or 230400 if you want the max. download speed as described in section 5)
- Serial Port: the COM port number assigned to the FTDI driver
- Stop Bits: 2
- Enable Processor verification: DISABLE (for X-Graph modules)
- Enable Processor verification: ENABLE (for OP7300/OP7400/OP... modules)
- Use USB to Serial Converter: ENABLE

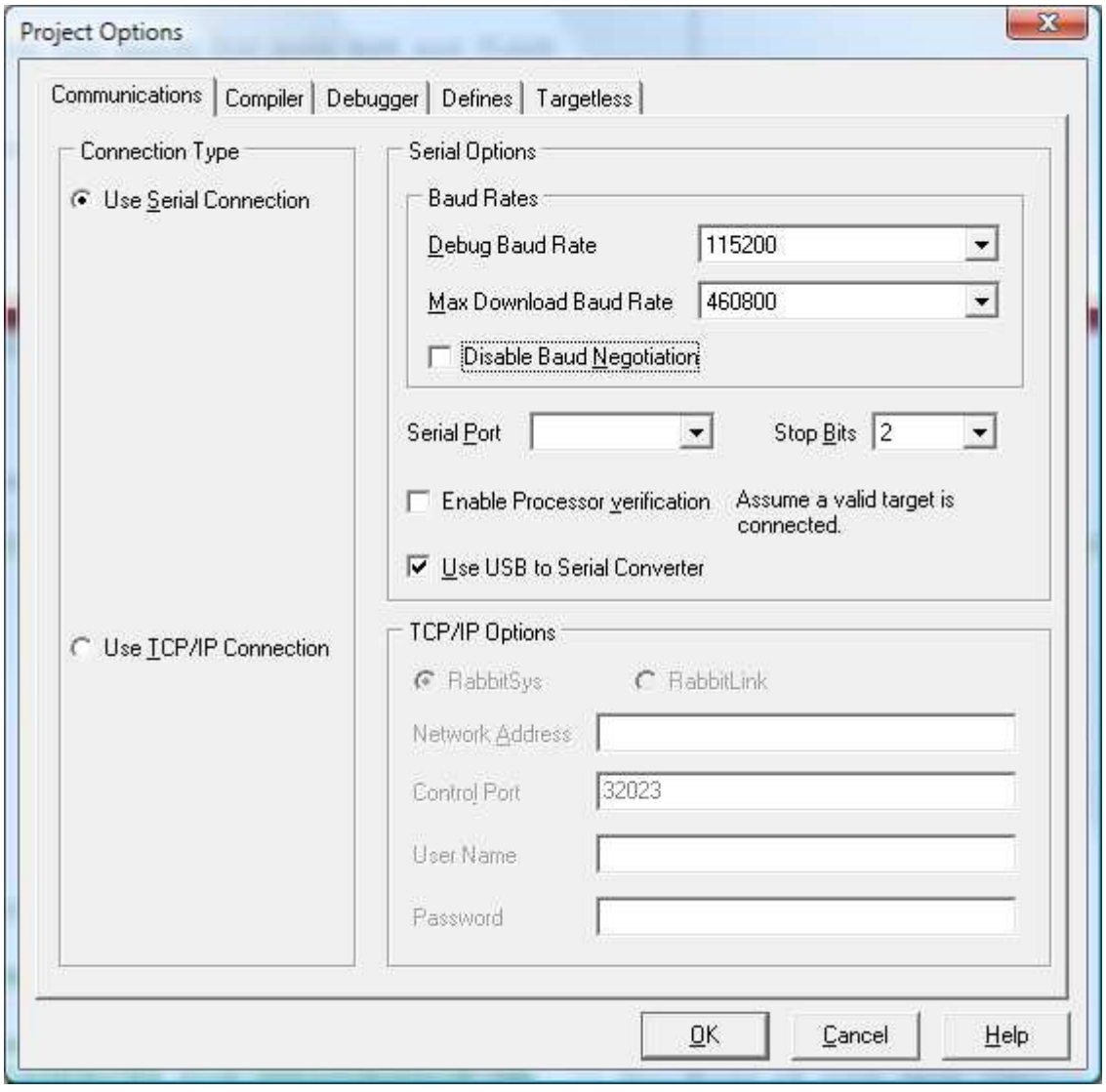

Figure 8: Dynamic C 9.x Set Communication Parameters

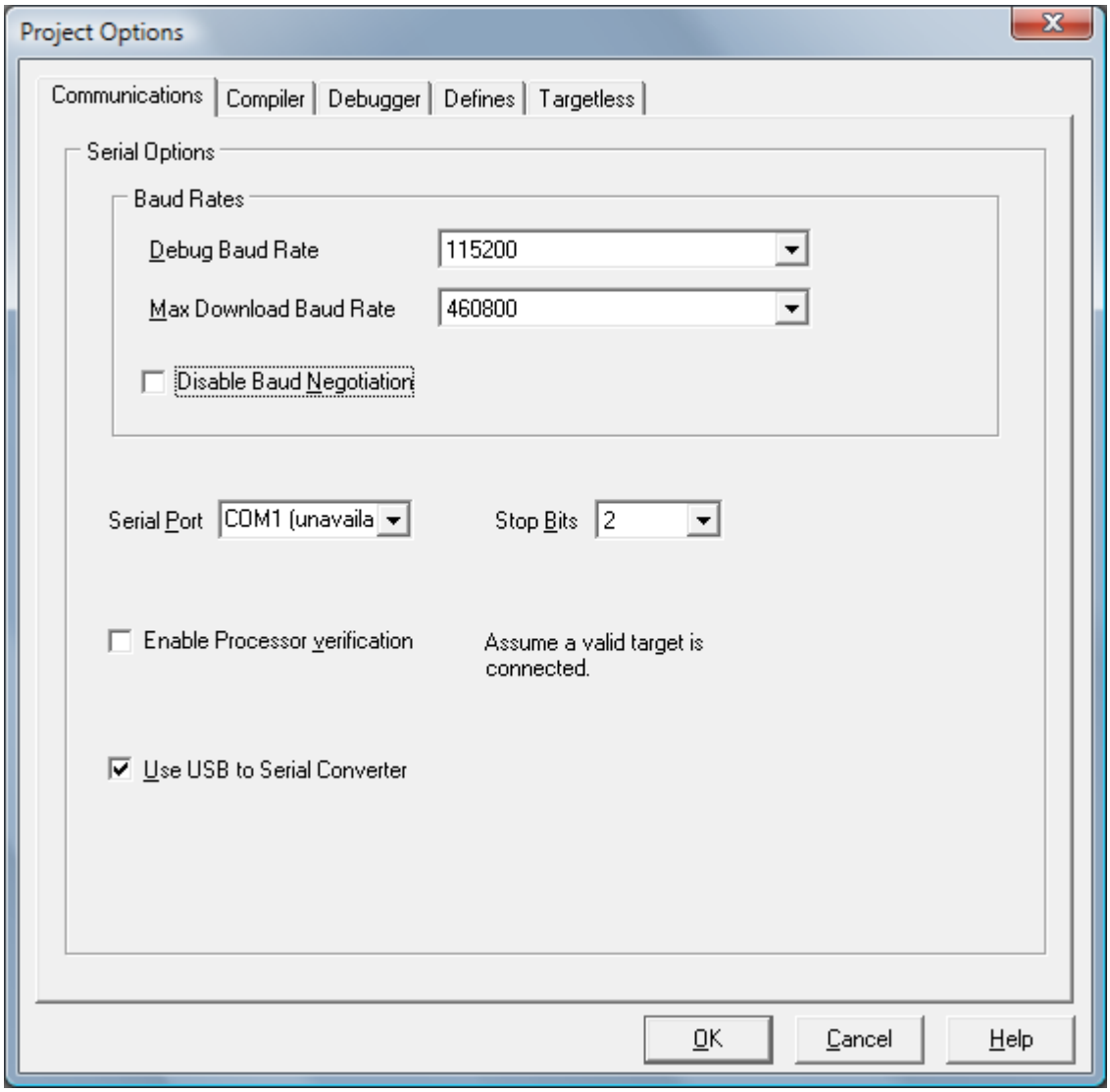

Figure 9: Dynamic C 10.x Set Communication Parameters

6.4 Configure the Compiler Options

- Select Options – Project Options
- Then the 'Compiler' tab
- In the item 'BIOS Memory Setting', make sure the 'Code and BIOS in Flash, Run in RAM' option is enabled
- Select the other Compiler and Debugger options as you require it

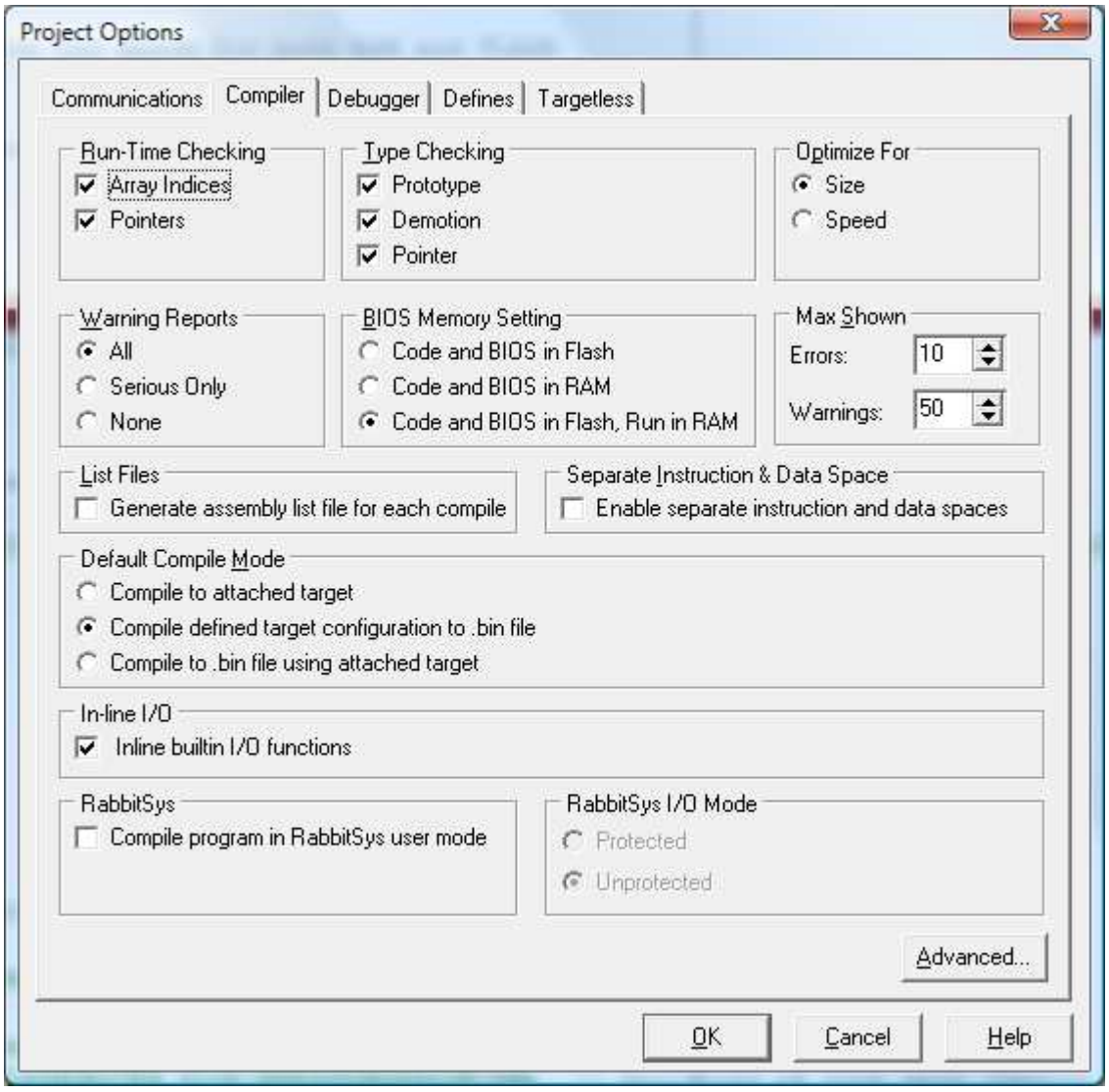

Figure 10: Dynamic C 9.x Compiler Options

- In Dynamic C 10.x always select 'Flash' in the 'Store Program in' section.
- For X-Graph modules sure to DISABLE the 'Attached Target Memory Type' 'Detect Settings' checkbox. Then select 'Flash Type' as Parallel and 'Memory Width' as '8-bit'
- For OP modules 'Detect Settings' MUST be checked.

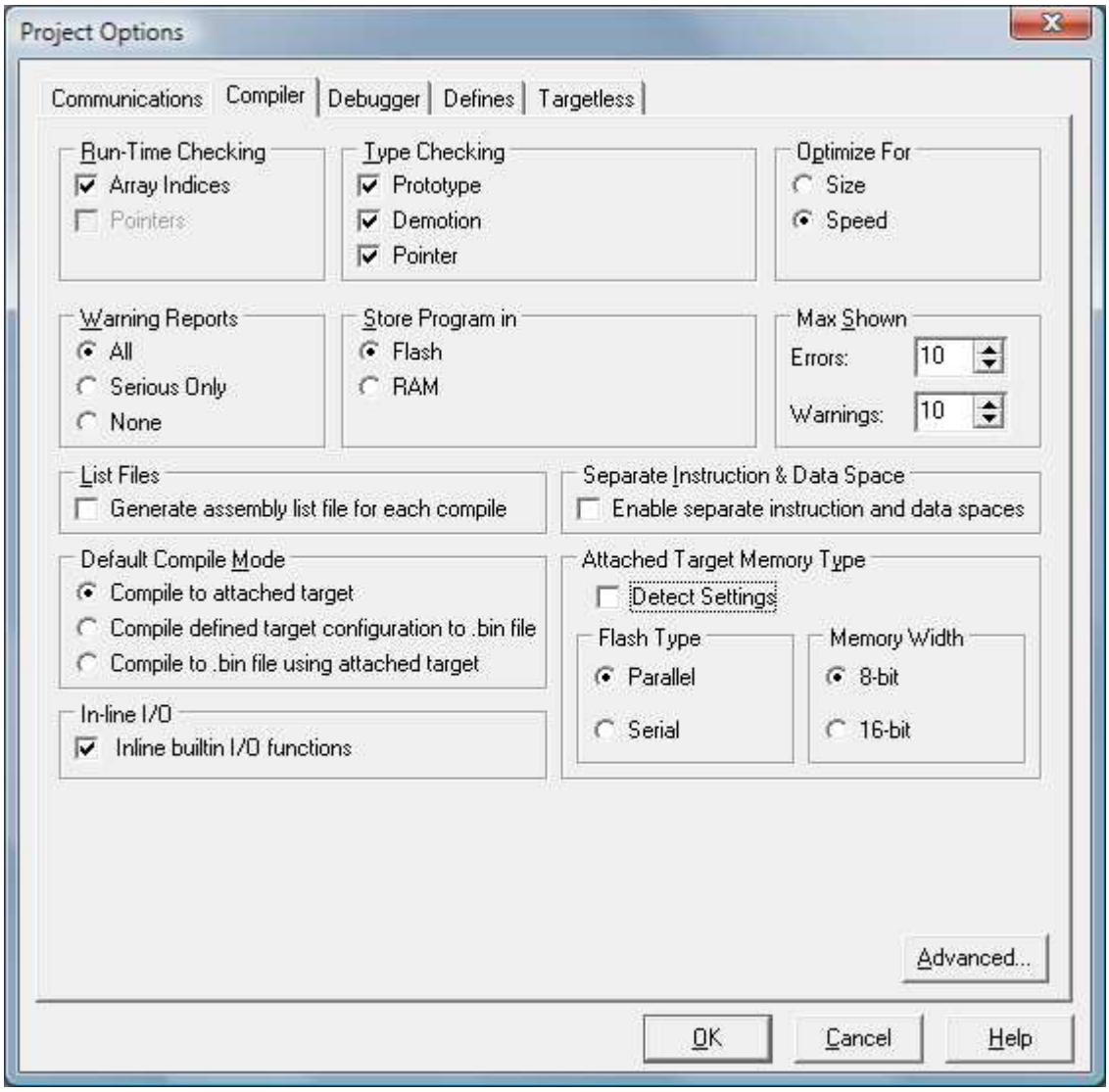

Figure 11: Dynamic C 10.x Compiler Options

6.5 Configure the Paths

- Press the 'Advanced' button
- If you prefer to work with seperated projects you can tick one or more of the 'Use buttons' and include the correct paths and filenames.

Note: for Dynamic C 9.50 and before it's recommened to use a specific X-Graph LIB.DIR and use this system to attach this DIR file to the project. For all newer Dynamic C version this is no longer required.

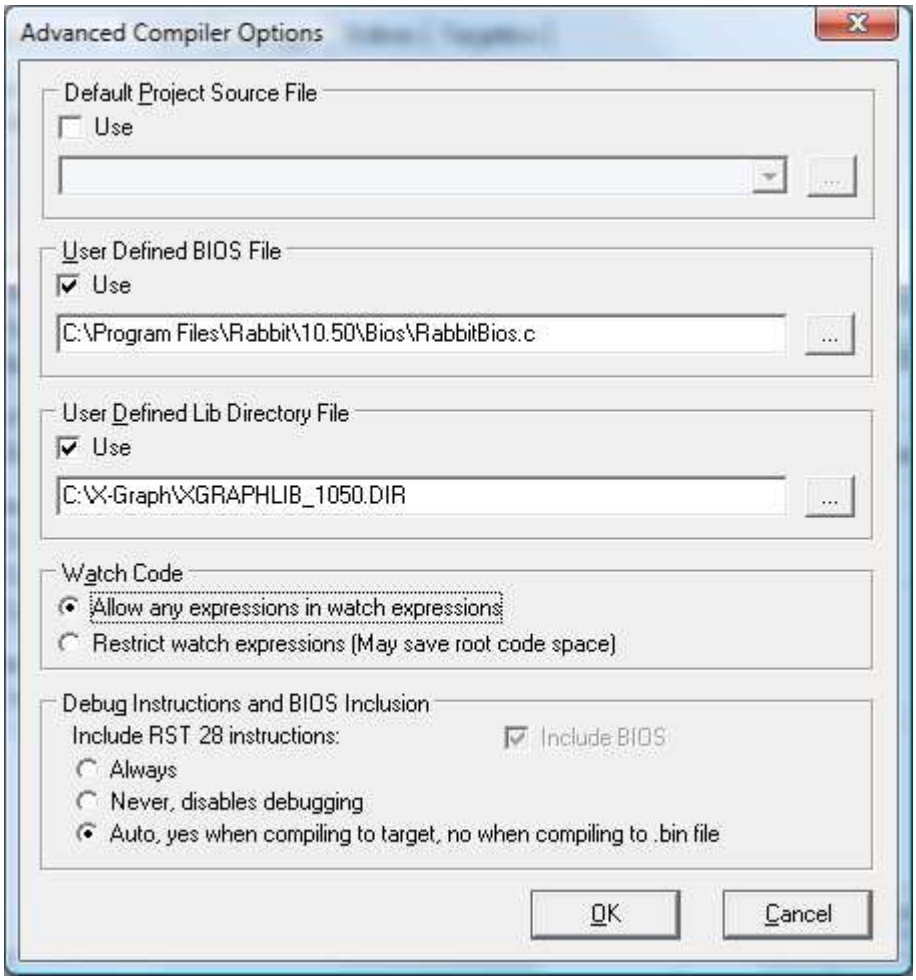

Figure 12: Dynamic C Advanced Compiler Options

6.6 Load the X-Graph Jumpstart Application

The X-Graph jumpstart application is included in the samples directory to get a new project quickly running. By always using the jumpstart file, you will not have to mind any configuration parameters for the X-Graph module.

If you have followed the instruction in paragraph 6.1, you've already copied the xgraph_jumpstart.c file and renamed it.

Now open the file and have a look at the macro's in the top part of the file. Carefully read the comments and enable the options you need for your application.
IMPORTANT: take enough time to set the correct options !!!!

6.7 Sample Programs

You can also just try to compile and download the samples included with the X-Graph module and/or with Dynamic C. Before loading any of the X-Graph samples, remember to do the steps mentioned in section 6.3.

Learn how to use Dynamic C by just browsing through the samples and testing them.

7 X-Graph Dynamic C Libraries

The X-Graph zip file contains many X-Graph related libraries to support the new X-Graph features. There are libraries to support the standard features. There are also libraries to replace existing Dynamic C libraries, mainly to get a better performance or smaller memory footprint. Some additional high-level libraries are available to make the design of X-Graph software applications easier. Refer to the X-Graph Software Module Manuals for more information on these libraries.

7.1.1 Standard Dynamic C Libraries

The standard Dynamic C libraries support most X-Graph interfaces, i.e the Ethernet, RS485, RS232C, USB (RS232C), B/W Graphics, …

We suggest you read the full Dynamic C library documentation. Also check the documentation available on the Z-World OP7200 operator interface. This documentation includes information on using the graphic and touch screen libraries. Due to copyright reasons, we cannot include copies of this documentation, but the files are available free of charge at the Rabbit Semiconductor website (www.rabbitsemiconductor.com and www.zworld.com).

7.1.2 Specific X-Graph Libraries

Standard Dynamic C libraries do not support some advanced X-Graph features. That is why DELGEN includes specific X-Graph libraries to support all new hardware features. You will find information on these libraries in section 7.2 and following.

7.1.3 Advanced X-Graph Libraries

DELGEN is offering, free-of-charge to all its X-Graph customers several libraries which add advanced functions to Dynamic C. These include the X-Graph Firmware Upgrader, FAT library, and a GUI library. Refer to the X-Graph Software Module Manuals for more information on these libraries.

7.2 X-Graph Dynamic C Hardware Support Libraries

7.2.1 X-Graph Low-Level Board Support

void xgraph_init(void);
void brdInit(void);
Initialize the board to default values based on the XG_**** macros enabled in the jumpstart application. This function MUST be called before any other X-Graph function is used.

IMPORTANT: this function is OBSOLETE, only included for compatibility with old projects.

PARAMETER
None

RETURN VALUE
None

void xgraph_init_all(void);
Initialize the board to default values based on the XG_**** macros enabled in the jumpstart application. This function MUST be called before any other X-Graph function is used.

PARAMETER
None

RETURN VALUE
None

void xgraph_tick(void);
Checks changes in low-level X-Graph drivers and updates low-level status based on user XG_**** defines. Must be called in the mainloop regularly. No minimum call frequency is defined.

IMPORTANT: this function is OBSOLETE, only included for compatibility with old projects.

PARAMETER
None

RETURN VALUE
None

void xgraph_tick_all(void);
Checks changes in low-level X-Graph drivers and updates low-level status based on user XG_**** defines. Must be called in the mainloop regularly. No minimum call frequency is defined.

PARAMETER
None

RETURN VALUE
None

void XGraphReset(void);
Forces the slave processor to hardware reset the Rabbit. The complete board is reset with the slave processor watchdog.

PARAMETER
None

RETURN VALUE
None

void SlaveInit(void);
Initialize the slave processor, reads the version of the slave's firmware and configures the slave processor based on the macro's defined in the jumpstart file

This function is called by xgraph_init() and should never be called by the user application.

PARAMETER
None

RETURN VALUE
None

void SlaveTick(void);
Checks the slave event queue and reads pending events. Each detected event makes a call to ui_event() (located in your ui.c file).
In the standard x-graph application logic, the mainloop calls regularly xgraph_tick() (xgraph.lib). This function, among other things, calls SlaveTick(). This function checks for slave based events. If an event occurred, it will call ui_event() to process the event in the user interface.

Check paragraph 7.3 to get more information on X-Graph event handling.

This function is called by xgraph_tick(). It can be called in the user application to get a faster processing of events, but be carefull due to the callback function ui_event().

PARAMETER
None

RETURN VALUE
None

7.2.2 LCD

This section only includes the LCD hardware related functions. Check the 'xgGUI Users Manual' for more information on how to use graphic primitives and widgets.

void xgDispOnOff(int onOff);
void glDispOnOff(int onOff);
Turns the LCD on or off. The LCD is off by default (power saving mode) when the X-Graph powers up.
Note: most LCD's do not have an LCD off line. The reached power saving depends on the used LCD. The X-Graph hardware continuously generates LCD timing signals, independent of the LCD being switched on or off. This to prevent damage to the LCD's. If you need a low-power LCD-off solution, contact DELGEN.

PARAMETER
0 = LCD off
1 = LCD on

RETURN VALUE
None

void xgBackLight(int iLevel);
void glBackLight(int iLevel);
Controls the intensity of the backlight.

A Rabbit PWM channel is used to modulate the backlight intensity. Depending on the LCD type, this will result in a very good to a very poor backlight control. Unless required by the application, DELGEN advises to only switch on or off the backlight.
IMPORTANT: do NOT use the PWM modulation the set the backlight intensity of CCFL backlight LCD's. Using this PWM system will generate a lot of electric and magnetic interferences.

PARAMETER
0 backlight off up to 1023 (maximum backlight)

RETURN VALUE
None

void xgSetContrast(unsigned contrast);
void glSetContrast(unsigned contrast);
Electronic contrast control is supported on B/W and CSTN LCD's. TFT LCD's do not require contrast control.

Warning: some low or high contrast values might make the LCD unreadable. If such a contrast level is used, a user might not be able to access the software functions to change the contrast level to normal values. A security system should also be available in the application software.
The X-Graph library sets the contrast to a factory default value after power-on and limits the electronic contrast level to readable levels (batch dependant).

PARAMETER
0 (minimum contrast) up to 100 (maximum contrast)

RETURN VALUE
None

7.2.3 MMC/SD

The X-Graph MMC/SD slot accepts all available MMC, SD and SDIO cards. These cards are accessed with an SPI interface (ALL cards have this interface as it is standard requirement in the MMC/SD specification). Low level functions are included to read and write memory cards.
Check the 'xgFAT Users Manual' for more information on the X-Graph FAT library with support for MMC/SD memory cards.

Drivers for SDIO cards are very specific and not included. If you need a specific driver, you can contact DELGEN for advice or help. On request DELGEN can also write low level drivers for SDIO cards.

Two MMCSD libraries are included with the X-Graph cards. The MMCSD_SPI.LIB library uses the standard DC SPI.LIB library to access the SPI bus. The data transfer speed is limited by the DC SPI API (appr. 150kByte/sec).
The X-Graph MMCSD.LIB uses optimized assembly functions to read/write the SPI bus. This library offers the max. SPI speed possible with the Rabbit CPU (cpu clock / 8). This results on a 58.8MHz XG5000 in an SDCard read/write speed of 1.8Mbytes/sec.

The XGRAPH.LIB will '#use' the faster MMCSD.LIB by default. If you prefer to use the SPI.LIB based MMCSD_SPI.LIB, you should define the macro MMCSD_USE_SPI_LIB before using the XGRAPH.LIB.

The macro MMCSD_NO_REVERSE enables the fastest possible read/write transfers combined with the MMCSD.LIB. On a Rabbit 3000 CPU, the SPI hardware uses a wrong direction of the shiftregister (wrong compared to 99% of all available SPI devices including MMC/SD cards). Thus each read or written byte needs to be reversed. The software uses a lookup table to do this. But this slows down the MMC/SD functions about 10%. By enabling above macro, the reversing is NOT done anymore. This of course, requires that all data on the MMC/SD card is stored reversed. If the card is only used as an internal memory device, the order of the bits is not important, and the macro can be enabled always. If the memory card needs to be read by a PC, the PC application should inverse all bytes (only when the macro is enabled). The macro is not compatible with any FAT function ('xgFAT Users Manual') as the PC FAT functions of course don't do the reversing.

By including the DEBUG_MMCSD_PRINTF macro a lot of debug information is printed to the debug window. This is usefull while debugging SDCard functions.

void sdReset(void);
Power cycle the SDCard. This involves switching of the power to the SDCard but also switching off the SPI bus to read/write the SDCard. Because the SPI bus used to access the SDCard is also used to control several other functions on the X-Graph module, the SPI bus can not be disabled continuously. The SDCard must be powered always. To reset an SDCard, the spec states you have to powercycle the card. This can be done with this function without blocking then X-Graph modules system SPI bus.

PARAMETER
None

RETURN VALUE
None

int sdCardInserted(void);
This function returns the status of the SD card. Hot-swapping MMC/SD cards is not electrically supported on the X-Graph board.
On the XG5000/XG4xxx this function always returns 'SD card inserted'. The function is available for backwards compatibility with the XG1000 and XG3000.
The card inserted switch has proofed to work unreliable with most cards, reason to remove support for this function.
A software polling algorithm should be used to detect card presence.

PARAMETER
None

RETURN VALUE
0 = SD card not inserted
1 = SD card inserted

int sdWriteProtected(void);
This function returns the status of SD card write protect switch. The result is only valid if a SD card is inserted. MMC cards do not have a write-protect switch and always read not-write protected.
Due to unreliable operation, the XG5000/XG4xxx will always report 'SD card not write protected'. The function is available for backwards compatibility with the XG1000 and XG3000.

PARAMETER
None

RETURN VALUE
0 = SD card not write protected
1 = SD card write protected

int sdInit(void);
Initialize the MMC/SD card:
1. Reset the card
2. Initialize the card (wait for finished, this might take 500msec)
3. Set block length to 512 bytes
If one of these steps fails, the card should not be used.

The current library release only supports cards with a 512 byte block length. If the card does not accept this block length an error is reported (by step 3). Most MMC/SD memory cards use a 512 byte block length.

PARAMETER
None

RETURN VALUE
0: card detected and accepted all commands, ready for use
-1: card initialization failed
-2: timeout while waiting for card initialization
-3: card reported error while reading OCR register
-4: write block size does not equal 512 bytes
-5: read block size larger then 512 bytes
-6: card reported error while reading CSD register

*int sdWriteSector(unsigned long iSectorNr, char *iSDBuffer);*
Write a sector from the root buffer to the mmc/sd card.

PARAMETERS
iSectorNr: sector number, 0 is lowest sector on media
iSDBuffer: pointer to a 512 byte root buffer holding data to be written to the card

RETURN VALUE
0: buffer written
-1: timeout while waiting for Card Response on iCmd
-2: timeout while waiting for busy end (R1b response)
-3: CRC or Write error reported after write command
-4: timeout while waiting for busy end after write command
-5: timeout while waiting for datablock ready after read command
1->255: Card Response error (R1 format, see SDCard spec)
256->511: Data error token from block read operation (see SDCard spec)

int sdWriteSectors(unsigned long iSectorNr, unsigned int iSectorCtr, long iSDBuffer);
Writes the sectors from the xmem buffer to the mmc/sd card. For MMC compatibility single blocks are written (multiple block write is only supported by SD cards).

PARAMETERS
iSectorNr: sector number, 0 is lowest sector on media
iSectorCtr: number of sectors to write, minimum is 1
iSDBuffer: pointer to xmem buffer holding data to be written to the card

RETURN VALUE
0: buffer written
-1: timeout while waiting for Card Response on iCmd
-2: timeout while waiting for busy end (R1b response)
-3: CRC or Write error reported after write command
-4: timeout while waiting for busy end after write command
-5: timeout while waiting for datablock ready after read command
1->255: Card Response error (R1 format, see SDCard spec)
256->511: Data error token from block read operation (see SDCard spec)

*int sdReadSector(unsigned long iSectorNr, char *iSDBuffer);*
Read a sector from the mmc/sd card to a root buffer.

PARAMETERS
iSectorNr: sector number, 0 is lowest sector on media
iSDBuffer: pointer to a 512 byte root buffer

RETURN VALUE
0: buffer read
-1: timeout while waiting for Card Response on iCmd
-2: timeout while waiting for busy end (R1b response)
-3: CRC or Write error reported after write command
-4: timeout while waiting for busy end after write command
-5: timeout while waiting for datablock ready after read command

1->255: Card Response error (R1 format, see SDCard spec)
256->511: Data error token from block read operation (see SDCard spec)

int sdReadSectors(unsigned long iSectorNr, unsigned int iSectorCtr, long iSDBuffer);
Read the requested sectors from the sd card to the xmem buffer. This function uses the multiple block read command, only supported on SD Cards. Don't use this function with MMC cards.

PARAMETERS
iSectorNr: sector number, 0 is lowest sector on media
iSectorCtr: number of sectors to write, minimum is 1. The total buffer length can not exceed 64kByte.
iSDBuffer: pointer to xmem buffer to store read data, buffer must be large enough, no checking

RETURN VALUE
0: buffer read
-1: timeout while waiting for Card Response on iCmd
-2: timeout while waiting for busy end (R1b response)
-3: CRC or Write error reported after write command
-4: timeout while waiting for busy end after write command
-5: timeout while waiting for datablock ready after read command
1->255: Card Response error (R1 format, see SDCard spec)
256->511: Data error token from block read operation (see SDCard spec)

int sdReadBytes(unsigned long iSectorNr, unsigned int iBytes, long iSDBuffer);
Read iBytes bytes from the mmc/sd card to the xmem buffer. This function uses the multiple block read command, only supported on SD Cards. Don't use this function with MMC cards.

PARAMETERS
iSectorNr: sector number, 0 is lowest sector on media
iBytes: number of bytes to read (max. 64kByte)
iSDBuffer: pointer to xmem buffer to store read data, buffer must be large enough, no checking

RETURN VALUE
0: buffer read
-1: timeout while waiting for Card Response on iCmd
-2: timeout while waiting for busy end (R1b response)
-3: CRC or Write error reported after write command
-4: timeout while waiting for busy end after write command
-5: timeout while waiting for datablock ready after read command
1->255: Card Response error (R1 format, see SDCard spec)
256->511: Data error token from block read operation (see SDCard spec)

int sdReadGraphic(unsigned long iSectorNr, unsigned int iBytes, unsigned int iLines, unsigned long iSDBuffer, unsigned int iWidth);
Read iLines of iBytes from the MMC/SD card to an xmem video buffer. This function can be used to directly copy a bitmap file on the MMC/SD card to the video buffer.

PARAMETERS
iSectorNr: sector number, 0 is lowest sector on media
iBytes: number of bytes to read for each graphic line (512 bytes max.)
iLines: number of lines (= repeat loops) to copy
iSDBuffer: pointer to xmem buffer to store read data (video buffer)
iWidth: number of bytes in each graphic buffer line

RETURN VALUE
0: buffer read
-1: timeout while waiting for Card Response on iCmd
-2: timeout while waiting for busy end (R1b response)
-3: CRC or Write error reported after write command
-4: timeout while waiting for busy end after write command
-5: timeout while waiting for datablock ready after read command
1->255: Card Response error (R1 format, see SDCard spec)
256->511: Data error token from block read operation (see SDCard spec)

7.2.4 Buzzer

void buzzer(int onOff);
Turn on or off the X-graph buzzer

PARAMETER
0 = Buzzer Off
1 = Buzzer On

RETURN VALUE
None

7.2.5 High Voltage Outputs

int SetHighVoltOutput(char iLine, char iValue, unsigned int iPWM);
Control the high voltage/high current-sinking outputs. Each line can be set on or off, some lines can use a PWM channel.
Note that the PWM speed is set to maximum by the xgraph_init() function (X-Graph default).

PARAMETER
iLine: 0->4: select high voltage output (HVOUT0 -> HVOUT4)
iValue:
0 = switch line to high impedance
1 = switch line to current-sinking
2 = switch line to pwm output mode (if support by output line)
iPWM: 10-bit PWM value

RETURN VALUE
0: ok
-1: invalid high voltage output line
-2: invalid value
-3: PWM selected for line that does not support PWM
-4: PWM value out of range

7.2.6 High Current Outputs

void SetHighCurrentOutput(char iLine, char iValue);
Changes state of 2Amp High Current output.

PARAMETER
iLine: 0->7: select high current output
iValue:
0 = output off
1 = output is sinking
2 = output is sourcing

RETURN VALUE
None

7.2.7 DAC

int SetDAC(unsigned int iValue);
Puts value to output of DAC.

PARAMETER
0->4095
For 8-bit DAC versions the 4 LSB's must be set to 0.

RETURN VALUE
0: ok
-1: invalid value

7.2.8 ADC

*int GetADC(char iChannel, unsigned int *iValue);*
Read an ADC channel.
Refer to the datasheet of the ADS7844/ADS8344 for additional information.

If 16-bit ADC chips are installed on your board, the corresponding macro should be enabled in the jumpstart application (XG_ADC_16BIT).

The ADC_REF_VOLTAGE macro sets the reference voltage of the ADC chips in mVolt. If not predefined, the macro is automatically set to 3300 on the XG4xxx and 5000 on the XG1000/XG5000. On the X-Graph modules the ADC chips reference pins are connected to the analog power supply which is 3.3Volt on the XG4xxx and 5Volt on the XG1000/XG5000. On the XG5000 the ADC power supply can be set to 3.3Volt (factory option). This macro should be used to set the correct voltage when this option is used.

The ADC_AVERAGE macro is set to 1 in the XGRAPH.LIB library. This disables the averaging of the ADC results. It can be set to any value prior to using the XGRAPH.LIB. When ADC_AVERAGE is, for example, set to 4, each call to GetADC will result in 4 consecutive reads of the chosen ADC channel. Then the average of the 4 reads is reported.

PARAMETER
iChannel:
0->7: select one of the 8 input channels in single mode (ref is ground)
8->15: select one of the 8 differential input channels (left = + input, right = - input)
8: +CH0 –CH1
9: +CH2 –CH3
10: +CH4 –CH5
11: +CH6 –CH7
12: +CH1 –CH0
13: +CH3 –CH2
14: +CH5 –CH4
15: +CH7 –CH6
16->23: idem for second ADC chip
24->31: idem differential for second ADC chip

RETURN VALUE
0: ok
-1: invalid channel

7.2.9 X-Graph I/O Lines

void BitWrPortX(int io_line, int value);
Updates an X-Graph I/O line with value (0 or 1 (or 2)).

PARAMETER
io_line: Name of X-Graph I/O Line
#define XGIO_HVOUT0
#define XGIO_BUZZER
#define XGIO_HVOUT1
#define XGIO_HVOUT2
#define XGIO_HVOUT3
#define XGIO_HVOUT4
#define XGIO_HVOUT7
#define XGIO_HVOUT8
#define XGIO_HVOUT9
#define XGIO_HVOUT10
#define XGIO_HVOUT11
#define XGIO_HVOUT12
#define XGIO_HVOUT13
#define XGIO_HVOUT14
#define XGIO_HCOUT0
#define XGIO_HCOUT1
#define XGIO_HCOUT2
#define XGIO_HCOUT3
#define XGIO_HCOUT4
#define XGIO_HCOUT5
#define XGIO_HCOUT6
#define XGIO_HCOUT7
value:
0 = low level or off
1 = high level or sinking
2 = sourcing (only available for HighCurrent outputs)
RETURN VALUE
None

int BitRdPortX(int io_line;
Reads an X-Graph I/O line

PARAMETER
io_line: Name of X-Graph I/O Line
#define XGIO_BIN0
#define XGIO_BIN1
#define XGIO_BIN2
#define XGIO_BIN3
#define XGIO_BIN4
#define XGIO_BIN5
#define XGIO_BIN6
#define XGIO_BIN7

RETURN VALUE
Read value

7.2.10 Slave processor ADC

The X-Graph module can have a 8 channel 10-bit or 12-bit ADC. This ADC has advanced features. Full documentation can be found in the ADC section of the Atmel AVR ATmega8/ATmega88/ATxmega16A4 microcontroller specification. To fully understand all

the features and get the best possible AD results, we suggest you read this document carefully before using the X-Graph ADC.

void SlaveADCStart(char iChannel);
(only for ATMega8/ATMega88 versions)
Returns the analog digital converter value of one of the slave's analog inputs.
NOTE: the pin must be configured as input.
NOTE: when a touchscreen is installed, two inputs are used to control the touchscreen.

PARAMETER
ADC channel number (0 -> 7)
RETURN VALUE
Event: Read data (10-bit value)

void SlaveADCConfig(char iChannel, char iRefVolt, char iSingleDiff;
(only for Atxmega versions)
Sets the channel the slave ADC continuously monitors

PARAMETER1:
Channel number (single ended input or positive input for differential input)
SADC_CHANNEL_0 = TAIN0 input
...
SADC_CHANNEL_7 = TAIN7 input
For differential inputs the negative input selection must be added:
SADC_CHANNEL_NEG_0 = TAIN0 input
...
SADC_CHANNEL_NEG_7 = TAIN7 input

Example: i = SADC_CHANNEL_0 + SADC_CHANNEL_NEG_4;
This selects TAIN0 as positive input and TAIN4 as negative input.
For single ended measurements, the negative input is ignored.
It's important to understand the negative input selection depends on
the type of differential input (with or without gain), see parameter 3

PARAMETER2:
Reference voltage:
SADC_REF_1V = 1Volt accurate reference voltage
SADC_REF_2V = 2.0625Volt reference based on 3.3Volt power supply
SADC_REF_USER = TAIN0 (J41-0) pin used as reference voltage (range is 20V to 54V)

PARAMETER3:
Single / Diffential / Differential with Gain
SADC_SINGLE_12 = Single ended input and 12-bit resolution with range 18.9V to 0.97V (1V reference)
SADC_SINGLE_11 = Single ended input and 11-bit resolution with range 19.875V to 0V (1V ref)
SADC_DIFF = Differential input / 11-bit+sign resolution / + inputs (TAIN0->TAIN7) / - inputs (TAIN0->TAIN3)
SADC_DIFF_GAIN_1 = Differential input / 11-bit+sign resolution / + inputs (TAIN0->TAIN7) / - inputs (TAIN4->TAIN7)
SADC_DIFF_GAIN_2 = same with gain 2x
SADC_DIFF_GAIN_4
SADC_DIFF_GAIN_8
SADC_DIFF_GAIN_16
SADC_DIFF_GAIN_32
SADC_DIFF_GAIN_64 = ... same with gain 64x

RETURN VALUE
None

int SlaveADCRead(void);
(only for Atxmega versions)
Reads the result of the slave ADC (only for HWVersion > 3 slaves)

PARAMETER
None

RETURN VALUE
12-bit value read from ADC
Read value

7.2.11 Slave processor DAC

(only for Atxmega versions)

void SlaveDACWrite(char iChannel, int iValue);
Writes a value to a DAC channel

PARAMETER1:
Channel number (0->1)
PARAMETER2:
Value (0->4095 range)

RETURN VALUE:
None

int SlaveDACSetmA(char iChannel, int imA);
Sets a DAC channel in a 4mA-20mA range
Each Slave DAC channel has a 0V to 1V output range.
A UI convertor converts this to a 3.282mA to 21.615mA range.
Due to tolerances on the reference voltages and external resistor networks, it's advised to calibrate the DAC outputs per your requirements.

PARAMETER1:
Channel number (0->1)
PARAMETER2:
Value (4000 -> 20000 range = 4mA till 20 mA)

RETURN VALUE:
0 = ok
-1 = out of range mA

7.2.12 Slave I/O Ports

void BitWrSlave(char iPort, char bValue, char iLine);
Writes a bit to a slave I/O port or configures the port as output, input or open-collector.

PARAMETER
iPort:
0 = configure the direction of the selected line
1 = configure the level of the selected line
bValue (for direction setting)
0 = direction is input
1 = direction is output
bValue (for level setting)
0 = low output
1 = high output

NOTE: writing a high output bit to an 'input' line, enables a pull-up resistor on this line. With this system an open-collector output can be made.
iLine:
1: AIN2
2: AIN3
3: AIN4
4: AIN5
5: AIN6
6: AIN7
7: UART TXD
8: UART RXD
9: 1-Wire
10: RC-5
11: ATS Up
12: ATS Right
13: xgBus OE
14: xgBus GP0
15: xgBus GP1
16: ZigBee CTS
17: ZigBee RTS
Do not use settings 11 and 12, they are used by the touchscreen driver.

RETURN VALUE
None

char BitRdSlave(char iPort, char iLine);
Reads a bit from a slave I/O port.

PARAMETER
iPort:
0 = read input port
1 = read output port (bit read from output port setting)
This can be used to read the set state of an output line. This is not always the same as the state of the input line, because an output line can be electrically shorted (only valid if it is configured as an open-collector line).
iLine:
1: AIN2
2: AIN3
3: AIN4
4: AIN5
5: AIN6
6: AIN7
7: UART TXD
8: UART RXD
9: 1-Wire
10: RC-5
11: ATS Up
12: ATS Right
13: xgBus OE
14: xgBus GP0
15: xgBus GP1
16: ZigBee CTS
17: ZigBee RTS
Do not use settings 11 and 12, they are used by the touchscreen driver.

RETURN VALUE
0 = line is low
1 = line is high

7.2.13 1-Wire

The slave processor uses a software protocol to communicate with 1-Wire devices. Due to the strict timing requirements of this protocol it is advised to not use any other features controlled by the slave processor during 1-Wire activity.

int oneWireInit(void);
Initializes the 1-Wire bus. This function must be called before using any other 1-Wire command.

This function will activate a 1-wire blocking function in the slave processor. Doing a 1-Wire init might take several 100msec up to seconds.
During this time the slave processor will be blocked, i.e.:
- It will NOT accept any Rabbit commands (timeout in XGSLAVE.LIB for each command is 50msec)
- It will NOT process any other local interface. I.e. if active, touchscreen, keypad, PC/AT, RC-5, ... inputs are not detected and lost.
If this is not acceptable in your application, DONT use the OneWireInit() function. In stead use the OneWireRead()/OneWireWrite() and BitRd/WrSlave() functions to build a 1-wire init function. Contact DELGEN if you need help with this.

PARAMETER
None

RETURN VALUE
0 = command accepted
-1 = 1-Wire is busy executing another command, this command is rejected
Event: number of devices on the bus

int oneWireSearch(char iSearchCmd);
Searches the 1-Wire bus for devices and stores the found 1-Wire ID's in a buffer. For 1-Wire busses with more then 1 device connected this function must be called before any read or write 1-Wire function.

This function will activate a 1-wire blocking function in the slave processor. Doing a 1-Wire init might take several 100msec up to seconds.
During this time the slave processor will be blocked, i.e.:
- It will NOT accept any Rabbit commands (timeout in XGSLAVE.LIB for each command is 50msec)
- It will NOT process any other local interface. I.e. if active, touchscreen, keypad, PC/AT, RC-5, ... inputs are not detected and lost.
If this is not acceptable in your application, DONT use the OneWireSearch() function. In stead use the OneWireRead()/OneWireWrite() and BitRd/WrSlave() functions to build a 1-wire init function. Contact DELGEN if you need help with this.

IMPORTANT: This function requires for each 1-Wire device a 9 byte buffer. The 128 byte buffer allows for maximum 14 devices. If your 1-Wire setup has more devices, do NOT use this function as this will overflow the buffer and possible corrupt the slave processors stack. This will result in unpredictable behavior which might damage your X-Graph SBC and void your warranty.
A search function supporting more then 14 devices can always be build with the standard OneWireRead/Write functions. By using an R3K buffer, this setup can handle any number of devices on the 1-Wire bus (such a function is NOT included in this library, contact DELGEN is you need help designing such a function).

PARAMETER
iSearchCmd = 0xf0: search ROM
iSearchCmd = 0xec: alarm search

Other values might be used dependant on the 1-Wire devices used. See the 1-Wire device specification.

RETURN VALUE
0 = command accepted
-1 = 1-Wire is busy executing another command, this command is rejected
Event: number of devices on the bus and a the ID's of the found devices. Each device ID requires 9 bytes. The first 8 bytes of each ID contain the ROM code of the found device and the 9th byte contains the status bit (not available on all 1-Wire devices).

*int oneWireWrite(char *iData, char iNr, char iType);*
Writes a string to the 1-Wire bus.

A parameter is available to enable a strong pull-up after the write.
This is needed for some 1-Wire devices if no 1-Wire power line is available. Check the datasheets of the 1-Wire devices you want to use.
Note that to release the strong-pullup the OneWireInit() function must be called or the BitRd/WrSlave() functions can be used.

PARAMETER
iData = pointer to buffer containing the data to write to the 1-Wire bus
iNr = number of bytes to write to 1-Wire bus
iType:
0 = normal write
1 = enable strong pullup after write

RETURN VALUE
0 = command accepted
-1 = 1-Wire is busy executing another command, this command is rejected

int oneWireRead (char iNr);
Reads a string from the 1-Wire bus.

PARAMETER
iNr = number of bytes to read from the 1-Wire bus

RETURN VALUE
0 = command accepted
-1 = 1-Wire is busy executing another command, this command is rejected
Event: read data

NOTE: Reading 1 byte takes about 2 msec. The state byte read has a 20msec timeout.

Example to read/write a Dallas 18B20 temperature sensor:

```
OneWireInit();
iData[0] = 0xcc;                    // Skip ROM (only one device on the bus)
iData[1] = 0xbe;                    // Reads scratchpad
OneWireWrite(iData, 2, 0);
OneWireRead(iData, 9);
```

7.2.14 Eeprom

The slave processor includes a 512 or 1024 byte eeprom array which can be randomly written or read. Each byte write requires maximum 10 msec.

char EepromWrite(unsigned int iAddress, char iData);
Writes a byte to the slave eeprom array.

PARAMETER
iAddress: address in the eeprom (0 up to 511/1023)
iData: data to be written

RETURN VALUE
0 = byte written
-1 = error, byte not written
-2 = address out of range

*int EepromWriteBlock(unsigned int iAddress, char *iData, unsigned int iLength);*
Write a block of data to the slave eeprom array. This function is a wrap-around function of EepromWrite() to make it easier to access structs and strings stored in the eeprom array.

Note: as a single byte write takes about 10 msec, writing long blocks of data will take a considerable amount of time. This function just waits until all bytes are written. If that is not acceptable, you should use an event driven function (not available in this lib).

PARAMETER
iAddress: address in the eeprom (0 up to 511/1023)
iData: pointer to a datablock
iLength: nr of bytes to write

RETURN VALUE
0 = byte written
-1 = error, byte not written
-2: block length does not fit address range

int EepromRead(unsigned int iAddress);
Reads a byte from the slave eeprom array.

PARAMETER
address in the eeprom (0 up to 511/1023)

RETURN VALUE
-1: read error
-2: address out of range
other value: read byte

*int EepromReadBlock(unsigned int iAddress, char *iData, unsigned int iLength);*
Reads a block of data from the slave eeprom array. This function is a wrap-around function of EepromRead() to make it easier to access structs and strings stored in the eeprom array.

PARAMETER
iAddress: address in the eeprom (0 up to 511/1023)
iData: pointer to a datablock
iLength: nr of bytes to read

RETURN VALUE
0: read ok
-1: read error
-2: block length does not fit address range

int EepromErase(void);
Erase the complete eeprom = fills with 0x00.
Execution time is about 5 seconds.

PARAMETER
None

RETURN VALUE
0: write ok
-1: write error

7.2.15 Slave Flash Storage

The slave processor includes a 1024 or 4096 byte flash array which can be randomly written or read. Note that 64 byte flash pages are used, but the slave processor automatically handles this. Use this memory to store settings which don't change very often. There is a limited number of write cycles.

*int FlashWrite(unsigned int iAddress, char *iData, char iLength);*
Writes data to the slave flash array.

Notes on slave flash writes:
- The 1kByte flash array consists out of sixteen 64-byte pages
- The 4kByte flash array consists out of sixteen 256-byte pages
- A write requires appr. 10 msec and the slave cpu will be locked during flash writes (also no interrupts). This will stop all event handling (RC-5, keypad, keyboard, …).

Use this function with care.

The slave cpu will always write a full 64/256-byte page, even if a single byte needs to be written. The flash has a limited rewrite capability (10.000 times). If structs or long strings need to be written to the Flash, it's better to group these in 64-/256-byte pages and do 64-/256-byte group writes.

IMPORTANT: this routine does only accept writes which fit a single 64-/256-byte page. Check also the FlashWriteBlock() function.

PARAMETER
iAddress: address in the flash (0 up to 1023/4095)
iData: pointer to string of data
iLength: length of string

RETURN VALUE
0 = byte written
-1 = address out of range
-2 = length > 64
-3 = page border crossing

*int FlashWriteBlock(unsigned int iAddress, char *iData, char iLength);*
Writes a block of data not 64-byte page aligned to the slave flash array.

As a single page write takes about 10 msec, writing long blocks of data will take up to 160 msec of time. This function just waits until all pages are written. If that is not acceptable, you should use an event driven function (not available in this lib)

PARAMETER
iAddress: address in the flash (0 up to 1023/4095)
iData: pointer to string of data
iLength: length of string

RETURN VALUE
0 = byte written

-1 = block does not fit flash array

int FlashRead(unsigned int iAddress);
Reads a byte from the slave flash array.

PARAMETER
address in the flash (0 up to 1023/4095)

RETURN VALUE
-1 = address out of range
any other value = read byte

*int FlashReadBlock(unsigned int iAddress, char *iData, unsigned int iLength);*
Reads a block of data from the slave flash array.

PARAMETER
iAddress: Start address (0 up to 1023/4095)
iData: pointer to datablock
iLength = number of bytes to read

RETURN VALUE
0 = read ok
-1 = block length does not fit address range

int FlashErase(void);
Erase the complete flash = fills with 0xff.
Execution time is about 160 msec.

PARAMETER
None

RETURN VALUE
0 = write ok

7.2.16 Slave SRAM Storage

The slave processor includes a 512 or 1024 byte sram array which can be randomly written or read. Take care with this function, the sram array is used by the slave processor to handle event buffers for the I2C, UART and 1-Wire.
Note: it's impossible to write/read the critical SRAM data of the slave processor with this function, i.e. stack or system variables.

The sram array mapping (for 512 byte version only):
0->127: I2C buffer
128->255: RS232C transmit buffer
256->383: dual-buffered (2x64 bytes) RS232C receive buffer
384->511: 1-Wire Buffer

ONLY USE THIS FUNCTION IF REALLY NEEDED.
Prefer the dedicated functions to access the I2C, RS232C and 1-Wire buffers.
If none of these features are used, the SRAM buffer, or part of it, can be used to store temp. data. The buffer is not battery backed up.

int SRAMWrite(unsigned int iAddress, char iData);
Writes a byte to the slave sram array.

PARAMETER
iAddress: address in the sram (0 up to 511/1023)
iData: data to be written

RETURN VALUE
0 = byte written
-1 = address out of range

*int SRAMWriteBlock(unsigned int iAddress, char *iData, unsigned iLength);*
Writes a block of data to the slave sram array.

PARAMETER
iAddress: address in the sram (0 up to 511/1023)
iData: pointer to the data block
iLength: number of bytes to write

RETURN VALUE
0: write ok
-1: block length does not fit address range

int SRAMRead(unsigned int iAddress);
Reads a byte from the slave sram array.

PARAMETER
address in the sram (0 up to 511/1023)

RETURN VALUE
-1: address out of range
any other value: read byte

*int SRAMReadBlock(unsigned int iAddress, char *iData, unsigned int iLength);*
Reads a block of data from the slave sram array.

PARAMETER
iAddress: address in the sram (0 up to 511/1023)
iData: pointer to the data block
iLength: number of bytes to read

RETURN VALUE
0: read ok
-1: block length does not fit address range

7.2.17 Slave UART

The slave processor has a fully configurable UART.

void SlaveUARTBaudrate(unsigned long iBaudrate);
Sets the slave UART baudrate.

PARAMETER
Any baudrate, but the actual set baudrate will be rounded.

The slave processor is running on a build-in RC oscillator, factory calibrated. On the X-Graph module it's recalibrated by the Rabbit to appr. 7.3728MHz wich results in nearly perfect dividers for baudrates up to 921600 baud.
Due to temperature variations the actual baudrate might vary 4% with temperatures set to -25deg or +80deg.

RETURN VALUE
None

void SlaveUARTConfig(char iParity, char iStopBits, char iCharacterSize, char iMulti, char iBuffer);
Configures the slave UART.

Refer to the Atmel ATMega8 documentation for more information on this serial port.

PARAMETER
iParity:
0: hardware parity disabled
1: even parity (hardware generated)
2: odd parity (hardware generated)
iStopBits:
1 = 1 stop bit
2 = 2 stop bits
iCharacterSize:
5->9: number of bits in the character size
iMulti:
0 = multiprocessor mode disabled
1 = multiprocessor mode enabled
iType:
0 = single byte buffer
1 = dual byte buffer (stores error flags and 9-bit in 9-bit character size)

RETURN VALUE
None

*int SlaveUARTTransmit(char *iData, char iLength);*
Transmits a string (binary or ASCII) of data via the slave UART.

PARAMETER
iData: Pointer to start of string
iLength: length of string (max. 126 bytes)

RETURN VALUE
0: transmit started
-1: transmitter is still busy with previous data, data not transmitted
-2: length > 126 bytes

void SlaveUARTreceive(void);
Normally data received via the UART RS232C input port is tranmitted in 64-byte chunks to the Rabbit 3000A. The slave also reports (with an event) once a single byte has been stored in the receive buffer.

This command can be used to force a transmit of the receive buffer.

PARAMETER
None

RETURN VALUE
None

7.2.18 I2C

The X-Graph module slave processor has a build-in hardware I2C module. Functions are available to transmit and receive data to and from the I2C bus. Additional multi-master functions can be made available on simple request.

*int I2CTxdRxd(char *iData, char iTxdLength, char iRxdLength);*
Transmits and/or receives data on the I2C bus.

Received data is available via the Event system.

PARAMETER
iData: pointer to string to be transmitted. Received data is only available after event warnings from the slave.
iTxdLength: nr of bytes to transmit (min. = 1 byte)
iRxdLength: nr of bytes to receive (min. = 0)

RETURN
0 = I2C communication ok
-1 = transmit length must be larger then 0
-2 = transmit length can not be larger then 125 bytes
-3 = slave still busy with pending I2C transmits, no new transmit can be started.

void I2CSpeed(unsigned int iSpeed);
Set the I2C communication speed. Most devices support a 400kHz clock (the default value). But the I2C clock speed can be reduced to 100kHz for backwards compatibility with older I2C devices.

Note: A limitation of the slave processor might result in errors for bitrates > 200kHz only with some devices and in master mode.
Note: The slowest speed which can be set with this function is 14kHz

PARAMETER
Busspeed in kHz (the spec does not allow speeds > 400kHz)

RETURN VALUE
None

void I2CReset(void);
Resets the slave I2C hardware and the I2C event system.

The slave I2C hardware module will lock-up if on pull-up resistors are installed on the I2C slave processor I/O lines. Note that this is a I2C specification requirement.
The slave processor I2C functions and the X-Graph event system can be reset once a fatal I2C error is detected (using a Rabbit 3000A timeout).

PARAMETER
None

RETURN VALUE
None

7.2.19 PC/AT

The PC/AT keyboard is scanned automatically and scancodes are pushed in the event queue and handled by the ui_event() function in the xgraph_ui.c library.

The current firmware release does not support commands to be send to the keyboard. Thus it's not possible to change the keyboard repetition rate or set/reset the keyboard leds.

7.2.20 RC-5 Receive

WARNING: the current slave firmware release does not yet support this function. Contact DELGEN for a release date if you need this function.

Received RC-5 codes are pushed in the event queue and handled by the ui_event() function in the xgraph_ui.c library.

7.2.21 Analog Touchscreen

Touchscreens events are automatically pushed in the event queue and handled by the ui_event() function in xgraph_ui.c. See paragraph 6.6 for more information.

void ATSSpeed(unsigned int iSpeed);
The standard scan speed of the touchscreen is set to 8msec. You can make this run slower if your application needs this. Note that 8msec is a generally accepted reaction speed for touchscreen operation.

PARAMETER
Speed in msec (8 msec resolution)

RETURN VALUE
None

7.2.22 Keypad

The slave processor I/O lines can be used to scan a keypad without the need for pullup resistors.

Keypad events are automatically pushed in the event queue and handled by the ui_event() function in xgraph_ui.c. See paragraph 6.6 for more information.

7.2.23 Slave Firmware Upgrade

The slave firmware can be upgraded. Use this function with the most care.
A sample application (SFU.C) is included in the samples directory. If possible only use this application to upgrade the slave firmware.

A life application could at reset check the firmware version number and then decide to upgrade the slave firmware. The slave firmware version is available in the global variable iSlaveSWVersion.

Before doing a slave firmware upgrade also the global variable iSlaveHWVersion should be checked. It MUST match the hardware version number of the ximported firmware.

NOTE: A compatible hex file must be #ximporte'd. Use:
#ximport XG_SLAVE_FIRMWARE ….. (location of .hex file)

int SlaveUpgrade(void);
Upgrades the firmware of the slave processor. All functionality of the slave processor will be halted. After the upgrade the board will reset.

PARAMETER
None

RETURN VALUE
None

7.3 X-Graph Event Handling

7.3.1 X-Graph Event Definition

The X-Graph modules do not only support a GUI touchscreen, but also support a keypad, a PC/AT compatible keyboard interface, a remote input and several other user input interfaces.

A typical X-Graph user application will use one or more of these interfaces to gather user input and react on these inputs, i.e.: send data on interfaces or build graphic LCD GUI elements.

The X-Graph libraries include a very simple but effective event handler. It has a very limited program overhead and can be used for most X-Graph user interfaces. Basically the X-Graph hardware drivers capture hardware events and translate these in a standard X-Graph event format.

In a standard X-Graph application the user interface will periodically call the xgraph_tick() function. This function handles are X-Graph low-level hardware drivers. Once an event is detected, the function ui_event() will be called. This function is a part of the user interface application. It's different for each application and should be written by the user interface software designer (see the included examples).

The ui_event() function should filter the events and call the appropiate user interface functions.

So, the mainloop of your program calls periodically xgraph_init(). This function calls ui_event() which need to handle all event driver actions of the application.

All standard X-Graph events are defined in the XGEVENT.LIB library. You can easily add your own user interface based events.

An example of a basic keypad event handler:

```
void ui_event(unsigned int iEvent, unsigned int iX, unsigned int iY, void *sData) {
  switch (iEvent) {
    case EVENT_KEYPAD_PRESSED:
       printf("Key %u pressed\r\n", iX);
       break;
    case EVENT_KEYPAD_RELEASED:
       printf("Key released\r\n");
       break;
  }
}
```

7.3.2 X-Graph Events

EVENT_ATS_PRESSED
The very first time the touchscreen is pressed
iX = x coordinate of touch
iY = y coordinate of touch

EVENT_ATS_COORD_CHANGED
Each time the touchscreen coordinates change
iX = x coordinate of touch
iY = y coordinate of touch

EVENT_ATS_RELEASED
The touchscreen is released
iX = last known x coordinate
iY = last known y coordinate

EVENT_PCAT
PC/AT interface scancode
iX = scancode

EVENT_RC5_PRESSED
RC-5 key pressed
iX = address (bit 7 is toggle bit)
iY = command

EVENT_RC5_RELEASED
RC-5 key released

EVENT_KEYPAD_PRESSED
A key on the keypad is pressed
iX = key number

EVENT_KEYPAD_RELEASED
iX = key released

EVENT_I2C_READY
The current I2C command is completed

EVENT_I2C_ERROR
The current I2C command is aborted due to a I2C error

EVENT_I2C_RXD_READY
I2C data has been receied
iX = number of bytes received
sData = pointer to data

EVENT_UART_READY
RS232C transmission is finished

EVENT_UART_RXD
At least one byte is received in the UART receive buffer

EVENT_UART_RXD_BUFFER
UART received data is available
iX = number of bytes received
sData = pointer to data

EVENT_1WIRE_RXD_READY
1-Wire received data is available
iX = number of bytes received
sData = pointer to data

EVENT_1WIRE_READY
The 1-Wire interface is available, the previous command has completed.

EVENT_ADC
Read back value of ADC
iX = read data

7.4 X-Graph Special Libraries

7.4.1 Screendump

This library makes a screendump of the current LCD screen in BMP format on an FTP server. It can be used for making bmp files to be used in a product manual.
To use this library, Ethernet support must be active and you must have a running FTP server on your pc which accepts uploads.

The following defines are preset but can be changed to your FTP servers settings:
XGFTP_SERVER "192.168.1.102"
XGFTP_USERNAME "Anonymous"
XGFTP_PASSWORD ""
XGFTP_DIR "FTP"

Note: Currently this library only supports the B/W screen. Contact DELGEN if you need support for other LCD's.

void ScreenDump(void);
This dumps the LCD screen data to a file on an FTP server.

PARAMETER
None

RETURN VALUE
None

void screendump_tick(void);
Call this function regularly, it handles the FTP protocol.

PARAMETER
None

RETURN VALUE
None

7.4.2 Character LCD

The CHARLCD.LIB library offers support to initialize a character LCD and print some data on them.

There are a number of defines to configure the I/O ports used by the character LCD. Such an LCD needs an 8-bit databus and 3 control lines (RW, RS, ENABLE). The library is preconfigured for the X-Graph module connection, but you can change these settings if you want to use the library with other I/O ports. Check the lib for the correct defines.

A 2 line by 8 character LCD requires some special attention, the HW_CD_IFSP macro MUST be defined before using the CHARLCD.LIB library.

void clcdInit(void);
Initializes the character LCD.

PARAMETER
None

RETURN VALUE

None

void clcdBacklight(char iCommand);
Switches the backlight of the character LCD on or off.

PARAMETER
1 = backlight on
0 = backlight off

RETURN VALUE
None

*void clcdDispString(char iColumn, char iRow, char iLength, char *iDisplayBuffer);*
Puts a string in the character LCD display buffer.

PARAMETER
iColumn: column of first character (starting with column 0)
iRow: row (starting with row 0)
iLength: length of string
iDisplayBuffer: pointer to string

RETURN VALUE
None

7.4.3 Compact Flash

The CF library information will be added to the manual, once the XG5000 CF plug-in card is released.

Warranty

DELCOMp warrants that the product delivered hereunder shall conform to the applicable DELCOMp datasheet or mutually agreed upon specifications and shall be free from defects in material and workmanship under normal use and service for a period of 1 year from the applicable date of invoice. Products which are "samples", "design verification units", and/or "prototypes" are sold "AS IS," "WITH ALL FAULTS," and without a warranty.
If, during such warranty period, (1) DELCOMp is notified promptly in writing upon discovery of any defect in the goods, including a detailed description of such defect; (2) such goods are returned to DELCOMp, DDP DELCOMp's facility accompanied by DELCOMp's Returned Material Authorization form; and (3) DELCOMp's examination of such goods discloses to DELCOMp's satisfaction that such goods are defective and such defects are not caused by accident, abuse, misuse, neglect, alteration, improper installation, repair, improper testing, or use contrary to any instructions issued by DELCOMp, DELCOMp shall (at its sole option) either repair, replace, or credit Buyer the purchase price of such goods. No goods may be returned to DELCOMp without DELCOMp's Returned Material Authorization form. Prior to any return of goods by Buyer pursuant to this Section, Buyer shall afford DELCOMp the opportunity to inspect such goods at Buyer's location, and any such goods so inspected shall not be returned to DELCOMp without its prior written consent.
DELCOMp shall return any goods repaired or replaced under this warranty to Buyer transportation prepaid. The performance of this warranty does not extend the warranty period for any goods beyond that period applicable to the goods originally delivered.
The foregoing warranty constitutes DELCOMp's exclusive liability, and the exclusive remedy of buyer, for any breach of any warranty or other nonconformity of the goods covered by this agreement. This warranty is exclusive, and in lieu of all other warranties, express, implied, or statutory, including without limitation any warranties of merchantability or fitness for a particular purpose. The sole and exclusive remedy for any breach of this warranty shall be as expressly provided herein.

Limitation on Liability

Notwithstanding anything to the contrary contained herein, DELCOMp shall not, under any circumstances, be liable to Buyer or any third parties for consequential, incidental, indirect, exemplary, special, or other damages. DELCOMp's total liability shall not exceed the total amount paid by Buyer to DELCOMp hereunder. DELCOMp shall not under any circumstances be liable for excess costs of reprocurement

Notice to Users

DELCOMp PRODUCTS ARE NOT AUTHORIZED FOR USE AS CRITICAL COMPONENTS IN LIFE-SUPPORT DEVICES OR SYSTEMS UNLESS A SPECIFIC WRITTEN AGREEMENT REGARDING SUCH INTENDED USE IS ENTERED INTO BETWEEN THE CUSTOMER AND DELCOMp PRIOR TO USE.

Life-support devices or systems are devices or systems intended for surgical implantation into the body or to sustain life, and whose failure to perform, when properly used in accordance with instructions for use provided in the labeling and user's manual, can be reasonably expected to result in significant injury.

No complex software or hardware system is perfect. Bugs are always present in a system of any size. In order to prevent danger to life or property, it is the responsibility of the system designer to incorporate redundant protective mechanisms appropriate to the risk involved.

All DELCOMp products are 100 percent functionally tested. Additional testing may include visual quality control inspections or mechanical defects analyzer inspections. Specifications are based on characterization of tested sample units rather than testing over temperature and voltage of each unit. DELCOMp products may qualify components to operate within a range of parameters that is different from the manufacturer's recommended range. This strategy is believed to be more economical and effective. Additional testing or burn-in of an individual unit is available by special arrangement.

Software License Agreement

Notice to Users
This is a legal agreement between you (an individual or single entity, referred to hereinafter as "you") and DELCOMp for the computer software product(s) including any accompanying explanatory written materials (the "Software"). BEFORE INSTALLING, COPYING OR OTHERWISE USING THE SOFTWARE, YOU MUST AGREE TO THE TERMS AND CONDITIONS OF THIS AGREEMENT.
If you agree, you are allowed to use the software. If you do not agree with the terms and conditions of this Agreement, you are not allowed to use the software and must destroy all copies of the software.

DELCOMp licenses this software to its customers upon acceptance of all the terms and conditions of this license agreement. Please read the terms carefully before downloading or installing the software.
If you do not accept all the terms, you may not install or use this software, and should contact your sales representative to receive a full refund.
If you have any questions, call +32-475-60.64.33, or write to the DELCOMp office at Technologielaan 3, BE-3001 Leuven, Belgium.

1. Definitions. "Software" means the accompanying computer programs, data compilation(s), and documentation. "You" means the licensee, and are referred to as "You."

2. Term. The term of the license granted herein shall continue until terminated either (a) by You, for your convenience, by written notice to DELCOMp or (b) automatically if a material breach by You is not cured within thirty (30) days of such breach. Immediately upon any termination of this license for any reason, You must return to DELCOMp all copies of the Software.

3. License Grant. You are granted non-exclusive rights to install and use the Software on a single computer only; however, if the Software is permanently installed on the hard disk or other storage device of a computer (other than a network server), and one person uses that computer more than 80% of the time, then that person may also use the Software on a portable or home computer. You may not install the Software on a network or transmit the Software electronically from one computer to another or over a network. You may copy the Software for archival purposes, provided that any copy must contain the original Software's proprietary notices in unaltered form.

4. Restrictions. You may not: (i) rent, lease, sublicense, loan, timeshare, or permit others to use the Software, except as expressly provided above; (ii) modify or translate the Software; (iii) reverse engineer, decompile, or disassemble the Software, except to the extent this restriction is expressly prohibited by applicable law; (iv) except as permitted by Section 5 below, create a derivative work based on the Software or merge the Software with another product; (vi) copy the Software, except that a reasonable number of copies may be made for archival purposes; or (vii) remove or obscure any proprietary rights notices or labels.

5. Transfers. You may not transfer or assign, in any manner, including by operation of law, the Software or any rights under this Agreement without the prior written consent of DELCOMp, which consent shall not be unreasonably withheld. A condition to any transfer or assignment shall be that the recipient agrees to the terms of this Agreement. Any attempted transfer or assignment in violation of this provision shall be null and void.

6. Ownership. DELCOMp and its suppliers own the Software and all intellectual property rights embodied therein, including patents, copyrights and valuable trade secrets embodied in the Software's design and coding methodology. The Software is protected by EC and United States patents, copyright and trade secret laws and international treaty provisions.

Change List

1.0

Initial release

1.1

- #include replaced by #use
- added #fatal alternative for #error

1.2

- changes to reflect library 2.0 release
- Dynamic C 9.62 and Dynamic C 10.50 updates

1.3

- removed references to Softools compiler
- updated installation instructions for new Dynamic C versions
- Chapter 7 functions updated