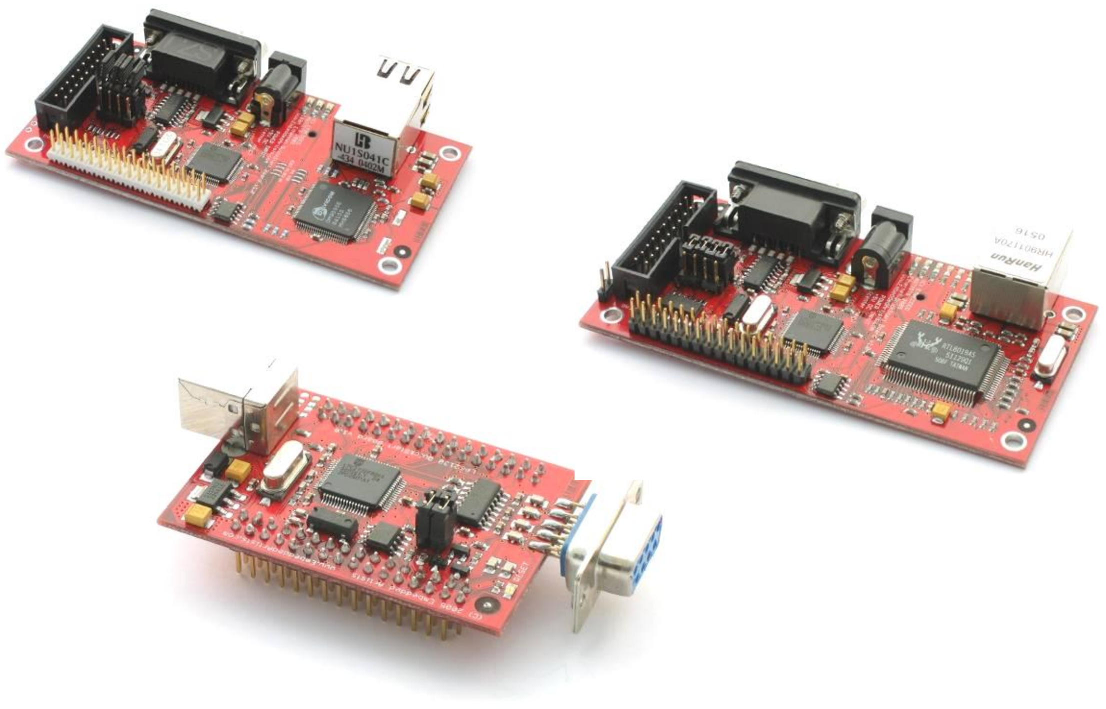
Copyright 2009 © Embedded Artists AB

QuickStart Program Development User's Guide



Get Up-and-Running Quickly and Start Developing on Day 1…

Embedded Artists

Embedded Artists AB
Södra Promenaden 51
SE-211 38 Malmö
Sweden

info@EmbeddedArtists.com
http://www.EmbeddedArtists.com

Copyright 2005-2009 © Embedded Artists AB. All rights reserved.

No part of this publication may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language or computer language, in any form or by any means, electronic, mechanical, magnetic, optical, chemical, manual or otherwise, without the prior written permission of Embedded Artists AB.

Disclaimer

Embedded Artists AB makes no representation or warranties with respect to the contents hereof and specifically disclaim any implied warranties or merchantability or fitness for any particular purpose. Information in this publication is subject to change without notice and does not represent a commitment on the part of Embedded Artists AB.

Feedback

We appreciate any feedback you may have for improvements on this document. Please send your comments to support@EmbeddedArtists.com.

Trademarks

InfraBed and ESIC are trademarks of Embedded Artists AB. All other brand and product names mentioned herein are trademarks, services marks, registered trademarks, or registered service marks of their respective owners and should be treated as such.

Table of Contents

1 Introduction

Thank you for buying a product from Embedded Artists. This document is a User's Guide that describes how to get started with application program development on day 1 using our *QuickStart Build Environment* for NXP's LPC2xxx family of ARM7 microcontrollers. This document is common for our LPC2xxx *QuickStart/Education/OEM Boards* from Embedded Artists.

1.1 Low Cost QuickStart Boards

Our *QuickStart/Education/OEM Boards* are very low cost and can be used for prototyping / development as well as for OEM production. Modifications for OEM production can easily be done, even for modest production volumes. Contact Embedded Artists for further information about design and production services.

1.1.1 Design and Production Services

Embedded Artists provide design services for custom designs, either completely new or modification to existing boards. Specific peripherals and/or I/O can easily be added to the different designs, for example communication interfaces, specific analogue or digital I/O, and power supplies. Embedded Artists has a broad, and long, experience in designing industrial electronics in general, and specifically with NXP LPC2xxx microcontroller family. Our competence also include wireless and wired communication for embedded systems, such as IEEE802.11b/g (WLAN), Bluetooth™, ZigBee™, ISM RF, Ethernet, CAN, RS485, and Fieldbuses.

1.2 Other QuickStart/Education/OEM Boards and Kits

Visit Embedded Artists' home page, www.EmbeddedArtists.com, for information about other *QuickStart/Education/OEM* boards / kits or contact your local distributor.

2 Getting Started

2.1 Test program

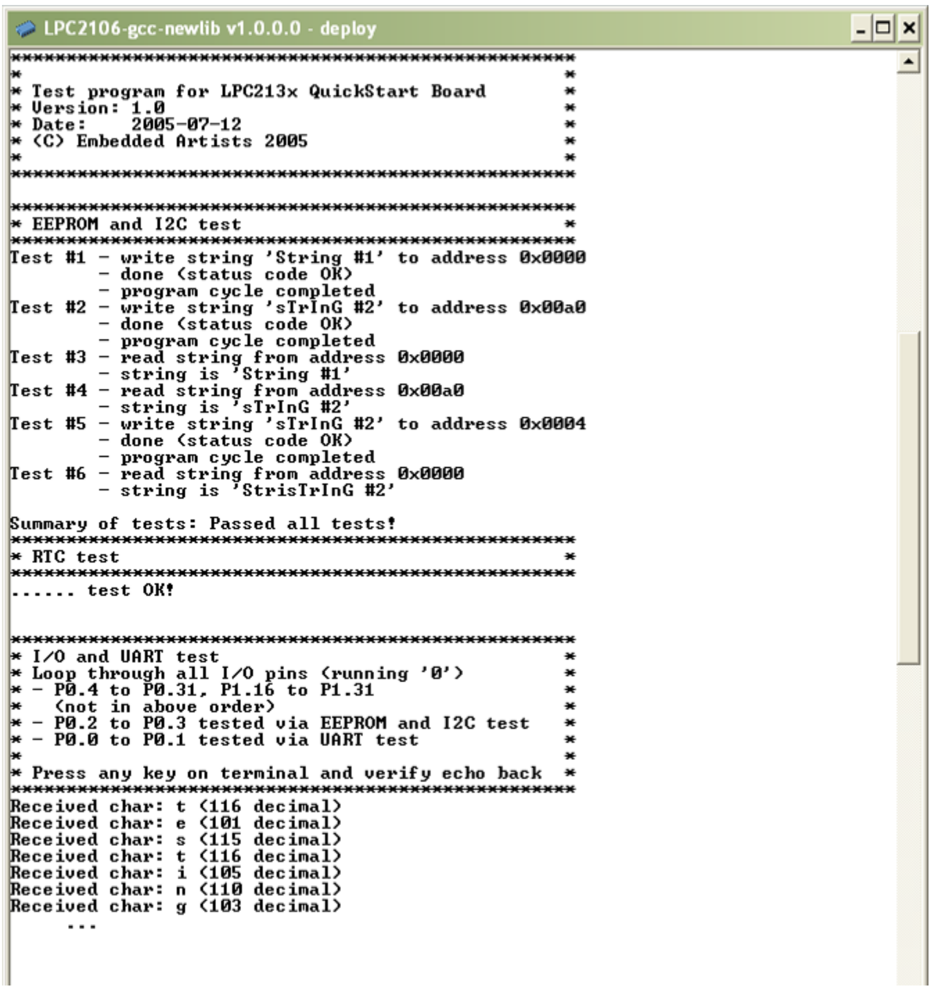
Some *QuickStart/Education/OEM Boards* come with a pre-installed test program. The test program is great for verifying that the board really works, and can for example be used if you suspect that the board has been damaged (for some reason).

See details about a possible test program in the User's Manual for the specific board. It can for example be a program that output a running-one pattern on the I/O pins. LEDs can then be connected to each I/O pin for simple verification. Internal functions, like the Real-Time Clock (RTC) and EEPROM will also be tested (if present) as well as the UART.

Connect the board to a terminal program on the PC. A great one is included on the disk. The baud rate differs between boards depending on different crystal clock frequencies. Check with the User's Manual for the specific board you are using, but in general the baud rate is 115200 bps for 14.7456 MHz crystals and 38400 bps for 12.0000 MHz crystals. 8 data bits, no parity bits, and one stop bit (i.e., 8N1) is typically used.

The test program will output test result information regarding the internal tests on the terminal, for example from the RTC, I^{2}C and E^{2}PROM tests. Also, the UART/RS232 channel can be tested by typing characters in the terminal program.

The output from the test program will look something like in *Figure 1* below. Note that the screen shot below is just an example. Every board is unique.



```
***************************************************
*                                                 *
* Test program for LPC213x QuickStart Board       *
* Version: 1.0                                    *
* Date:    2005-07-12                             *
* (C) Embedded Artists 2005                       *
*                                                 *
***************************************************

***************************************************
* EEPROM and I2C test                             *
***************************************************
Test #1 - write string 'String #1' to address 0x0000
        - done (status code OK)
        - program cycle completed
Test #2 - write string 'sTrInG #2' to address 0x00a0
        - done (status code OK)
        - program cycle completed
Test #3 - read string from address 0x0000
        - string is 'String #1'
Test #4 - read string from address 0x00a0
        - string is 'sTrInG #2'
Test #5 - write string 'sTrInG #2' to address 0x0004
        - done (status code OK)
        - program cycle completed
Test #6 - read string from address 0x0000
        - string is 'StrisTrInG #2'

Summary of tests: Passed all tests!
***************************************************
* RTC test                                        *
***************************************************
...... test OK!


***************************************************
* I/O and UART test                               *
* Loop through all I/O pins (running '0')         *
* - P0.4 to P0.31, P1.16 to P1.31                 *
*   (not in above order)                          *
* - P0.2 to P0.3 tested via EEPROM and I2C test   *
* - P0.0 to P0.1 tested via UART test             *
*                                                 *
* Press any key on terminal and verify echo back  *
***************************************************
Received char: t (116 decimal)
Received char: e (101 decimal)
Received char: s (115 decimal)
Received char: t (116 decimal)
Received char: i (105 decimal)
Received char: n (110 decimal)
Received char: g (103 decimal)
     ...
```

Figure 1 – Example Test Program Output

2.2 Program Download

For now, it is assumed that the program to be downloaded is already developed and there exist a HEX-file to be downloaded. This HEX-file represents the binary image of the application program.

There are basically two ways of downloading a program into the LPC2xxx microcontroller:

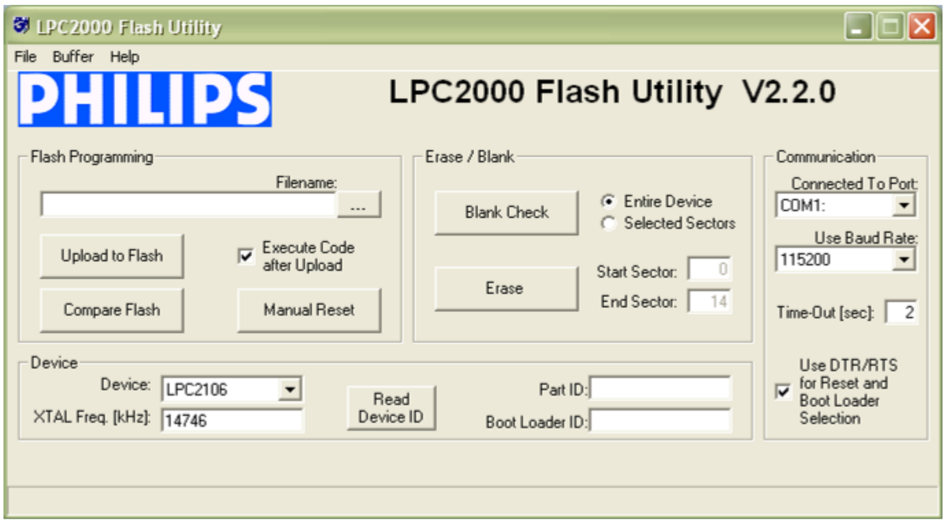
- ISP – In-System Programming
  The LPC2xxx microcontroller provides on-chip bootloader software that allows programming of the internal flash memory over the serial channel. The bootloader is activated by pulling port pin P0.14 (sometimes P2.10) low during reset of the microcontroller. Most boards contain a circuit for automatically controlling pin P0.14/P2.10 and the reset signal over the RS232 channel. This allows the program download to be fully automated.
    - FlashMagic (see link on CDROM/DVD)
    - NXP provides a utility program for In-System Flash (ISP) programming called *LPC2000 Flash Utility*.
    - Alternatively, there is a program called LPC21ISP that can be used. Source code is available. This program also provides a terminal functionality, which can be very helpful when developing your application program. The same serial channel that is used to download the program is typically also used for printing out information from the running program. The program immediately switch to terminal mode after program download and will hence not miss any characters sent on the serial channel directly after program start.
- JTAG
  For specific information about program download (i.e., Flash programming) with a JTAG interface, consult the manual for the specific JTAG interface that is used (e.g., CrossConnect from Rowley Associates, J-link from Segger, Ulink from Keil, or Wiggler from MacRaigor).

Most *QuickStart/education/OEM Boards* have two jumpers / links that will connect the RS232 channel to the active control over pin P0.14/P2.10 and the reset signal. See the User's Manual for the specific board you are using for details. Some boards connect the serial channel to a USB-to-serial bridge chip, like the FT232RL from FTDI.

After program download, the jumpers / links can be left connected, or removed if needed. If for example the PC end controls the RS232 signals DTR and/or RTS during normal program execution, then it might be required that the jumpers / links are removed after program download.

2.2.1 NXP LPC2000 Flash Utility

NXP LPC2000 Flash Utility program looks like *Figure 2* below. Never versions than v2.2.0 may exist when you read this document.



Filename:
[✓] Execute Code after Upload
[✓ Entire Device] Selected Sectors
Start Sector: 0
End Sector: 14
Connected To Port: COM1:
Use Baud Rate: 115200
Time-Out [sec]: 2
[✓] Use DTR/RTS for Reset and Boot Loader Selection
Device: LPC2106
XTAL Freq. [kHz]: 14746
Part ID:
Boot Loader ID:

Figure 2 – NXP LPC2000 Flash Utility Screenshot

Configure the dialog as shown above. The program will control the RS232 signals DTR and RTS if the appropriate checkbox is checked, and hence provide fully automated program download.

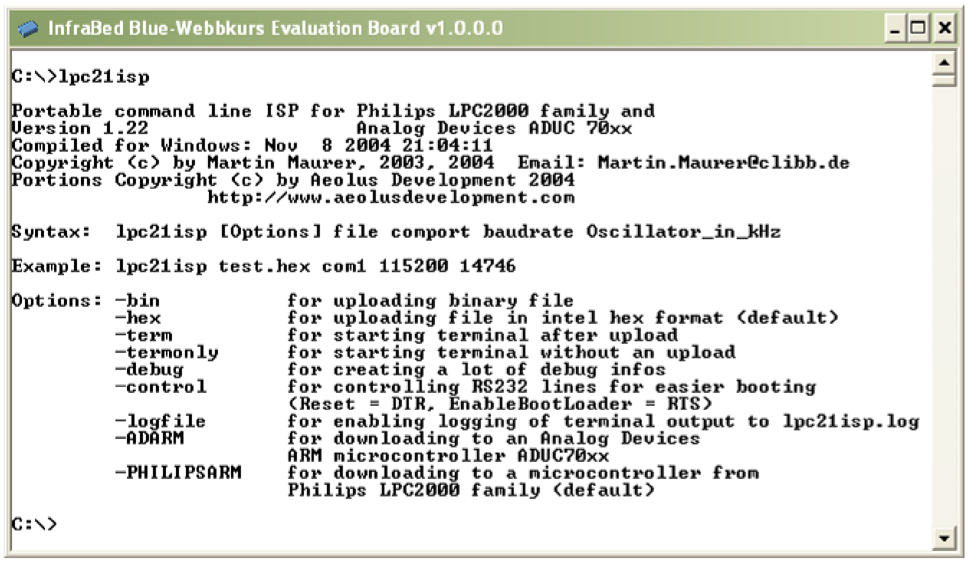
You can easily test the connection with a *QuickStart/Education/OEM Board* by pressing the *Read Device ID* button. The text fields for *Part ID* and *Boot Loader ID* will then contain uploaded information from the microcontroller. Note that the XTAL Freq. must be set to appropriate value. For most boards it's either 14.7456 MHz or 12.0000 MHz. In these cases the value 14746 or 12000 shall be written in the text box. If the crystal frequency has been changed, make sure the appropriate value is set. Set the communication baud rate to 115200 (for 14.7456 MHz crystal) or 38400 (a reliable connection cannot be achieved with higher baud rates if the crystal is 12 MHz). If no connection can be established test with a low *Baud Rate*, for example 1200 bps. Also verify that the correct COM-port has been selected (under *Connected to Port*).

Select the HEX file to be downloaded and then press the *Upload to Flash* button.

The downloaded program will immediately start after the download (i.e. the *Upload to Flash* operation is ready) is the option *Execute Code after Upload* is checked.

2.2.2 LPC21ISP

The LPC21ISP program is made publicly available by Martin Maurer. Source code is also available from a yahoo group that has been created for the program. *Figure 3* below shows the command syntax for the program.

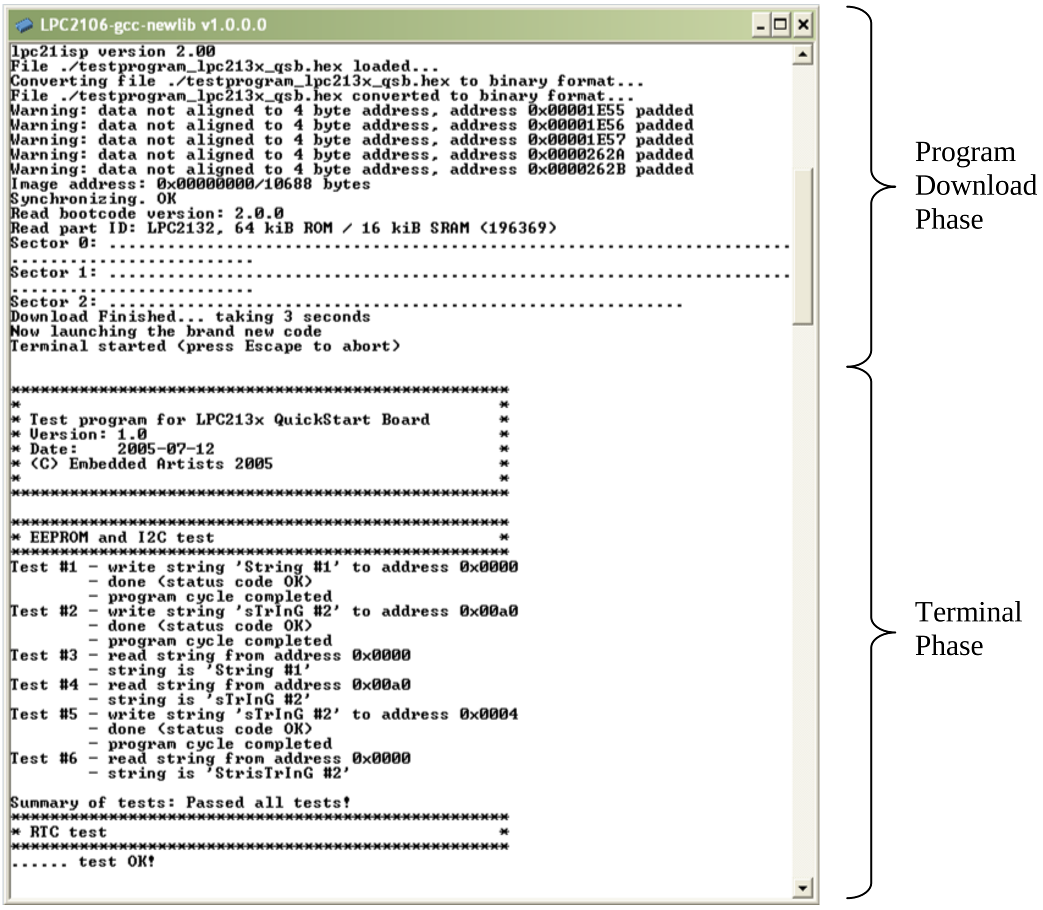


```
C:\>lpc21isp

Portable command line ISP for Philips LPC2000 family and
Version 1.22                 Analog Devices ADUC 70xx
Compiled for Windows: Nov  8 2004 21:04:11
Copyright (c) by Martin Maurer, 2003, 2004  Email: Martin.Maurer@clibb.de
Portions Copyright (c) by Aeolus Development 2004
                 http://www.aeolusdevelopment.com

Syntax:  lpc21isp [Options] file comport baudrate Oscillator_in_kHz

Example: lpc21isp test.hex com1 115200 14746

Options: -bin            for uploading binary file
         -hex            for uploading file in intel hex format (default)
         -term           for starting terminal after upload
         -termonly       for starting terminal without an upload
         -debug          for creating a lot of debug infos
         -control        for controlling RS232 lines for easier booting
                         (Reset = DTR, EnableBootLoader = RTS)
         -logfile        for enabling logging of terminal output to lpc21isp.log
         -ADARM          for downloading to an Analog Devices
                         ARM microcontroller ADUC70xx
         -PHILIPSARM     for downloading to a microcontroller from
                         Philips LPC2000 family (default)

C:\>
```

Figure 3 – LPC21ISP Portable Command Line ISP Screenshot

A typical program download sequence may look like in *Figure 4* below. Here, the test program is downloaded. As seen, the first part is the actual program download phase. Then this is done, the program switches to being a terminal (the second part) and the messages from the test program is displayed. It also sends anything typed on the keyboard back to the *QuickStart/Education/OEM Board*. As seen the program ends when ESC is pressed.

This sequence illustrates the benefits from using the program as a terminal directly after program download. No characters are missed after program start.



```
lpc21isp version 2.00
File ./testprogram_lpc213x_qsb.hex loaded...
Converting file ./testprogram_lpc213x_qsb.hex to binary format...
File ./testprogram_lpc213x_qsb.hex converted to binary format...
Warning: data not aligned to 4 byte address, address 0x00001E55 padded
Warning: data not aligned to 4 byte address, address 0x00001E56 padded
Warning: data not aligned to 4 byte address, address 0x00001E57 padded
Warning: data not aligned to 4 byte address, address 0x0000262A padded
Warning: data not aligned to 4 byte address, address 0x0000262B padded
Image address: 0x00000000/10688 bytes
Synchronizing. OK
Read bootcode version: 2.0.0
Read part ID: LPC2132, 64 kiB ROM / 16 kiB SRAM (196369)
Sector 0: ..................................................................
.........................
Sector 1: ..................................................................
.........................
Sector 2: ..........................................................
Download Finished... taking 3 seconds
Now launching the brand new code
Terminal started (press Escape to abort)


***************************************************
*                                                 *
* Test program for LPC213x QuickStart Board       *
* Version: 1.0                                    *
* Date:     2005-07-12                            *
* (C) Embedded Artists 2005                       *
*                                                 *
***************************************************

***************************************************
* EEPROM and I2C test                             *
***************************************************
Test #1 - write string 'String #1' to address 0x0000
        - done (status code OK)
        - program cycle completed
Test #2 - write string 'sTrInG #2' to address 0x00a0
        - done (status code OK)
        - program cycle completed
Test #3 - read string from address 0x0000
        - string is 'String #1'
Test #4 - read string from address 0x00a0
        - string is 'sTrInG #2'
Test #5 - write string 'sTrInG #2' to address 0x0004
        - done (status code OK)
        - program cycle completed
Test #6 - read string from address 0x0000
        - string is 'StrisTrInG #2'

Summary of tests: Passed all tests!
***************************************************
* RTC test                                        *
***************************************************
...... test OK!
```

Figure 4 – LPC21ISP Command Line ISP Download Screenshot Example

Another benefit with this program is that it runs under Linux.

Use version 1.48, or later, of LPC21ISP.EXE since older versions must be recompiled with increased reset timeout (when the program tries to synchronize to our *QuickStart/Education/OEM Boards*). The timeout should be increased to at least 350 ms.

2.3 Program Development Environment

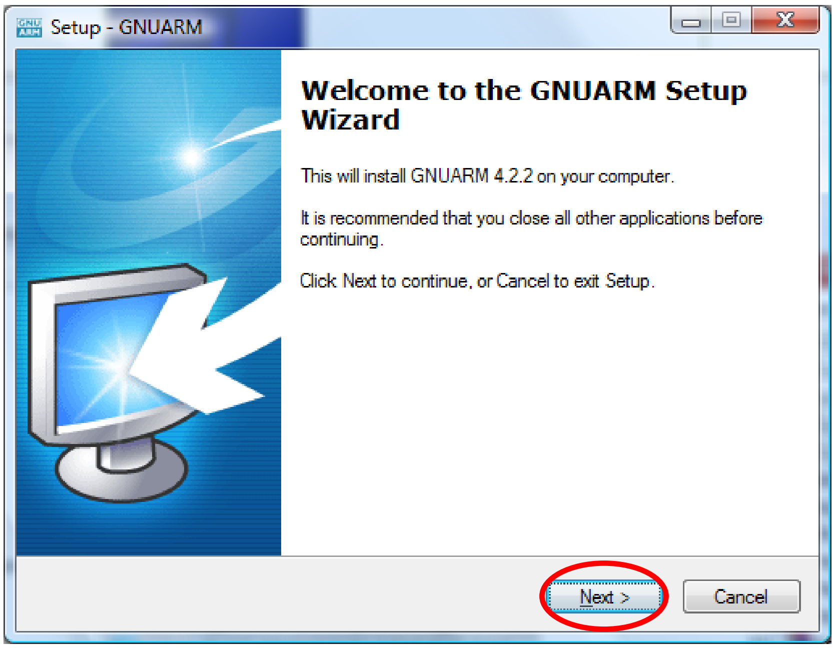
There are many options when it comes to the actual application program development. First of all, you must select a development environment, i.e., an editor (preferably with project management capabilities), a compiler package (compiler plus linker), and a debugger. Fortunately, there are many different choices for ARM program development, each with its pros and cons. The list below is far from complete but gives a general overview.

- ***QuickStart Build Environment from Embedded Artists***
  Embedded Artists has created a complete GCC build environment for all *QuickStart/Education/OEM Boards*. This will ease program development for novel users. By installing the *QuickStart Build Environment* you will automatically get a complete setup of the build environment.
- ***Keil uVision***
  This is another complete development environment, but from Keil. It includes an editor, project manager, a complete compiler build environment, and a debugger. A code size limited evaluation version can be downloaded from Keil's homepage.
- ***Rowley Associates CrossWorks for ARM***
  A complete development environment from Rowley Associates, including an editor, project manager, a complete compiler build environment, and a debugger. A 30-day limited evaluation version can be downloaded from Rowley's homepage.
- ***IAR Embedded Workbench***
  A complete development environment from IAR Systems, including an editor, project manager, a complete compiler build environment, and a debugger. A code size limited evaluation version can be downloaded from IAR's homepage.
- ***Programmers notepad***
  This is a very good editor and project manager that is increasing in popularity. The program can easily be integrated with the GCC compiler.
- ***Eclipse + CDT***
  This is a very good development environment (editor and project manager) with specific support for C/C++ code development. It does not contain a compiler but can easily be connected to one, for example GCC.
- ***GCC distribution GNUARM***
  A complete distribution of GCC, specifically for ARM processors. Current version of GCC is 4.2.2.
- ***WinARM***
  This is another distribution that not only contains GCC but also Programmers Notepad, LPC21ISP, a terminal program, and JTAG drivers.

2.4 Installing the QuickStart Build Environment

This section describes the necessary steps of program installation that is needed to get the *QuickStart Build Environment* ready for your use.

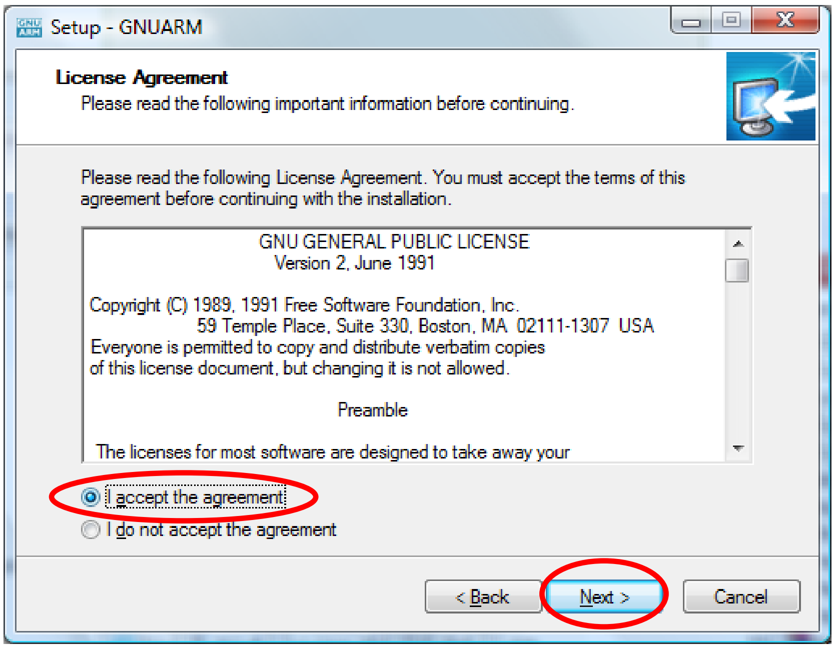
- Start with installing the GNUARM distribution that can be downloaded from Embedded Artists web site. The current version of the file is called: **`bu-2.18_gcc-4.2.2-c-c++ _nl-1.15.0_gi-6.7.1.exe`**.
  The installation is very simple and straightforward. It's just following the default installation steps as illustrated in the pictures below:



This will install GNUARM 4.2.2 on your computer.

It is recommended that you close all other applications before continuing.

Click Next to continue, or Cancel to exit Setup.



Please read the following important information before continuing.

Please read the following License Agreement. You must accept the terms of this agreement before continuing with the installation.

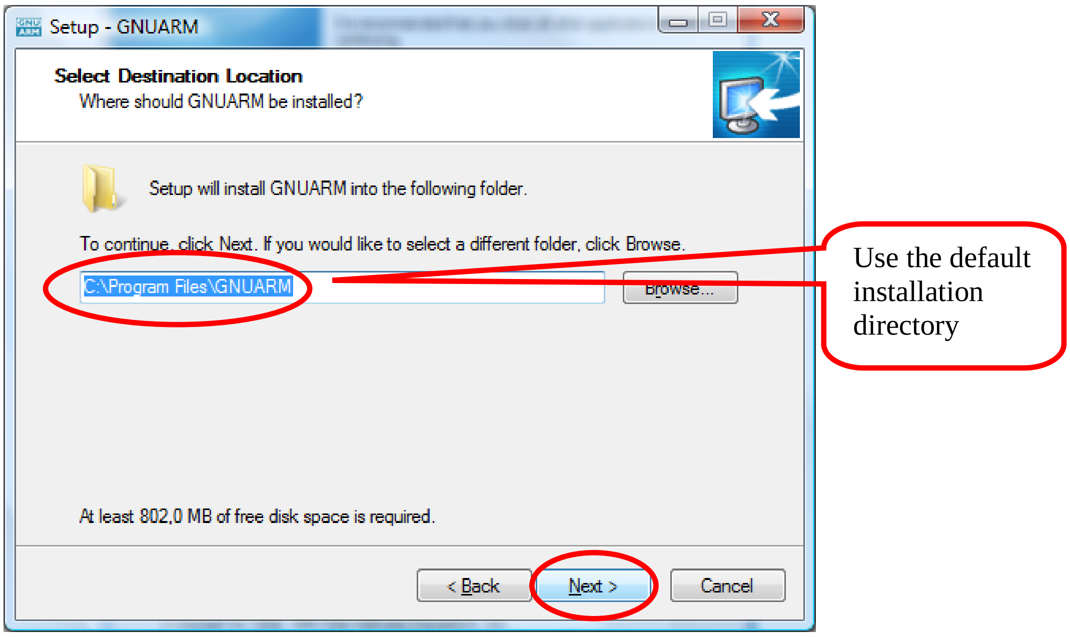
GNU GENERAL PUBLIC LICENSE
Version 2, June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc.
59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.

Preamble

The licenses for most software are designed to take away your

[✓ I accept the agreement]
I do not accept the agreement



Setup - GNUARM

Select Destination Location

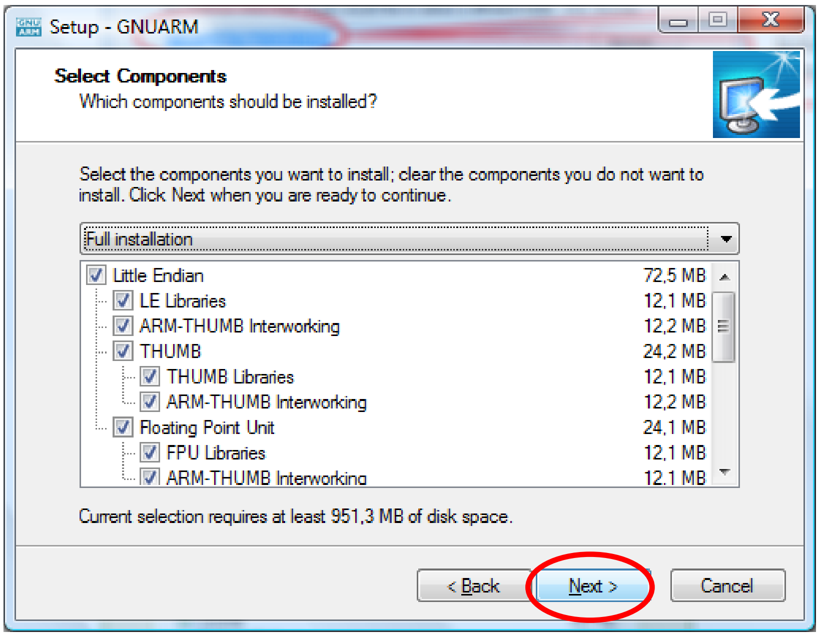
Where should GNUARM be installed?

Setup will install GNUARM into the following folder.

To continue, click Next. If you would like to select a different folder, click Browse.

C:\Program Files\GNUARM

At least 802,0 MB of free disk space is required.



Setup - GNUARM

Select Components

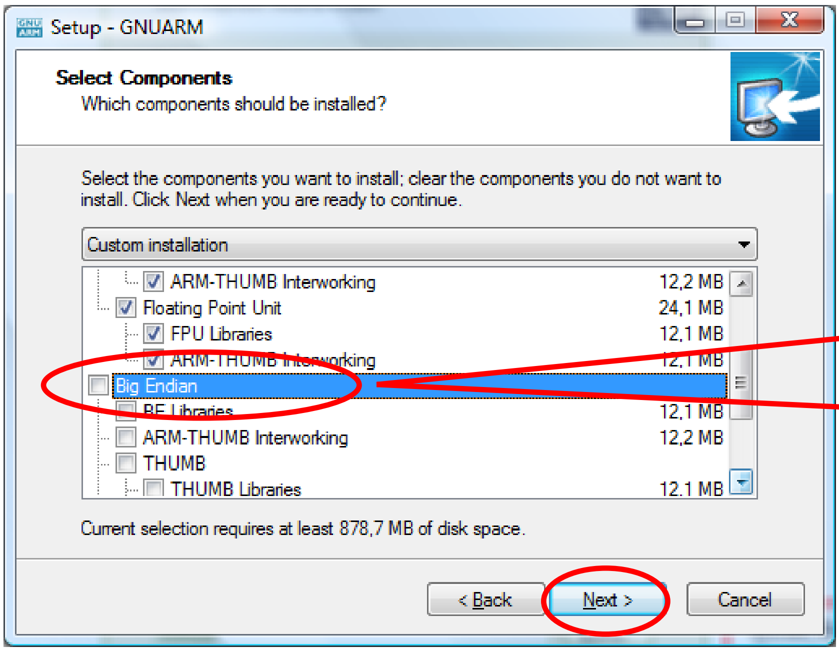
Which components should be installed?

Select the components you want to install; clear the components you do not want to install. Click Next when you are ready to continue.

Full installation

☑ Little Endian	72,5 MB
☑ LE Libraries	12,1 MB
☑ ARM-THUMB Interworking	12,2 MB
☑ THUMB	24,2 MB
☑ THUMB Libraries	12,1 MB
☑ ARM-THUMB Interworking	12,2 MB
☑ Floating Point Unit	24,1 MB
☑ FPU Libraries	12,1 MB
☑ ARM-THUMB Interworking	12,1 MB

Current selection requires at least 951,3 MB of disk space.



Setup - GNUARM

Select Components

Which components should be installed?

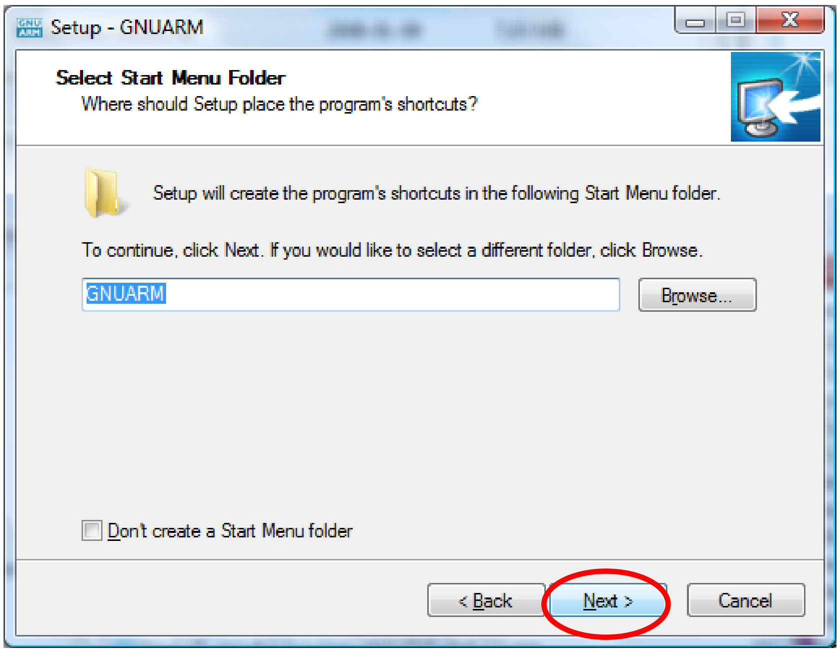
Select the components you want to install; clear the components you do not want to install. Click Next when you are ready to continue.

Custom installation

[✓] ARM-THUMB Interworking	12,2 MB
[✓] Floating Point Unit	24,1 MB
[✓] FPU Libraries	12,1 MB
[✓] ARM-THUMB Interworking	12,1 MB
[ ] Big Endian	
[ ] BE Libraries	12,1 MB
[ ] ARM-THUMB Interworking	12,2 MB
[ ] THUMB	
[ ] THUMB Libraries	12,1 MB

Current selection requires at least 878,7 MB of disk space.

If you want to save space on your harddisk, you can deseclect the *Big Endian* component.



Setup - GNUARM

Select Start Menu Folder

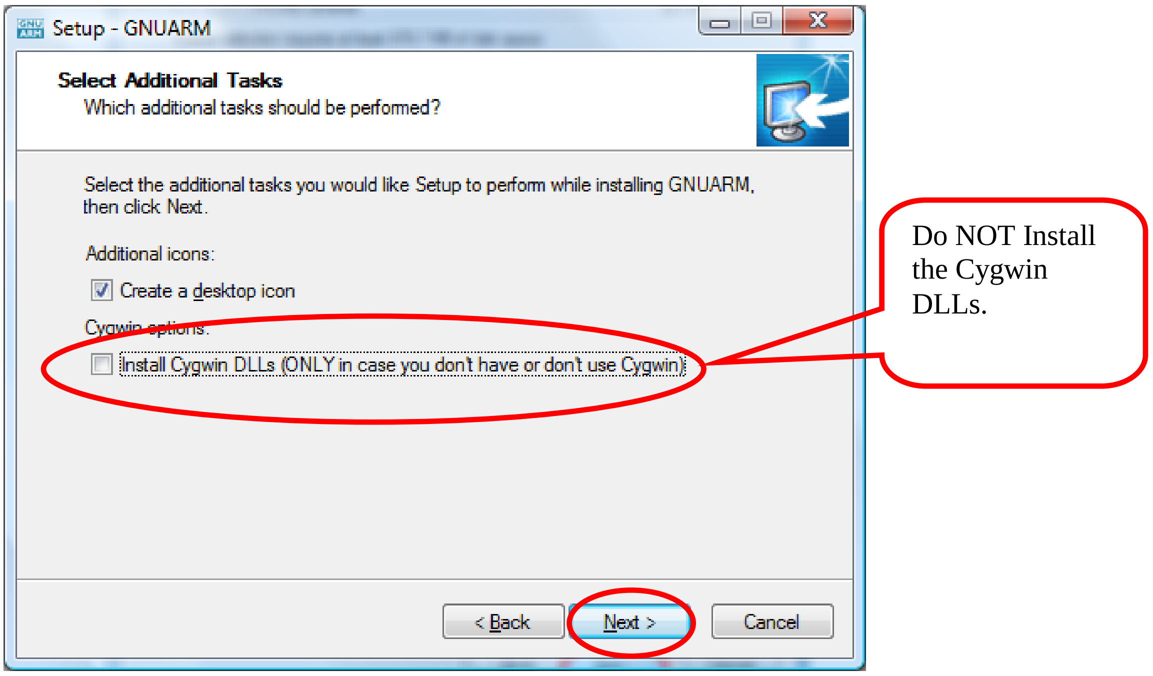
Where should Setup place the program's shortcuts?

Setup will create the program's shortcuts in the following Start Menu folder.

To continue, click Next. If you would like to select a different folder, click Browse.

GNUARM

[ ] Don't create a Start Menu folder


Setup - GNUARM

Select Additional Tasks

Which additional tasks should be performed?

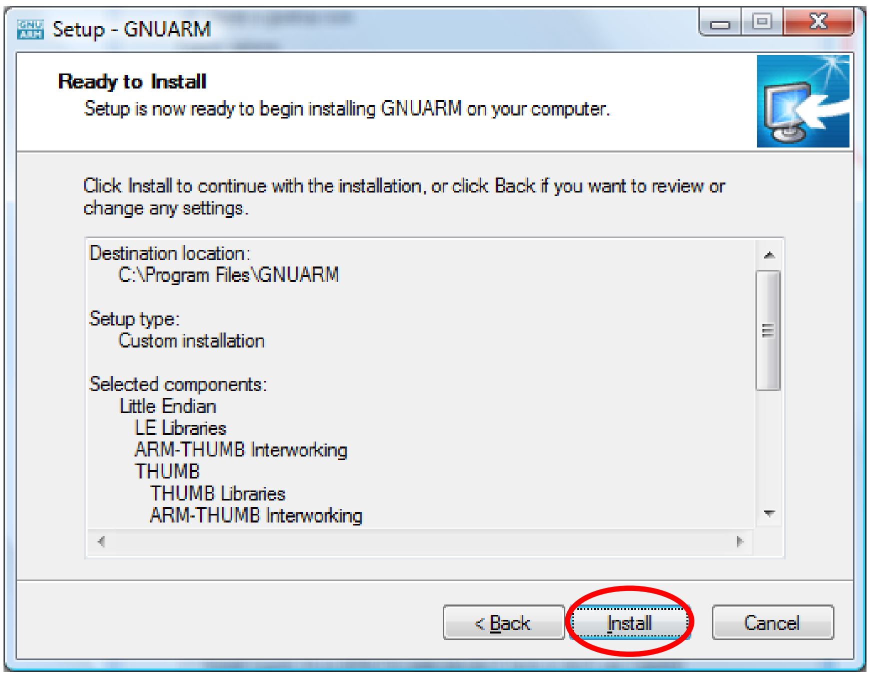
Select the additional tasks you would like Setup to perform while installing GNUARM, then click Next.

Additional icons:

- [✓] Create a desktop icon

Cygwin options:

- [ ] Install Cygwin DLLs (ONLY in case you don't have or don't use Cygwin)


Setup - GNUARM

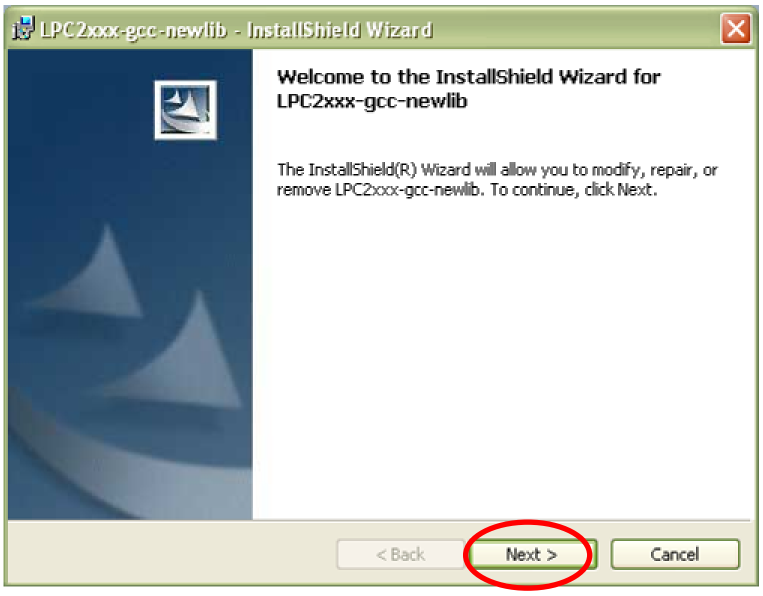
Ready to Install

Setup is now ready to begin installing GNUARM on your computer.

Click Install to continue with the installation, or click Back if you want to review or change any settings.

```
Destination location:
    C:\Program Files\GNUARM

Setup type:
    Custom installation

Selected components:
    Little Endian
      LE Libraries
      ARM-THUMB Interworking
      THUMB
        THUMB Libraries
        ARM-THUMB Interworking
```

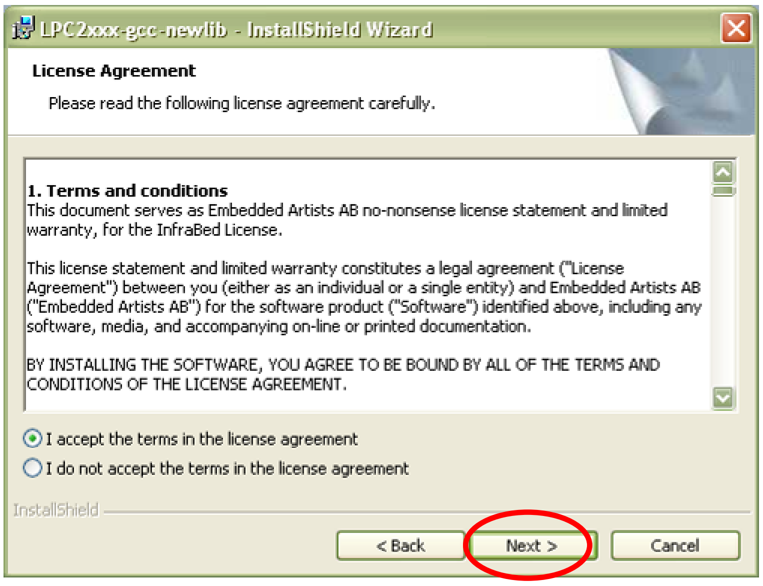
- Now install the **`LPC2xxx-gcc-newlib_vX_X_X_X`** *QuickStart Build Environment* (vX_X_X_X is the current version of the file). The installation is also in this case very simple and straightforward. Just follow the default installation steps.



LPC2xxx-gcc-newlib - InstallShield Wizard

Welcome to the InstallShield Wizard for LPC2xxx-gcc-newlib

The InstallShield(R) Wizard will allow you to modify, repair, or remove LPC2xxx-gcc-newlib. To continue, click Next.



LPC2xxx-gcc-newlib - InstallShield Wizard

License Agreement

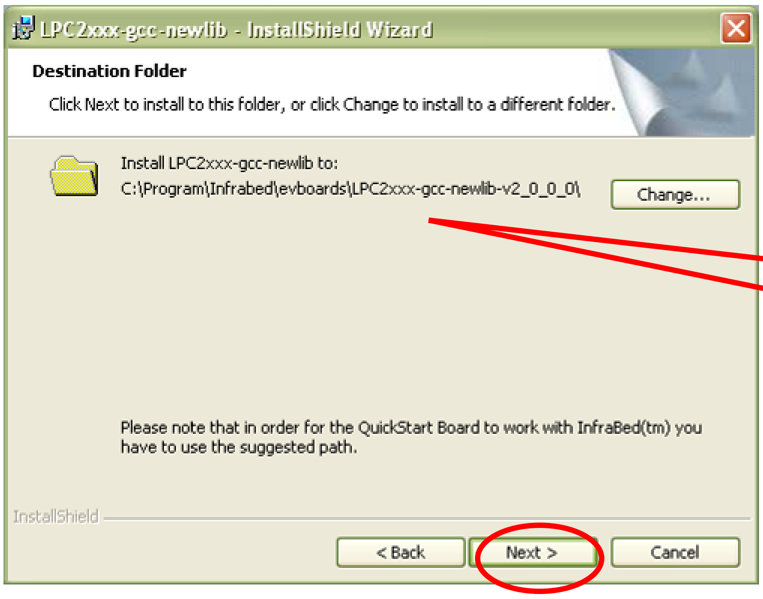
Please read the following license agreement carefully.

1. Terms and conditions
This document serves as Embedded Artists AB no-nonsense license statement and limited warranty, for the InfraBed License.

This license statement and limited warranty constitutes a legal agreement ("License Agreement") between you (either as an individual or a single entity) and Embedded Artists AB ("Embedded Artists AB") for the software product ("Software") identified above, including any software, media, and accompanying on-line or printed documentation.

BY INSTALLING THE SOFTWARE, YOU AGREE TO BE BOUND BY ALL OF THE TERMS AND CONDITIONS OF THE LICENSE AGREEMENT.

[✓ I accept the terms in the license agreement]
I do not accept the terms in the license agreement



LPC2xxx-gcc-newlib - InstallShield Wizard

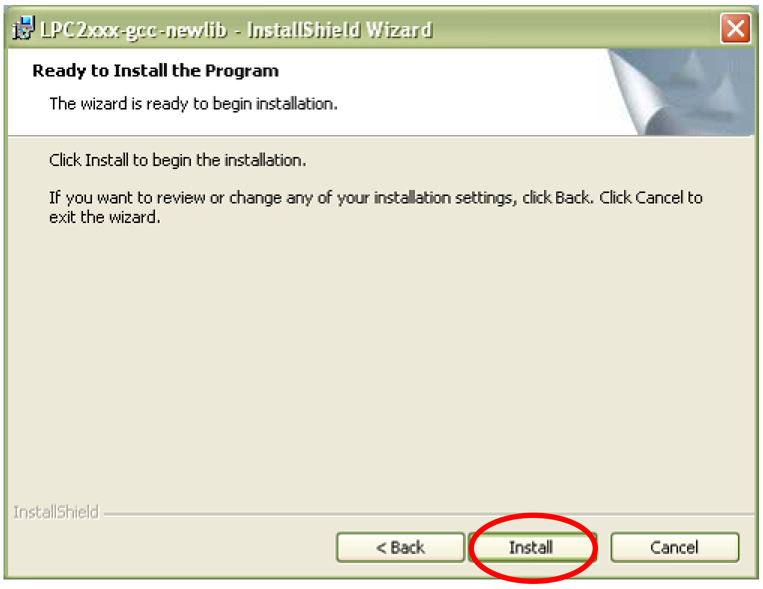
Destination Folder

Click Next to install to this folder, or click Change to install to a different folder.

Install LPC2xxx-gcc-newlib to:
C:\Program\Infrabed\evboards\LPC2xxx-gcc-newlib-v2_0_0_0\

Please note that in order for the QuickStart Board to work with InfraBed(tm) you have to use the suggested path.

Use the default installation directory



Ready to Install the Program

The wizard is ready to begin installation.

Click Install to begin the installation.

If you want to review or change any of your installation settings, click Back. Click Cancel to exit the wizard.

Note that if the compiler is not installed on the default location (**`c:/Program/GNUARM/`**) the new path must be set in the files **`build.sh`** and **`build_environment.sh`**. Both files can be found in:
`C:\Program\InfraBed\evboards\LPC2xxx-gcc-newlib-vX_X_X_X\bin`).
It is the variable **`COMPILERDIR2`** that must be set (can be found on row 13 in both files).
The compiler path must be to the **`GNUARM/bin`** directory.

Note that the path above must contain the correct version number instead of …**`vX_X_X_X\bin`**. It may for example be: …**`v2_4_0_1\bin`**.

3 QuickStart Build Environment

The *QuickStart Build Environment* is a complete build environment for GCC including program downloading via ISP. The build environment is built around a bash script. This script sets up all necessary paths. When installing the *QuickStart Build Environment* you will automatically get shortcuts to this bash script. A practical feature is that there can be different scripts for different hardware platforms, for controlling different hardware specific details of the platforms. There can also be many different compilers (including different versions of the same compiler) without conflicting with each other.

The use of the bash script is optional but is recommended for non-experienced users.

A typical project has two subdirectories; **`build_files`** and **`startup`**. *Figure 5* below illustrate the general structure.

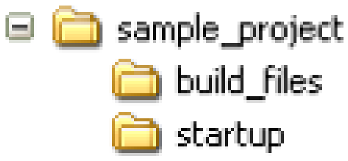


Figure 5 – Typical Project Directory Structure

You can download example projects from Embedded Artists support page (available after registration). These sample application projects have this basic structure. The followings sections describe different aspects of the *QuickStart Build Environment*.

3.1 Makefiles

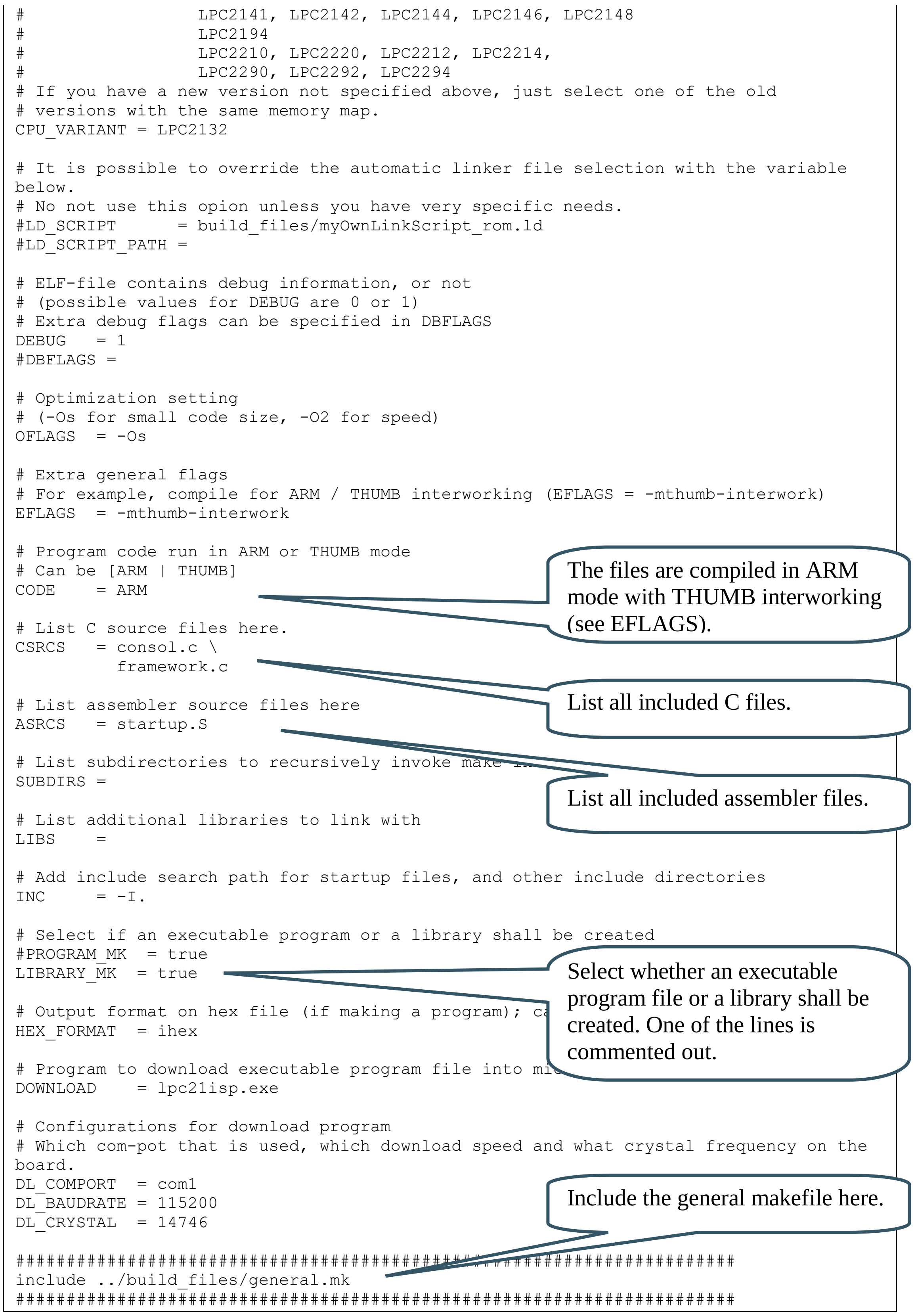
The subdirectory **`build_files`** contains a general makefile and linker script files. The subdirectory startup contains a configurable startup framework for *QuickStart/Education/OEM Board* projects. The startup files form a library that is linked to the main application.

The makefiles have a hierarchical structure. Each project, either an executable program file or a library, has a simple **`makefile`** that just describe the specifics of the project. This simple **`makefile`** includes the general **`makefile`** that is placed in the **`build_files`** subdirectory.

Figure 6 below illustrates the simple **`makefile`**. The example comes from the startup library, found under the **`startup`** subdirectory. The name of the resulting library is **`libea_startup_thumb.a`**. Two C-source code files are listed: **`consol.c`** and **`framework.c`**. An assembler file called **`startup.S`** is also included in the library.

```
##########################################################
#
# General makefile for building executable programs and
# libraries for Embedded Artists' QuickStart Boards.
# (C) 2001-2005 Embedded Artists AB
#
##########################################################

# Name of target (executable program or library)
NAME      = libea_startup_thumb

# Link program to RAM or ROM (possible values for LD_RAMROM is RAM or ROM,
# if not specified = ROM)
LD_RAMROM = ROM

# Name if specific CPU used (used by linker scripts to define correct memory map)
# Valid CPUs are: LPC2101, LPC2102, LPC2103, LPC2104, LPC2105, LPC2106
#                 LPC2114, LPC2119
#                 LPC2124, LPC2129
#                 LPC2131, LPC2132, LPC2134, LPC2136, LPC2138
```

Name of resulting library.



```
#                  LPC2141, LPC2142, LPC2144, LPC2146, LPC2148
#                  LPC2194
#                  LPC2210, LPC2220, LPC2212, LPC2214,
#                  LPC2290, LPC2292, LPC2294
# If you have a new version not specified above, just select one of the old
# versions with the same memory map.
CPU_VARIANT = LPC2132

# It is possible to override the automatic linker file selection with the variable
below.
# No not use this opion unless you have very specific needs.
#LD_SCRIPT      = build_files/myOwnLinkScript_rom.ld
#LD_SCRIPT_PATH =

# ELF-file contains debug information, or not
# (possible values for DEBUG are 0 or 1)
# Extra debug flags can be specified in DBFLAGS
DEBUG   = 1
#DBFLAGS =

# Optimization setting
# (-Os for small code size, -O2 for speed)
OFLAGS  = -Os

# Extra general flags
# For example, compile for ARM / THUMB interworking (EFLAGS = -mthumb-interwork)
EFLAGS  = -mthumb-interwork

# Program code run in ARM or THUMB mode
# Can be [ARM | THUMB]
CODE    = ARM

# List C source files here.
CSRCS   = consol.c \
          framework.c

# List assembler source files here
ASRCS   = startup.S

# List subdirectories to recursively invoke make
SUBDIRS =

# List additional libraries to link with
LIBS    =

# Add include search path for startup files, and other include directories
INC     = -I.

# Select if an executable program or a library shall be created
#PROGRAM_MK  = true
LIBRARY_MK  = true

# Output format on hex file (if making a program); ca
HEX_FORMAT  = ihex

# Program to download executable program file into mi
DOWNLOAD    = lpc21isp.exe

# Configurations for download program
# Which com-pot that is used, which download speed and what crystal frequency on the
board.
DL_COMPORT  = com1
DL_BAUDRATE = 115200
DL_CRYSTAL  = 14746

#######################################################################
include ../build_files/general.mk
#######################################################################
```

The files are compiled in ARM mode with THUMB interworking (see EFLAGS).
List all included C files.
List all included assembler files.
Select whether an executable program file or a library shall be created. One of the lines is commented out.
Include the general makefile here.

Figure 6 – Example QuickStart Build Environment Makefile from Startup Library

As seen in *Figure 6* above the makefile ends with the command: **`include ../build_files/general.mk`**. This is a general make file that is part of the complete build environment. This part contains all specific details of compiler and linker invocation.

Also at the end, the target must be decided; either an executable program or a library. Either **`PROGRAM_MK`** or **`LIBRARY_MK`** must be set to **`true`**.

The example makefile above is quite simple to its structure. It is possible to create more complex project structures that contain many subprojects. A typical example is to have an application project in a root folder. Under this root folder a number of subdirectories exist containing different blocks of functionality. For example, this can be a Real-Time Operating System and a TCP/IP stack. This calls for a recursive **`makefile`** structure.

The **`makefile`** in the root filer will create an executable program. It also includes the **`makefile`** in each of the subdirectories. The **`makefiles`** that exist in subdirectories will create libraries. An example of a root make file is presented in *Figure 7* below.

```
###########################################################
#
# General makefile for building executable programs and
# libraries for Embedded Artists' QuickStart Boards.
# (C) 2001-2005 Embedded Artists AB
#
#######################################################

# Name of target (executable program or library)
NAME      = testprogram_10m_eth

# Link program to RAM or ROM (possible values for LD_RAMROM is RAM or ROM,
# if not specified = ROM)
LD_RAMROM = ROM

# Name if specific CPU used (used by linker scripts to define correct memory map)
# Valid CPUs are: LPC2101, LPC2102, LPC2103, LPC2104, LPC2105, LPC2106
#                 LPC2114, LPC2119
#                 LPC2124, LPC2129
#                 LPC2131, LPC2132, LPC2134, LPC2136, LPC2138
#                 LPC2141, LPC2142, LPC2144, LPC2146, LPC2148
#                 LPC2194
#                 LPC2210, LPC2220, LPC2212, LPC2214,
#                 LPC2290, LPC2292, LPC2294
# If you have a new version not specified above, just select one of the old
# versions with the same memory map.
CPU_VARIANT = LPC2132

# It is possible to override the automatic linker file selection with the variable
below.
# No not use this opion unless you have very specific needs.
#LD_SCRIPT      = build_files/myOwnLinkScript_rom.ld
#LD_SCRIPT_PATH =

# ELF-file contains debug information, or not
# (possible values for DEBUG are 0 or 1)
# Extra debug flags can be specified in DBFLAGS
DEBUG    = 1
#DBFLAGS =

# Optimization setting
# (-Os for small code size, -O2 for speed)
OFLAGS   = -Os

# Extra general flags
# For example, compile for ARM / THUMB interworking (EFLAGS = -mthumb-interwork)
EFLAGS   =

# Program code run in ARM or THUMB mode
# Can be [ARM | THUMB]
CODE     = THUMB

# List C source files here.
CSRCS    = main.c

# List assembler source files here
ASRCS    =

# List subdirectories to recursively invoke make in
```

Name of resulting program file.

Custom linker scripts can be used, but not in this makefile.

The files are compiled in THUMB mode.

The root folder only contains one file, the main-file.

Three different subdirectories that contains different blocks of functions in the final application.

```
SUBDIRS = startup \
          tcpip \
          pre_emptive_os

# List additional libraries to link with
LIBS    = startup/libea_startup_thumb.a \
          tcpip/tcpip.a \
          pre_emptive_os/pre_emptive_os.a

# Add include search path for startup files, and other include directories
INC     = -I./startup

# Select if an executable program or a library shall be created
PROGRAM_MK  = true
#LIBRARY_MK  = true

# Output format on hex file (if making a program); can be [srec | ihex]
HEX_FORMAT  = ihex

# Program to download executable program file into microcontroller's FLASH
DOWNLOAD    = lpc21isp.exe

# Configurations for download program
# Which com-pot that is used, which download speed and what crystal frequency on the
board.
DL_COMPORT  = com1
DL_BAUDRATE = 115200
DL_CRYSTAL  = 14746

######################################################################
include build_files/general.mk
######################################################################
```

The three libraries that are created in the recursive invocation of make are included in the final application.
Note the startup library.

Figure 7 – Example Root Makefile and Recursive Invocation

To build the application program, start a command prompt (the bash script), change directory to the project root, and type: **`make`**. Depending on the make file content, either an executable program or a library will be created. To also download the executable program, type: **`make deploy`** instead of just **`make`**.

A final note about the make file; **`make clean`** will erase all object files and **`make depend`** will recreate dependency files (this is also always done when typing just **`make`**). Finally, **`make terminal`** will just start the terminal function in the download program (lpc21isp). The specific settings for using the ISP download program can be set with the **`DL_XXX`** variables (as seen at the end of *Figure 7* above).

3.2 Startup Framework

As already mentioned, the startup files form a configurable startup framework. This is often called a *Board Support Package* or BSP for short. It contains the very basic startup and initialization code as well as a console with printf()- and scanf()-like functionality. The BSP is very configurable and can be changed according to your specific needs. Each project can have its specific settings. An example of how the configuration file looks like can be found in *Figure 8* below, and can be found in file **`config.h`** in the **`startup`** subdirectory.

```
/******************************************************************************
 *
 * Copyright:
 *    (C) 2000 - 2005 Embedded Artists AB
 *
 * Description:
 *    Framework for ARM7 processor
 *
 *****************************************************************************/
#ifndef _config_h_
#define _config_h_

/******************************************************************************
 * Defines, macros, and typedefs
 *****************************************************************************/
#define FOSC 14745600                 /* External clock input frequency (must be
                                         between 10 MHz and 25 MHz) */

#define USE_PLL 1                     /* 0 = do not use on-chip PLL,
                                         1 = use on-chip PLL) */
#define PLL_MUL 4                     /* PLL multiplication factor (1 to 32) */
#define PLL_DIV 2                     /* PLL division factor (1, 2, 4, or 8) */
#define PBSD    4                     /* Peripheral bus speed divider (1, 2, or 4) */

/* initialize the MAM (Memory Accelerator Module) */
#if USE_PLL == 1
#define CORE_FREQ (FOSC * PLL_MUL)
#else
#define CORE_FREQ (FOSC)
#endif

#if CORE_FREQ < 20000000
#define MAM_TIMING   1                /* number of CCLK to read from the FLASH */
#elif CORE_FREQ < 40000000
#define MAM_TIMING   2                /* number of CCLK to read from the FLASH */
#else
#define MAM_TIMING   3                /* number of CCLK to read from the FLASH */
#endif

#define MAM_SETTING  2                /* 0=disabled,
                                         1=partly enabled (enabled for code prefetch,
                                           but not for data),
                                         2=fully enabled */

#define IRQ_HANDLER  1                /* 0 = Jump to common IRQ handler
                                         1 = Load vector directly from VIC, i.e.,
                                             LDR PC,[PC,#-0xFF0] */

/* initialize the exception vector mapping */
#ifndef RAM_EXEC
#define MAM_MAP      1         /* 1 = exception vectors are in FLASH at 0x0000 0000,
                                  2 = exception vectors are in SRAM at 0x4000 0000 */
#else
#define MAM_MAP 2              /* When exec. from RAM, MAM_MAP should always be 2 */
#endif

/* setup stack sizes */
#define stackSize_SYS    600
#define stackSize_SVC     64
#define stackSize_UND     64
#define stackSize_ABT     64
#define stackSize_IRQ    600
#define stackSize_FIQ     64

/* define consol settings */
#define CONSOL_UART               0
#define CONSOL_BITRATE      115200
#define CONSOL_STARTUP_DELAY            /* Short startup delay in order to remove
                                           risk for false startbit detection,
                                           timer #1 will be used in polled mode */
#define CONSOL_STARTUP_DELAY_LENGTH 100 /* 100 us is slightly more than one
                                           character at 115200 bps */


#define USE_NEWLIB           0   /* 0 = do not use newlib (= save about 22k FLASH),
                                    1 = use newlib = full implementation of printf(),
                                        scanf(), and malloc() */
#define CONSOLE_API_PRINTF   1   /* 0 = printf() = sendString,
                                    1 = simple, own implementation of printf() */
#define CONSOLE_API_SCANF    0   /* 0 = none,
                                    1 = simple, own implementation of scanf() */

/* define SRAM size */
#ifdef LPC2101
#define SRAM_SIZE  (2 * 1024)   /* LPC2101 */
#elif defined (LPC2102)
#define SRAM_SIZE  (4 * 1024)   /* LPC2102 */
#elif defined (LPC2103)
#define SRAM_SIZE  (8 * 1024)   /* LPC2103 */
```

Copyright 2009 © Embedded Artists AB

```
#elif defined (LPC2104)
#define SRAM_SIZE  (16 * 1024)   /* LPC2104 */
#elif defined (LPC2105)
#define SRAM_SIZE  (32 * 1024)   /* LPC2105 */
#elif defined (LPC2106)
#define SRAM_SIZE  (64 * 1024)   /* LPC2106 */

#elif defined (LPC2114)
#define SRAM_SIZE  (16 * 1024)   /* LPC2114 */
#elif defined (LPC2124)
#define SRAM_SIZE  (16 * 1024)   /* LPC2124 */
#elif defined (LPC2119)
#define SRAM_SIZE  (16 * 1024)   /* LPC2119 */
#elif defined (LPC2129)
#define SRAM_SIZE  (16 * 1024)   /* LPC2129 */

#elif defined (LPC2131)
#define SRAM_SIZE  (8 * 1024)   /* LPC2131 */
#elif defined (LPC2132)
#define SRAM_SIZE  (16 * 1024)   /* LPC2132 */
#elif defined (LPC2134)
#define SRAM_SIZE  (16 * 1024)   /* LPC2134 */
#elif defined (LPC2136)
#define SRAM_SIZE  (32 * 1024)   /* LPC2136 */
#elif defined (LPC2138)
#define SRAM_SIZE  (32 * 1024)   /* LPC2138 */

#elif defined (LPC2141)
#define SRAM_SIZE  (8 * 1024)   /* LPC2141 */
#elif defined (LPC2142)
#define SRAM_SIZE  (16 * 1024)   /* LPC2142 */
#elif defined (LPC2144)
#define SRAM_SIZE  (16 * 1024)   /* LPC2144 */
#elif defined (LPC2146)
#define SRAM_SIZE  (32 * 1024)   /* LPC2146 */
#elif defined (LPC2148)
#define SRAM_SIZE  (32 * 1024)   /* LPC2148 */

#elif defined (LPC2194)
#define SRAM_SIZE  (16 * 1024)   /* LPC2194 */

#elif defined (LPC2210)
#define SRAM_SIZE  (16 * 1024)   /* LPC2210 */
#elif defined (LPC2220)
#define SRAM_SIZE  (64 * 1024)   /* LPC2220 */
elif defined (LPC2212)
#define SRAM_SIZE  (16 * 1024)   /* LPC2212 */
#elif defined (LPC2214)
#define SRAM_SIZE  (16 * 1024)   /* LPC2214 */
#elif defined (LPC2290)
#define SRAM_SIZE  (16 * 1024)   /* LPC2290 */
#elif defined (LPC2292)
#define SRAM_SIZE  (16 * 1024)   /* LPC2292 */
#elif defined (LPC2294)
#define SRAM_SIZE  (16 * 1024)   /* LPC2294 */

#else
#error CPU_VARIANT not defined in the makefile, or illegal value
#endif

#define SRAM_SADDR   0x40000000               /* SRAM starting address */
#define SRAM_TOP     (SRAM_SADDR+SRAM_SIZE)   /* SRAM end address + 1 */
#define SRAM_EADDR   (SRAM_SADDR+SRAM_SIZE-1) /* SRAM end address */
#define STK_SIZE     (stackSize_SYS+stackSize_SVC+stackSize_UND+
                      stackSize_ABT+stackSize_IRQ+stackSize_FIQ)
#define STK_SADDR    (SRAM_EADDR+1-STK_SIZE)  /* Stack start address */

#endif  /* _config_h_ */
```

Figure 8 – Board Support Package (BSP) Configuration File

There are three versions of the consol in order to best fit different situations:

- A very simple version that basically only supports printing strings (without any formatting parts) and printing numbers (decimal or hexadecimal).
- A simple printf() implementation that supports the simplest formatting tags. The implementation has been designed for least possible stack usage (about 40 bytes).
- A full ANSI printf() implementation from newlib (part of the compiler environment that comes with GNUARM). This routine requires about 600 bytes of stack space and should normally not be used in resource constraint systems.

The code size for the first two alternatives is minimal (about 2k in program size for the entire framework). When using printf() from newlib, the code size is about 30 k for the entire framework (including a large part of the newlib library).

Just edit the configuration file above and recompile your project. The recursive nature of the makefiles will make sure that the startup library is recompiled and linked with the final executable program.

You can download example projects from Embedded Artists support page (available after registration).

The startup framework (BSP) is very simple and can best be understood by studying the sample application source code files. If using the console functionality (printf()- and scanf()-like functions) note that the function eaInit() must be called before printf() and the console can be used. The following code segment illustrates this.

```
#include <ea_init.h>
...
...
int main(void)
{
  eaInit();  //Now, the console/printf can be used
  ...
}
```

Also note that whenever the BSP printf() should be used, the following include file must be included into the source code file.

```
#include <printf_P.h>
```

As a summary; Embedded Artists' *QuickStart Build Environment* is comprised of:

- A make build environment, controlled by bash script. A program or library build is started via the command: **`make`**.
- A program download feature, by using the LPC21ISP program. A program build and download is started via the command: **`make deploy`**.
- A Board Support Package (BSP) with startup code and console functions (i.e., printf() and scanf()-like functionality).

3.3 GCC

This will be very similar to the *QuickStart Build Environment* example, except that you will have to set up all paths manually and create your own startup files. The **`make`** files will also be a bit more complex. An example **`makefile`** is presented in *Figure 9* below. Much more complex examples than the **`makefile`** below also exist.

```
#
# Example makefile that creates a program called 'test', containing the
# C-source code files: main.c, eeprom.c, and i2c.c plus the assembler
# file startup.S
#

LIBS    =
DEBUG   = -g
CFLAGS  = -Wall -nostartfiles -mthumb-interwork -mthumb
INCLUDE = -Iinc/ -Iinc/specific/               #specify include paths here
ARMCC   = arm-elf-gcc
OBJS    = main.o eeprom.o i2c.o startup.o
LDFLAGS = -Wl,-Trom.ld                         #this file controls the linker

all: test.hex

test: $(OBJS)
       arm-elf-gcc $(CFLAGS) $(LDFLAGS) $(OBJS) $(LIBS) -o test.elf
%.o: %.c
       arm-elf-gcc -c $(INCLUDE) $(CFLAGS) $<
%.o: %.S
       arm-elf-gcc -c $(INCLUDE) $(CFLAGS) $<
%.o: %.c
       arm-elf-gcc -c $(INCLUDE) $(CFLAGS) $<
%.hex: %
       arm-elf-objcopy -O ihex $<.elf $@
clean:
       rm -f *.o test.elf test.hex
```

Figure 9 – Example GCC Makefile

4 Further Information

The LPC2xxx microcontroller is a complex chip and there exist a number of other documents with a lot more information. The following documents are recommended as a complement to this document.

[1] NXP LPC2xxx Datasheet

[2] NXP LPC2xxx User's Manual

[3] NXPs LPC2xxx Errata Sheet

[4] ARM7TDMI Technical Reference Manual. Document identity: DDI0029G
http://www.arm.com/pdfs/DDI0029G_7TDMI_R3_trm.pdf

[5] ARM Architecture Reference Manual. Document identity: DDI0100E
Book, Second Edition, edited by David Seal, Addison-Wesley: ISBN 0-201-73719-1
Also available in PDF form on the ARM Technical Publications CD

[6] ARM System Developer's Guide – Designing and Optimizing System Software, by A.N. Sloss, D Symes, C. Wright. Elsevier: ISBN 1-55860-874-5

[7] Embedded System Design on a Shoestring, by Lewin Edwards.
Newnes: ISBN 0750676094.

[8] GNU Manuals
http://www.gnu.org/manual/

[9] GNU ARM tool chain for Cygwin
http://www.gnuarm.com

[10] An Introduction to the GNU Compiler and Linker, by Bill Gatliff
http://www.billgatliff.com

[11] LPC2000 Yahoo Group. A discussion forum dedicated entirely to the NXP LPC2xxx series of microcontrollers.
http://groups.yahoo.com/group/lpc2000/

[12] The Insider's Guide to the NXP ARM7-Based Microcontrollers, by Trevor Martin.
http://www.hitex.co.uk/arm/lpc2000book/index.html

Note that there can be newer versions of the documents than the ones linked to here. Always check for the latest information / version.

Copyright 2009 © Embedded Artists AB