Setting Up Dev C++ to Work with the ARM GCC Cross-Compiler

by

Richard A. Roth II

Table of Contents

Thanks

Thanks to all the people who have worked on and will continue to work on providing "free" tools for the world.

Great thanks should be given to the people who have worked on GCC and it derivatives along with the developers and people working on the Dev C++ project.

In short THANKS !

What this document is

This document is a short (and in many ways incomplete) list of things I have learned, struggled, and worked on to get arm-elf-gcc to work with Dev C++. Most of the information contained in this document should be similar for other GCC compliers, but I cannot be sure, I have only worked with the ARM version.

This is a simple "one shot" document, which will most likely not be maintained by the author, (I do not have the time to keep up with this document, or make changes, but I feel that this is important information that was difficult for a Windows user to obtain).

This document is provided without warranty or any support. The author assumes no responsibility for anything that happens because of or as a result of this document.

If this document is useful, please provide a "thanks" email to the following address: sixtsixcuda@hotmail.com. Suggestions and comments are welcome, but do not expect a response.

Why use Dev C++

Dev C++ is a very well laid out GUI front end for the GNU tool chain, which support many features that a programmer desires. It has a great editing compatibility, simple to use editor with many advanced features, like code completion and source browsing. This graphical interface is easy to use for most people experienced with using Microsoft® Visual C++ or similarly derived products. The Dev C++ package works great right "out of the box" for compiling programs for the windows archeture, but with a little work it can be configured to work with the wide range of cross-compilers the GNU tool chain offers.

Preliminary Steps and Notes

The first step to getting the Dev C++ to work is to setup the compiler environment; this must be working before any work to setup Dev C++ can be done. This is not included here, but the basic steps are as follows:

1. Download a suitable distribution for your target processor
2. Install the Cygwin "Linux command emulator" (this will give you the ability to use UNIX/Linux commands on your PC), this is required before setting upGCC.

3. Install Cygwin and the compiler distribution, to an appropriate path on your hardrive. I have heard it is best to have this installed drive and the drive used for the source files to be the same letter; but I have not had any problems.
4. Get you compiler to work in the Bash shell – I can provide no help as to how to do this, except it must work before going any farther – this was just trial and error (I am not at all a Linux / UNIX person).
5. Add the following (or something similar, depending upon where you have Cygwin installed) to you path statement (in Windows 9x edit your "autoexec.bat" file and reboot, in Windows 2000 press the window key and the break button, go to "Advanced" tab, press the "Environmental Variables" button and find the entry for "PATH", add the lines below). Also add an entry where the cross compiler is located (this will be something like arm-elf-gcc.exe – search for this file, or the file for your processor and add that directory to your path, mine was located in the "C:\Cygwin\usr\bin" directory).

```
C:\Cygwin\contrib\bin
C:\Cygwin\bin
C:\Cygwin\usr\bin
```

6. Open a DOS prompt (Start | programs | Accessories | Command Prompt) – or similar other version of Windows (I am using Windows 2000).

7. Type the following line verbatim (replace the pre-fix with your processor name).

```
C:\>arm-elf-gcc –v
Reading specs from …
gcc version …
```

8. If the following is not displayed (with your version number displayed and file path displayed in place of the dots), check your path statement (in Window 9x be sure to reboot the computer after changing the PATH in the "Autoexec.bat" file).

9. If all went well, you should be ready to set Dev C++.

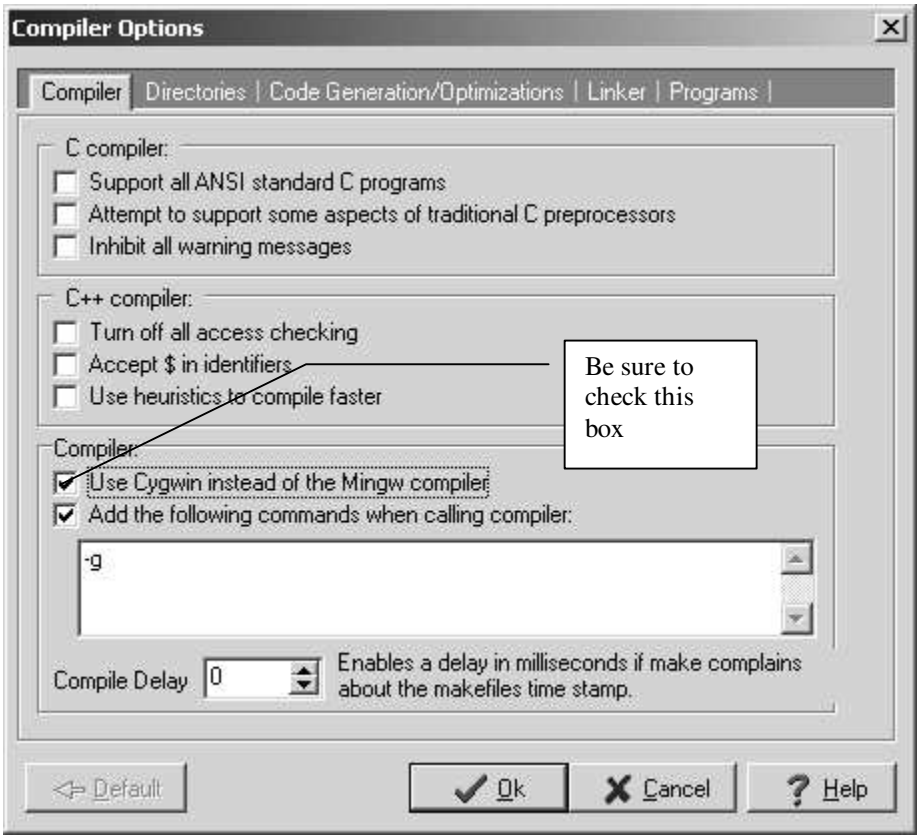
10. You must do all of the above and (Duh…) download Dev C++ from http://www.bloodshed.net/dev/devcpp.html).

Setting up the "Global" Dev C++ Options

Like most IDE there are a number of global options that apply to all projects and then there are project options that only affect the current project. This section aims to assist in setting up the "Global" options. All the screen shots in this document are from Version 4.9.5.0 of Dev C++, other version may differ, but should be similar in appearance.

The Compiler Tab

The first tab that is displayed is the compiler tab, in this tab be sure to select “Use Cygwin instead of the Mingw compiler”, and if you want to enable debugging add the option “-g” to the “Add the following commands when calling compiler:”.



Setting up the directories

The next tab is the directories tab, this is where you enter where Dev C++ can find the executables along with the standard libraries, **This is an important step.**

Please use something similar to the following entries in the table to setup the rest of the directories (note the exact path will differ from one distribution to the next, so please look at the notes to find out how to find the correct directory).

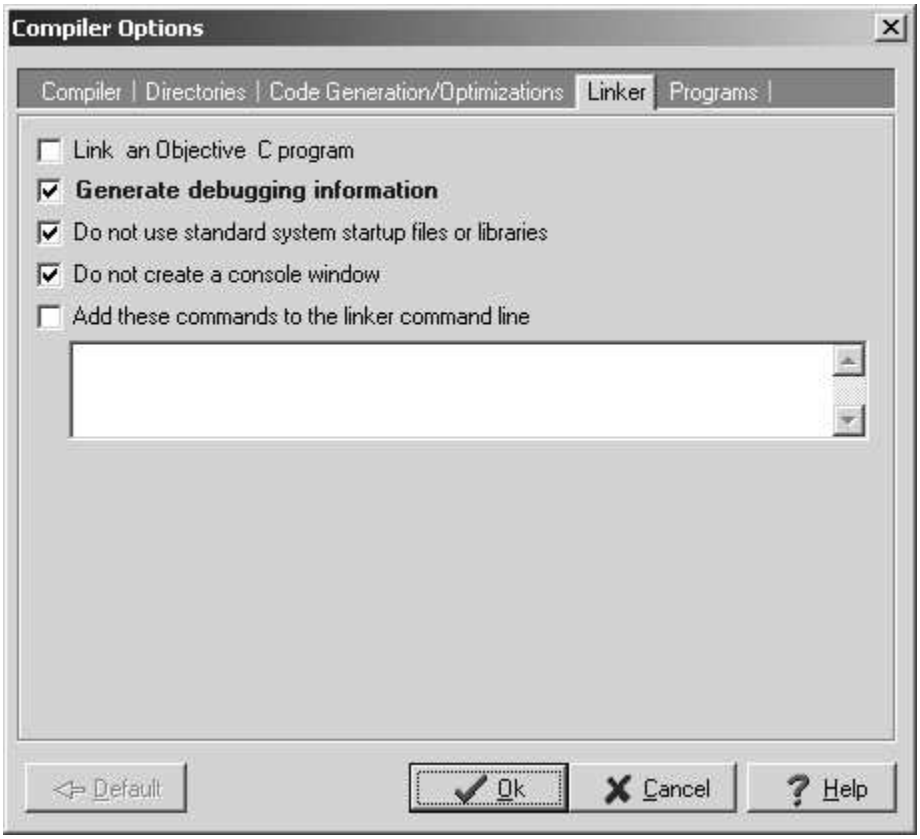
Directory Name in Dev C++	My path	Notes
Binaries	C:\Cygwin\usr\arm-elf\bin	This will be where the file "arm-elf-gcc.exe" (or similar is located)
Libraries	C:\Cygwin\usr\lib\gcc-lib\arm-elf\2.95.2	The file "libgcc.a" will be located in this directory (be sure to pick the one for the target processor)
C Includes	C:\Cygwin\usr\arm-elf\include	The file "stdio.h" (along with the rest of the "standard" C includes) will be located in this directory (be sure to pick the one for the target processor)
C++ Includes	C:\Cygwin\usr\arm-elf\include\g++	The file "iostream.h" (along with the rest of the "standard" C++ includes) will be located in this directory (be sure to pick the one for the target processor)

Code Generation/Optimizations

The user can also configure the optimizations used by the compiler and linker in this panel; I assume most of these command line arguments will be the same regardless of the target processor. Currently I have only selected "Do not optimize" for now at least…

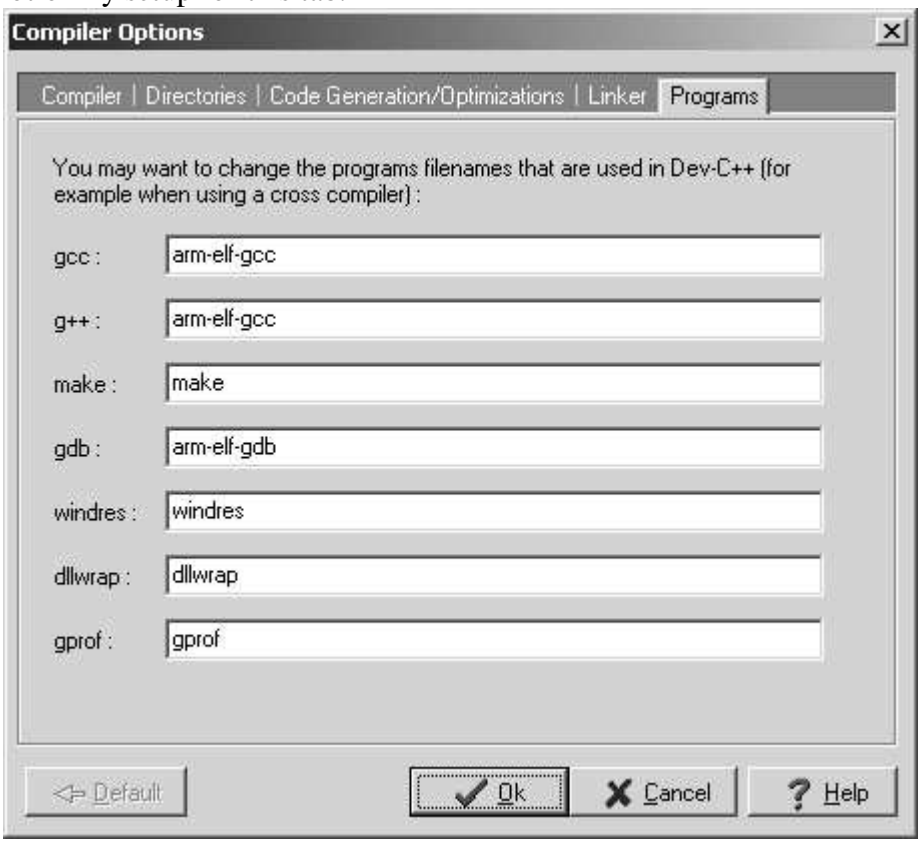
Linker

This panel is used to interface directly with the linker (there is also a linker panel in the project options which does the same thing – but only for the current project). In this panel be sure to select "Do not use standard system startup files or libraries", because (from what I understand) this will attempt to link in libs for he host not the target processor. I also selected "Do not create a console window" because it is not applicable to an embedded application. Below is a screen shot of my configuration.



Programs

The program tab is used to override the default program names used by Dev C++. The names for most of the embedded processors will have the name of the processor as part of the name, for example the ARM version of GCC is called “arm-elf-gcc.exe”. Below is a screen shot of my setup for this tab.



After the programs are setup the Dev C++ should be able to generate an executable for the target processor. Next we will investigate what is required to get a project to work with eCOS and some external libraries.

Important Note (Bug workarounds)

In version 4.9.5.0 of Dev C++ there is a very annoying bug that makes the program about useless as an editor, this is a documented bug and will surely be fixed in an upcoming release, but for now the Code completion and Source Browsing should be disabled. De-Select “Enable class browser” from the Tools | Editor Options panel. Also be sure to de-select “Enable code-completion” from the Tools | Editor Options panel.

Also it should be noted that there must be an entry in the all of the directories or the program will not even start to compile the program. If the user wishes not to add a

default “global” directory to the options, include a valid directory that does not contain any valid files, like “C:/” (Use the UNIX file separator – see below).

Also note that Dev C++ appears to support both the UNIX file separator “/” and the DOS separator “\”, but do not end a path with the DOS file separator “\” or you will get an error about an “unmatched quote”, use no ending separator or use the UNIX version.

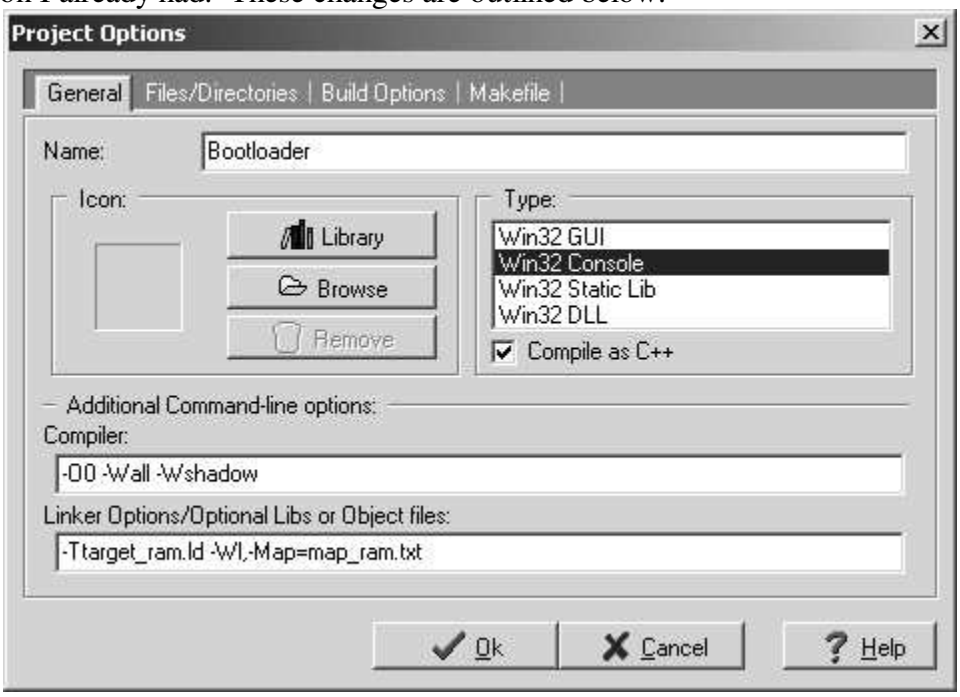
Also, currently there is no way to change the executable filename, therefore the executable has the normal PC extension of “*.exe”, if this causes a problem with other systems, be sure to change the extension before continuing.

Setting up the Project Options

This section outlines the options required for a simple application for the ARM.

General

In the general tab select a name for this project (this is used at the root of the tree view). Then be sure to select Win32 Console so we do not attempt to link in the GUI stuff for Windows. I also added the compiler options from the working make file from the application I already had. These changes are outlined below.



The compiler options tell the complier to generate warnings for everything (-Wall), and to generate warnings for variables with the same name that are used as both locals and global (-Wshadow), and, finely, to do no optimization (-O0).

The linker options specify a linker description file which is located in the same directory as source files and the project (-Ttarget_ram.ld), and to generate a map file (-Wl,-Map=map_ram.txt).

Files/Directories

Under the files and directories tab you can enter where your libraries and includes are for the specific project, if they are different then the ones for the entire program. This example did not require these options, so they were left blank.

Build Options

Under the build options you can specify a folder to place the object and final executable files. Note these folders must be generated before evoking the compiler; the current version will not generate these directories. I did not enter anything in these fields, so my object files and output executable were placed in the same directory as the source.

Makefile

Here you can add information to the makefile for your own specific uses, because this was not required by this application, it was not included.

Setting up a Project

The final test to be sure the entire environment is setup properly is to actually make a project and to see if it generates the correct executable.

The first step in creating a project is to generate some source files (I was lucky enough to have an application that already compiled and worked with the GCC compiler). I took this application and found the source files and added them to the Dev C++ project. Selected Execute | Rebuild All, and an executable was generated….