Compiling the WP 34S / WP 31S Binaries on Linux

You need at least two compilers, preferably three:

- A native C compiler to compile the Qt emulator for your Linux distribution,
- the Yagarto GCC compiler to compile most of the flash image,
- and preferably another, faster GCC to compile a few parts of the flash image.

Compiling the Qt emulator is the easiest part, you only need the standard Qt development tools and libraries. On Debian systems and its derivatives (e.g. Ubuntu, Mint) it means the qt4-dev-tools package and its dependencies. On non-Debian Linux distributions it should be similarly easy to get everything installed but the name of the package may be different. Then you only need to run 'make' in the QtGui directory of either the 34S or the 31S. You will probably need to configure the QMAKESPEC environment variable first. For instance, see:
http://stackoverflow.com/questions/5303634/qmake-qmakespec-has-not-been-set

To compile the flash image, you want to use Yagarto GCC 4.6.0. Other builds of GCC and even other versions of the Yagarto package are known to miscompile the flash image. It cannot be ruled out that some other GCC builds might work but so far nobody has found such a GCC. The miscompiled binaries aren't easy to notice: often the problems are quite subtle. Steer clear of it unless you have the skills and inclination to debug compiler bugs. Version 20110429 of Yagarto is known to work well and can be downloaded from here:
http://sourceforge.net/projects/yagarto/files/YAGARTO%20for%20Windows/20110429/

The Yagarto package is Windows software so you'll be installing it under Wine. No special configuration is necessary. It will install under $HOME/.wine/drive_c/Program Files/yagarto but the installer should add the necessary configuration to wine so that binaries like arm-none-eabi-gcc will be found even if the whole path isn't specified. In order to make Linux processes find the Wine binaries and be able to invoke them, create a $HOME/bin directory if it doesn't already exist and put a helper script there:

Contents of $HOME/bin/arm-none-eabi-gcc:

```
#!/bin/sh
exec wine "${0##*/}" "$@"
```

This invokes Wine and executes the same command in the context of Wine as the script's own name, with the same parameters the script received. In case you aren't familiar with shell substitutions: ${A##*/} strips everything from the beginning of variable A up to the last forward slash. Variable 0 (zero) is the 0th argument, which is the name of the program as it was invoked, so ${0##*/} is the last component (the file name) in the path of the invoked program. "$@" simply repeats all command line arguments.

Then link the script to a few other names you'll need, here are the necessary commands:

```
$ ln -s arm-none-eabi-gcc ~/bin/arm-none-eabi-ar
```

```
$ ln -s arm-none-eabi-gcc ~/bin/arm-none-eabi-nm
$ ln -s arm-none-eabi-gcc ~/bin/arm-none-eabi-objcopy
$ ln -s arm-none-eabi-gcc ~/bin/arm-none-eabi-ranlib
```

By default, many Linux distributions, e.g. recent versions of Debian, add $HOME/bin to your PATH if the directory exists when you login. After you're done with the above, try logging in again, or just open a new terminal, and see if you can invoke arm-none-eabi-gcc:

```
$ arm-none-eabi-gcc
arm-none-eabi-gcc.exe: fatal error: no input files
compilation terminated.
```

If you did everything correctly, it'll work, and as you can see, it's the Windows .exe program that's executed. If it doesn't work, you may have to add the $HOME/bin directory to your path, for example by adding the following at the end of your $HOME/.profile file (if it doesn't already exist, create one):

```
# set PATH so it includes user's private bin if it exists
if [ -d "$HOME/bin" ] ; then
      PATH="$HOME/bin${PATH:+:$PATH}"
fi
```

Now you should be able to run 'REALBUILD=1 make' in the 34S or 31S directory and build a flash image, but don't do it just yet. Invoking programs through Wine has a lot of overhead and it becomes painfully slow if you have to do it very often, which is what happens when the library of constants is compiled, because GCC is invoked very many times to compile lots of small files one by one.

Fortunately those files only contain data and aren't miscompiled by other versions of GCC. So what you want to do is install another ARM GCC package that runs natively on your Linux system, which will be much faster than Yagarto. The CodeSourcery ARM EABI 2011.09 package is known to work but in all likelihood any GCC targeting ARM would do. Be sure to get the EABI version, otherwise the names of the executables will be different and the below script won't work. Install it and if it added anything to your PATH (typically in the .profile file), remove it because you really don't want that GCC to be invoked by default.

The CodeSourcery installer for Linux can be downloaded from here:
https://sourcery.mentor.com/public/gnu_toolchain/arm-none-eabi/arm-2011.09-69-arm-none-eabi.bin
Alternatively, you can download and manually unpack this smaller TAR archive:
https://sourcery.mentor.com/public/gnu_toolchain/arm-none-eabi/arm-2011.09-69-arm-none-eabi-i686-pc-linux-gnu.tar.bz2

Add a new script in your $HOME/bin directory. Here's what should be in $HOME/bin/arm-none-eabi-gcc-fast (4 lines):

```
#!/bin/sh
```

```
FAST_PATH="$HOME/CodeSourcery/Sourcery_CodeBench_Lite_for_ARM_EABI/bin"
NAME="${0##*/}"
PATH="$FAST_PATH${PATH:+:$PATH}" exec "${NAME%-fast}" "$@"
```

Then link the script to another name you'll need, here is the necessary command:

```
$ ln -s arm-none-eabi-gcc-fast ~/bin/arm-none-eabi-objcopy-fast
```

Obviously, replace $HOME/CodeSourcery/Sourcery_CodeBench_Lite_for_ARM_EABI/bin in the script with whatever location the ARM GCC package got installed to if it's different in your system. Now you can invoke the second GCC by using the arm-none-eabi-gcc-fast command. Try it and observe that it's much faster than the Yagarto GCC.

In order to make the build process take advantage of the faster GCC (but only for compiling the constants!), you need to modify compile_consts.c. The following patch works for both the 34S and the 31S:

```
diff -ur a/compile_consts.c b/compile_consts.c
--- wp34s_r3748/compile_consts.c    2014-11-15 13:18:54.000000000 -0500
+++ wp34s_compile_consts_faster_r3748_20150209/compile_consts.c  0
@@ -508,7 +508,12 @@

              output(fm, cnsts[n].name, &y);
       }
-    fprintf(fm, "\n\n.SILENT: $(OBJS)\n\n"
+    fprintf(fm, "\n\nifdef REALBUILD\n"
+            "%%.o: %%.c\n"
+            "\t$(CC)-fast -c $(CFLAGS) -o $@ $<\n"
+            "\t$(OBJCOPY)-fast --remove-section=.ARM.attributes $@\n"
+            "endif\n"
+            "\n.SILENT: $(OBJS)\n\n"
                  "all: $(OBJS)\n"
                  "\t@rm -f ../%slibconsts.a\n"
                  "\t$(AR) q ../%slibconsts.a $(OBJS)\n"
```

The .ARM.attributes section is best removed because some newer versions of GCC generate ARM attributes that the older Yagarto tools cannot process. Those attributes make no difference for the files that only contain the constants.

If you copy and paste the above patch, you may need to instruct the patch utility to ignore whitespaces. On the command line, it's usually the -l (lower case L) option. The patch can also be downloaded from the following URL:
https://drive.google.com/file/d/0B9Um5zMfcUc6YnhnMUVUUzFIZFE/view?usp=sharing

All referenced files can also be obtained from the following location:
https://drive.google.com/folderview?id=0B9Um5zMfcUc6YnBLUVFKZnlrVXM&usp=sharing