Let's create a GDC toolchain for ARM Cortex-M microcontrollers

In order to build our GCC+GDC toolchain, we will need these sources:

1: binutils
2: gcc
3: gmp
4: mpc
5: mpfr
6: gdc
7: newlib
8: cortex-m0-trap.S.patch
9: t-arm-elf.patch

In order to be able to build gcc (gcc-4.6 later), we will also need gmp, mpc and mpfr.
These should not be compiled first.
Instead of compiling gmp, mpc and mpfr, we make symbolic links to the sources from within gcc's source directory. This speeds up build time dramatically and avoids a lot of problems.

We will download the archives and clone the git repositories only ***once***, in order to be polite to the servers.

This step-by-step guide has been designed so you can choose from several ways to use it:
1: Put it all in a script and run it.
2: Copy a box containing commands, then paste them into your terminal/shell (won't work with all systems)
3: Copy one line at a time and execute it (this should work for most people, though)

Pink boxes should be written directly to a file; nothing they contain should be entered into a shell.
Green boxes should be done on every build.
Cyan boxes can be skipped after first build.
Yellow boxes contains information only; nothing from those should be copied or typed.

If you don't want to copy and paste from this PDF, you can download a build-script from the following URL:

http://d.gpio.dk/dl/b

You'll prbably need to make the script executable. I also recommend making a convenience-alias:

```
chmod 755 b; alias b='./b'
```

I'd like to thank the following people for their hard work, which made this possible:

Timo Sintonen
Michael V. Franklin
Johannes Pfau
Iain Buclaw
And of course, the entire D compiler developer team

I know they all worked very hard on this for a very long time.

Please visit http://dlang.org for more information on the D language, including tutorials and documentation.

When you've gotten the toolchain working, I've made some startup files for a few microcontrollers, which you may find useful. See http://d.gpio.dk/ for more information.

Files

File: t-arm-elf (work by Timo Sintonen - an alternative to my t-arm-elf):

```
MULTILIB_OPTIONS  += mcpu=cortex-m0/mcpu=cortex-m3/mcpu=cortex-m4 mfloat-abi=hard
mfpu=fpv4-sp-d16
MULTILIB_DIRNAMES += cortex-m0 cortex-m3 cortex-m4
MULTILIB_REQUIRED += mcpu=cortex-m0
MULTILIB_REQUIRED += mcpu=cortex-m3
MULTILIB_REQUIRED += mcpu=cortex-m4/mfloat-abi=hard/mfpu=fpv4-sp-d16
MULTILIB_EXTRA_OPTS += mthumb
```

File: t-arm-elf (my own version; I do not know which one is best yet, but it's included for completeness):

```
MULTILIB_OPTIONS     = marm/mthumb
MULTILIB_DIRNAMES    = arm thumb
MULTILIB_EXCEPTIONS  = marm* mthumb
MULTILIB_OPTIONS    += mcpu=arm7tdmi-s/mcpu=cortex-m0/mcpu=cortex-m3/mcpu=cortex-m4
MULTILIB_DIRNAMES   += arm7tdmi-s cortex-m0 cortex-m3 cortex-m4
MULTILIB_EXCEPTIONS += mcpu=arm7tdmi-s* mcpu=cortex-m0* mcpu=cortex-m3* mcpu=cortex-m4*
MULTILIB_OPTIONS    += mfloat-abi=hard mfpu=fpv4-sp-d16
MULTILIB_DIRNAMES   += float-abi-hard fpuv4-sp-d16
MULTILIB_EXCEPTIONS += mfloat* mthumb/mfloat*
MULTILIB_EXCEPTIONS += mfpu* mthumb/mfpu*
MULTILIB_EXCEPTIONS += mthumb/mcpu=cortex-m4/mfloat-abi=hard
MULTILIB_EXCEPTIONS += mthumb/mcpu=cortex-m4/mfpu=fpv4-sp-d16
MULTILIB_EXCEPTIONS += *arm7tdmi-s*mfloat-abi* *arm7tdmi-s*mfpu*
MULTILIB_EXCEPTIONS += *cortex-m3*mfloat-abi* *cortex-m3*mfpu*
MULTILIB_EXCEPTIONS += *cortex-m0*mfloat-abi* *cortex-m0*mfpu*
```

File: cortex-M0-trap.S.patch (my own patch; this can also be downloaded from my server):

```
--- gcc-4.8.3/newlib/libc/sys/arm/trap.S      2014-08-20 19:15:35.000000000 +0200
+++ gcc-4.8.3/newlib/libc/sys/arm/trap.S      2014-08-20 19:17:42.000000000 +0200
@@ -1,5 +1,5 @@
         /* Run-time exception support */
-#if !defined(__thumb2__)
+#if !defined(__thumb2__) && !defined(__ARM_ARCH_6M__)
 #include "swi.h"

 /* .text is used instead of .section .text so it works with arm-aout too.  */
```

Build

The rest of this document contains build commands, which you can enter in the terminal.
Feel free to recycle anything you like in your own scripts.

Note: On a PowerMac with Mac OS X 10.5.8, newlib will fail to install on first attempt.
This is due to a folder will be missing - but it's created after it's needed, so I attempt to install newlib twice.
Second install always succeed for me.

If using these instructions as a script, you can supply the following parameters:

```
b [workdir] [prefix]
```

eg.

```
./b ~/temp /usr/local
```

-If not specifying a prefix, an 'Install' folder will be created in your workdir. The default workdir is 'toolchain'

Build configuration:

```
target="arm-none-eabi"
workdir="$HOME/toolchain"; [ "$1" ] && workdir="$1"
[ "$toolchain" ] || toolchain="$target"; [ "$3" ] && toolchain="$3"
[ "$prefix" ] || prefix="$workdir/Install/usr/local"; [ "$2" ] && prefix="$2"
```

Binutils configuration:

```
binutils_cfg=(--disable-nls --disable-libssp --with-gcc --with-gnu-as)
binutils_cfg=(${binutils_cfg[@]} --with-gnu-ld --enable-multilib --with-system-zlib)
binutils_cfg=(${binutils_cfg[@]} --enable-interwork --enable-plugins)
```

GCC configuration:

```
gcc_optional=(--with-cpu=cortex-m4 --with-tune=cortex-m4 --with-mode=thumb)
headers=--with-headers="$source/newlib-2.10.0/newlib/libc/include/"
gcc_cfg=(--disable-nls --disable-libssp --with-gnu-as --with-gnu-ld --with-system-zlib)
gcc_cfg=(${gcc_cfg[@]} --with-gcc --with-dwarf2 --enable-interwork --with-newlib)
gcc_cfg=(${gcc_cfg[@]} --disable-libphobos --disable-decimal-float --disable-libffi)
gcc_cfg=(${gcc_cfg[@]} --disable-libmudflap --disable-libquadmath --disable-shared)
gcc_cfg=(${gcc_cfg[@]} --disable-libgomp --disable-threads --disable-tls)
gcc_cfg=(${gcc_cfg[@]} --disable-libstdcxx-pch --disable-bootstrap ${gcc_optional[@]})
gcc1_cfg=(${gcc_cfg[@]} --enable-languages="c,c++,lto,d" --without-headers)
#gcc2_cfg=(${gcc_cfg[@]} --enable-languages="c,c++,lto,d" --with-headers)
```

Newlib configuration:

```
newlib_cfg=(--enable-interwork --enable-multilib --disable-libssp --disable-nls)
newlib_cfg=(${newlib_cfg[@]} --enable-newlib-io-long-long --enable-newlib-register-fini)
newlib_cfg=(${newlib_cfg[@]} --disable-newlib-supplied-syscalls)
```

Prepare other variables and directories:

```
start=`date +%s`; let cpucount=2*`getconf _NPROCESSORS_ONLN`; pmake="make -j$cpucount"
senv="true";[ -w "$prefix" ] && smake="$pmake" || smake="sudo $pmake" && senv="sudo env"
case `uname -s` in 'Darwin') isMac=1; smake="sudo $pmake" ;; 'Linux') isLinux=1 ;; esac
prefix="$prefix/$toolchain"; build="$workdir/Build"
source="$workdir/Source"; archives="$workdir/Archives"
bin="$workdir/bin"; export PATH="$bin:$PATH"; mkdir -p "$build" "$source" "$bin"
[ -d /arc ] && [ ! -e "$archives" ] && ln -s /arc "$archives"; mkdir -p "$archives"
```

Set up convenience functions:

```
cls(){ printf "\033c"; }; hr(){ eval printf '=%.0s' {1..$(tput cols)}; echo; }
good(){ printf "\e[1;32m$*\e[m\n"; }; bad(){ printf "\e[1;31m$*\e[m\n"; }
msg(){ printf "\e[1;34m$*\e[m\n"; }; msghr(){ msg "$@\n`hr`"; }
success(){ good "Success$pkg"; }; status="success || failure"
failure(){ bad "Failed$pkg $*"; [ "" == "$LINES" ] && exit 1; }
ipkg(){ local d="${1%%-*}"; pkg=" ($d)"; msghr "install $d"; mkdir -p "$build/$d"; }
sl(){ for f in ${@:2}; do ln -s "$source/$f" "$source/$1/${f%%-*}" 2>/dev/null; done; }
```

Set up build functions:

```
cfg(){ ipkg "$1"; cd "$build/${1%%-*}" && "$source/$1/configure" ${@:2} || failure; }
build(){ cfg $@; $pmake && $smake install && success || failure; unset pkg; }
targ(){ x=-xaf; case $1 in *.gz) x=-xzf ;; *.bz2) x=-xjf ;; *.xz) x=-xJf ;; esac; }
utar(){ local x; targ "$1"; tar "$x" "$1" || failure extracting "${1##*/}"; }
dpk(){ local f="${1##*/}"; [ -d "$source/${f%.tar*}" ] || utar "$1"; }
dload(){ curl -Lk --retry 3 "$1" -o "$2" || failure downloading ${1##*/}; }
dl(){ local a="$archives/${1##*/}"; [ -f "$a" ] || dload "$1" "$a"; }
dlc(){ unset pkg; local c="$DL_CACHE"; [ "$c" ] && f="$c/${1##*/}" || f="$1"; dl "$f"; }
dlx(){ dlc $1; cd "$source" && dpk "$archives/${1##*/}"; }
dlxb(){ dlx $1; local f=${1##*/}; f=${f%.*}; build ${f%.tar} ${@:2}; }
```

Set up a temporary (local) bin directory for GCC-4.2 (needed for PowerMac; we can't use GCC-4.0):

```
which gcc-4.2 >/dev/null && gcc_symlink=1
[ $gcc_symlink ] && ln -s "/usr/bin/gcc-4.2" "$bin/gcc" 2>/dev/null
[ $gcc_symlink ] && ln -s "/usr/bin/gcc-4.2" "$bin/gcc-4.2" 2>/dev/null
[ $gcc_symlink ] && ln -s "/usr/bin/gcov-4.2" "$bin/gcov" 2>/dev/null
[ $gcc_symlink ] && ln -s "/usr/bin/g++-4.2" "$bin/g++" 2>/dev/null
```

Download archives and extract sources:

```
msghr "Downloading and extracting..."; cd "$archives"
[ -e gdc ] || git clone --mirror git://github.com/D-Programming-GDC/GDC.git gdc
dlx "ftp://ftp.gnu.org/gnu/binutils/binutils-2.25.tar.bz2"
dl  "http://ftp.gnu.org/pub/gnu/gcc/gcc-4.9.2/gcc-4.9.2.tar.bz2"
dlx "ftp://ftp.gmplib.org/pub/gmp-5.1.3/gmp-5.1.3.tar.bz2"
dlx "http://www.mpfr.org/mpfr-3.1.2/mpfr-3.1.2.tar.bz2"
dlx "http://www.multiprecision.org/mpc/download/mpc-1.0.3.tar.gz"
dlx "http://isl.gforge.inria.fr/isl-0.12.2.tar.bz2"
dlx "http://www.bastoul.net/cloog/pages/download/count.php3?url=./cloog-0.18.1.tar.gz"
dlx "http://www.mr511.de/software/libelf-0.8.13.tar.gz"
dl  "ftp://sources.redhat.com/pub/newlib/newlib-2.1.0.tar.gz"
dlx "http://ftp.gnu.org/pub/gnu/gdb/gdb-7.9.tar.gz"
dlc "http://dl.gpio.dk/t-arm-elf-4.9.2.patch"
dlc "http://dl.gpio.dk/cortex-m0-trap.S.patch"
```

Update mirrored repository caches:

```
cd "$archives/gdc" && git remote update
```

Prepare GCC for multiple ARM libraries and archive the patched sources (in case we mess up):

```
msghr "Preparing GCC..."; gcc="gcc-4.9.2"; cd "$source"
gccp="$gcc+patch"; p="$archives/t-arm-elf-4.9.2.patch"
f="$archives/$gccp.tar.bz2"; [ -f "$f" ] && dpk "$f"
[ -e "$gccp" ] || dpk "$archives/$gcc.tar.bz2"
[ -e "$gccp" ] || { ( cd "$source/$gcc/" && patch -p1 < "$p" ) && mv "$gcc" "$gccp"; }
[ -e "$f" ] || tar -cjf "$f" "$gccp"
```

Fix Cortex-M0 exception bug in newlib and archive the patched sources (in case we mess up):

```
msghr "Preparing newlib..."; nl="newlib-2.1.0"; cd "$source"
nlp="$nl+patch"; p="$archives/cortex-m0-trap.S.patch"
f="$archives/$nlp.tar.bz2"; [ -f "$f" ] && dpk "$f"
[ -e "$nlp" ] || dpk "$archives/$nl.tar.gz"
[ -e "$nlp" ] || { ( cd "$source/$nl/" && patch -p1 < "$p" ) && mv "$nl" "$nlp"; }
[ -e "$f" ] || tar -cjf "$f" "$nlp"
```

Prepare GCC prerequisites:

```
rm -f "$source/gcc"; ln -s "$source/gcc-4.9.2+patch" "$source/gcc" 2>/dev/null
sl gcc gmp-5.1.3 mpfr-3.1.2 mpc-1.0.3 libelf-0.8.13 isl-0.12.2 cloog-0.18.1
sl newlib-2.1.0/newlib
```

Clone from mirrored repositories:

```
cd "$source"
( rm -Rf "$source/gdc"; git clone "$archives/gdc" && cd gdc && git checkout gdc-4.9 )
```

Duplicate GCC sources:

```
msghr "Duplicating GCC sources"; cd "$source"
rm -Rf "$source/gcc-4.9.2+gdc"; cp -R "gcc-4.9.2+patch" "$source/gcc-4.9.2+gdc"
```

Prepare GCC for GDC integration:

```
msghr "Setting up GCC with GDC..."; cd "$source/gdc"
./setup-gcc.sh ../gcc-4.9.2+gdc
```

Remove any previously installed toolchain of the same kind, to avoid conflict:

```
msghr "Removing old toolchain..."; cd "$workdir"; sudo rm -Rf "$prefix"
```

Build and install binutils:

```
sl binutils gmp-5.1.3 mpfr-3.1.2 mpc-1.0.3 libelf-0.8.13 isl-0.12.2 cloog-0.18.1
sl binutils newlib-2.1.0/newlib
build "binutils-2.25" --target="$target" --prefix="$prefix" ${binutils_cfg[@]}
```

Point to GCC+GDC source, remove old build directory and build GCC+GDC without bootstrap
(We would of course like to have a two-step build, but I have not found a way to do so yet)

```
rm -f "$source/gcc"; ln -s "$source/gcc-4.9.2+gdc" "$source/gcc"
rm -Rf "$build/gcc"; mkdir -p "$build/gcc"; cd "$build/gcc"
cfg "gcc" --target="$target" --prefix="$prefix" ${gcc1_cfg[@]}
$pmake all-gcc && $pmake all-target-libgcc && $pmake all-target-libstdc++-v3 || failure
$smake install-gcc && $smake install-target-libgcc || failure
$smake install-target-libstdc++-v3 && $status
```

Add new $prefix/bin to $PATH, so the new GCC will be found:

```
export PATH="$PATH:$prefix/bin"
```

Show supported multilib (you most likely get more than those shown):

```
msghr "Supported multilib:"; arm-none-eabi-gcc -print-multi-lib; msghr
```

```
.;@mthumb
cortex-m0;@mcpu=cortex-m0@mthumb
cortex-m3;@mcpu=cortex-m3@mthumb
cortex-m4;@mcpu=cortex-m4@mthumb
```

Build newlib:

```
rm -Rf "$build/newlib"; mkdir -p "$build/newlib"; cd "$build/newlib"
cfg "newlib-2.1.0+patch" --target="$target" --prefix="$prefix" ${newlib_cfg[@]}
cflags="-D__IEEE_BIG_ENDIAN -D__IEEE_BYTES_LITTLE_ENDIAN -D__BUFSIZ__=64"
$pmake CFLAGS_FOR_TARGET="$cflags" || failure
$senv "PATH=$PATH" $pmake install || $senv "PATH=$PATH" $pmake install && $status
```

Build final GCC+GDC if necessary:

```
[ "$gcc2_cfg" ] && cd "$build/gcc"
[ "$gcc2_cfg" ] && cfg "gcc" --target="$target" --prefix="$prefix" ${gcc2_cfg[@]}
[ "$gcc2_cfg" ] && $pmake all $smake install && $status
```

Show build time:

```
end=`date +%s`; let deltatime=end-start; let hours=deltatime/3600
let minutes=(deltatime/60)%60; let seconds=deltatime%60
msg `printf "Build time: %d:%02d:%02d\n" $hours $minutes $seconds`
```

Test that your compiler works; after this, you can add it to your $PATH in ~/.bashrc ...

```
$prefix/bin/$target-gdc --version && echo "Success. Now add '$prefix/bin:' to your PATH"
```