Übung 6: Programme und Kernel für RasPi Cross-Compilieren

Voraussetzung: Linux-System (Debian, Ubuntu, SuSE, Fedora, ...), das 32-bit Binaries starten kann (ansonsten müsste die 64-bit Variante des Cross-Compilers verwendet werden, die hier aber nicht verlinkt ist).

Anleitung nach: http://lab.ks.uni-freiburg.de/projects/linux_from_scratch/wiki/RaspberryPI_Cross-Compiling

1. Download des raspberry Cross-Compilers per git:

   `git clone https://github.com/raspberrypi/tools.git`
   oder (Kurs):
   `wget http://172.30.0.128/Windischgarsten/Cross/cross-toolchain.tar.xz`

2. Auspacken mit:
   `tar -Jxvf  cross-toolchain.tar.xz`

3. Such-Pfad setzen (erst mal nur für die aktuelle Session):

```
export PATH=$PATH:$HOME/tools/arm-bcm2708/arm-bcm2708-linux-gnueabi/bin/
export CROSS_COMPILE=arm-bcm2708-linux-gnueabi-
export ARCH=arm
```

4. Test des Compilers:
   `arm-bcm2708-linux-gnueabi-gcc -v`
   es sollte erscheinen:

```
Using built-in specs.
COLLECT_GCC=arm-bcm2708-linux-gnueabi-gcc
COLLECT_LTO_WRAPPER=/home/tbk/projekt/raspberrypi/tools/arm-bcm2708/arm-bcm2708-linux-gnueabi/bin/../libexec/gcc/arm-bcm2708-linux-gnueabi/4.7.1/lto-wrapper
Target: arm-bcm2708-linux-gnueabi
Configured with: /home/extra/crosstool/stagingsf/.build/src/gcc-linaro-4.7-2012.04/configure --build=i686-build_pc-linux-gnu --host=i686-build_pc-linux-gnu --target=arm-bcm2708-linux-gnueabi --prefix=/home/dc4/tools/arm-bcm2708/arm-bcm2708-linux-gnueabi --with-sysroot=/home/dc4/tools/arm-bcm2708/arm-bcm2708-linux-gnueabi/arm-bcm2708-linux-gnueabi/sysroot --enable-languages=c,c++ --with-cpu=arm1176jzf-s --with-tune=arm1176jzf-s --with-float=softfp --with-pkgversion='crosstool-NG 1.15.2' --enable-__cxa_atexit --disable-libmudflap --disable-libgomp --disable-libssp --with-gmp=/home/extra/crosstool/stagingsf/.build/arm-bcm2708-linux-gnueabi/buildtools --with-mpfr=/home/extra/crosstool/stagingsf/.build/arm-bcm2708-linux-gnueabi/buildtools --with-mpc=/home/extra/crosstool/stagingsf/.build/arm-bcm2708-linux-gnueabi/buildtools --with-ppl=/home/extra/crosstool/stagingsf/.build/arm-bcm2708-linux-gnueabi/buildtools --with-cloog=/home/extra/crosstool/stagingsf/.build/arm-bcm2708-linux-gnueabi/buildtools --with-libelf=/home/extra/crosstool/stagingsf/.build/arm-bcm2708-linux-gnueabi/buildtools --with-host-libstdcxx='-static-libgcc -Wl,-Bstatic,-lstdc++,-Bdynamic -lm -L/home/extra/crosstool/stagingsf/.build/arm-bcm2708-linux-gnueabi/buildtools/lib -lpwl' --enable-threads=posix --enable-target-optspace --disable-nls --disable-multilib --with-local-prefix=/home/dc4/tools/arm-bcm2708/arm-bcm2708-linux-gnueabi/arm-bcm2708-linux-gnueabi/sysroot --enable-c99 --enable-long-long --with-float=softfp
Thread model: posix
gcc version 4.7.1 20120402 (prerelease) (crosstool-NG 1.15.2)
```

5. Wir programmieren „Hallo, Welt!“ in C:

```
#include <stdio.h>
int main(int argc, char**argv){
 puts("Hallo, Welt!");
 return 0;
}
```

Quelltext 1: hallo.c

   Compilieren (`-s` = strip (Symboltabellen entfernen), `-Os` = optimieren nach Größe, `-Wall` = alle Syntax- und Semantik-Warnungen aktivieren, `-o` = Ergebnis speichern in Datei):

   ```
   arm-bcm2708-linux-gnueabi-gcc -Os -Wall -s -o hallo hallo.c
   ```

6. Untersuchen des resultierenden Binaries:

   ```
   file -sk hallo
   ```

   ```
   arm-bcm2708-linux-gnueabi-ldd --root --root
   /home/knopper/tools/arm-bcm2708/arm-bcm2708-linux-gnueabi hallo
   ```

7. Hochladen auf RasPi: am besten per ssh/scp:
   `scp hallo pi@172.30.0.XXX:`

8. Einloggen auf RasPi:
   `ssh pi@172.30.0.XXX`

9. Ausführen des „`hallo`“:
   `./hallo`
   oder
   `/home/pi/hallo`
   (sollte `hallo` noch nicht ausführbar sein, vorher: `chmod +x hallo`)

Kernel cross-compilieren

Ähnlich wie unter
http://www.raspberrypi.org/documentation/linux/kernel/building.md beschrieben

(auf dem RasPi selbst auch möglich, aber sehr langsam → Cross-Compiler auf Hostsystem verwenden!)

1. `Crosscompiler-Toolcain installieren (s.o., schon erledigt)`

2. `Kernel-Sourcen holen:`
   `git clone --depth=1 https://github.com/raspberrypi/linux`
   `(schon entpackt)`
   `ODER (Kurs)`
   `wget http://172.30.0.128/Windischgarsten/Cross/raspi-linux.tar.gz`
   `tar zxvf raspi-linux.tar.gz`

3. `Ggf. Kernel-Patches installieren (download, zcat … | patch), s.a.`
   `http://www.raspberrypi.org/documentation/linux/kernel/patching.md`

4. cd raspberrypi-linux-*

5. Mit Default-Config:
   make ARCH=arm CROSS_COMPILE=arm-bcm2708-linux-gnueabi- bcmrpi_defconfig
   ODER mit aktueller Konfiguration:
   ssh pi@raspi-adresse *zcat /proc/config.gz* > .config

6. Konfiguration:
   ACHTUNG: Bei Ubuntu ist die Developer-Library für Konsolenprogramme (libncurses) nicht standardmäßig installiert!
   → sudo apt-get install libncurses5-dev

   make ARCH=arm CROSS_COMPILE=arm-bcm2708-linux-gnueabi- menuconfig
   Alternativ:
   GNOME: make … gconfig (benötigt GTK2-Libraries!)
   KDE: make … xconfig (benötigt QT-Libraries)

7. Compilation:
   make ARCH=arm CROSS_COMPILE=arm-bcm2708-linux-gnueabi-

8. Einige embedded Boards benötigen eine Device Tree Table, die die Hardware-Adressierung in C-ähnlichem Sourcecode enthält. Diese DTS-Datei muss für das jeweilige Board mit dem Tool „dtc“ in eine .dtb (Binary) compiliert werden, und zusammen mit dem Kernel auf die Bootpartition installiert werden.
   Die DTS-Sourcen liegen im Kernel-Source unter **arch/arm/boot/dts/*.dts** vor. Man kann in der Kernel-Konfiguration auch angeben, dass das .dtb-Compilat im Kernel selbst angehängt wird, um die zusätzliche Datei zu sparen, das Anhängen passiert aber i.d.R manuell (mit cat datei.dtb >> zImage).