Erstellen einer Linux-Distribution für das Lyrtech SFF-SDR

Jan Krämer, Dipl.-Ing Michael Schwall

November 2011

Contents

1 Vorwort

Dieses Kompendium dient als Hilfestellung zur Erstellung einer auf das *Lyrtech Small Form Factor Software Defined Radio* (*SFF-SDR*) optimierten offenen Linux Distribution. Bei Mängel oder Fehler in der Dokumentation, diese bitte entweder an kraemer@int.uni-karlsruhe.de (Jan Krämer) oder michael.schwall@kit.edu (Dipl.-Ing Michael Schwall) weiterleiten.

Stand dieses Dokumentes: 20.12.2011

2 Benötigte Software

In diesem Abschnitt finden sie eine Liste der Software, welche man benötigt um eine Linux-Distribution für das *Lyrtech SFF-SDR* zu erstellen. Auf die Installation und Besonderheiten der Programme wird in den folgenden Abschnitten näher eingegangen.

1. Code Sourcery Glite++
2. U-Boot Bootloader
3. OpenEmbedded
4. BitBake

3 Codesourcery

3.1 Einleitung

Codesourcery ist ein freier Cross Compiler. Cross Compiler werden benötigt um Software, welche auf einer Host-Prozessorarchitektur entwickelt wurde, für eine andere Prozessorarchitektur zu kompilieren. In unserem Fall ist der Host PC x86 basiert, und wir entwickeln Software für einen *DaVinci ARM* Prozessor der Firma *Texas Instruments*.

3.2 Installation

Die neuste Version von *Codesourcery* kann bequem über die Seite des Herstellers Mentor bezogen werden [1]. Hier stehen für Linux Zwei Möglichkeiten zum Download bereit. In diesem Kompendium beschränken wir uns auf das *GNU/LINUX TAR* Paket. Ist das Paket heruntergeladen, kann der Ordner ganz einfach über die graphische Oberfläche oder mithilfe des Terminals installiert werden. Dafür wird im Terminal erstmal das Packet mittels bunzip2 entpackt [1].

```
> bunzip2 /path/to/package.tar.bz2
```

Danach wird der Zielordner für die Installation erstellt.

```
> mkdir -p $HOME/CodeSourcery
```

Wechseln in den eben erstellten Ordner.

```
> cd $HOME/CodeSourcery
```

Nun muss die Datei in diesen Ordner entpackt werden.

[1] `https://sourcery.mentor.com/sgpp/lite/arm/portal/release1803`

```
> tar xf /path/to/package.tar
```

Die Installation ist nun komplett. Weiterführende zur Installation Informationen sind in der Getting-Started Manual des Herstellers zu finden [2]

3.3 Einrichten des Compilers

Um den Compiler nutzen zu können muss *Codesourcery* noch zur PATH Umgebungsvariablen hinzugefügt werden. Dies geschieht wieder mithilfe des Terminals.

```
> PATH=$HOME/CodeSourcery/bin:$PATH
> export PATH
```

Desweiteren muss noch die Variable `CROSS_COMPILE` gesetzt werden.

```
> CROSS_COMPILE=arm-none-linux-gnueabi-
> export CROSS_COMPILE
```

4 Das U-Boot

Das *U-Boot* ist ein sogenannter Boot(strap)loader, also ein Programm, welches auf der gewünschten Plattform das fertige Betriebssystem läd und dann startet.

4.1 Installation

Vor der Installation sollte man überprüfen ob sie das Paket *binutils* bereits auf dem System installiert ist. Wenn nicht muss dieses Paket noch installiert werden.

```
> sudo apt-get install binutils
```

Desweiteren wird vorrausgesetzt dass auf dem System ein Crosscompiler, z.B. *Codesourcery*, installiert ist. Die Installation von Codesourcery wurde im vorherigen Kapitel beschrieben. Am einfachsten erhält man den *U-Boot* Bootloader über *GIT*.

```
> git-clone git://www.denx.de/git/u-boot.git u-boot
```

Man muss nun sicher stellen, dass die Adresse im DDR2-RAM des SFF- SDRs, an welche das Programm geladen wird, die rich- tige ist. Dazu muss die Datei *config.mk* bearbeit wer- den. In dieser Datei ist sicherzustellen, dass die Zeile

[2] `https://sourcery.mentor.com/sgpp/lite/arm/portal/doc11473/getting-started.pdf`

$CONFIG\_SYS\_TEXT\_BASE = 0x81080000$ vorhanden ist, da sonst der Bootloader an der falschen Adresse im DDR2- RAM sucht. Nun kann man das *U-Boot* kompilieren. Hierbei wird *U-Boot* extra für das *Lyrtech SFF-SDR* konfiguriert.

```
> make distclean
> make davinci_sffsdr_config
> make
```

Nun wurde eine `U-Boot.bin` Datei erzeugt, welche dann auf dem *DaVinci* als Bootloader fungiert. Desweiteren wurde im *U-Boot* Ordner ein Unterordner namens `Tools` installiert. In diesem Ordner befindet sich das Tool `mkImage.bin`. Mithilfe dieses Tools kann nun später mittels *OpenEmbedded* ein speziell für das *U-Boot* konfigurierter Linux-Kernel kompiliert werden. Dieser Kernel wird dann später durch das *U-Boot* geladen und ausgeführt. Damit *OpenEmbedded* auf das Tool *mkImage* zugreifen kann, muss dieses noch der PATH Umgebungsvariablen hinzugefügt werden.

```
> PATH=~/path/to/u-boot/tools/:$PATH
> export PATH
```

5 OpenEmbedded

OpenEmbedded ist ein Programm zur Erstellen verschiedener Distribution. *OpenEmbedded* baut hierfür auf dem Programm *BitBake* auf. Dafür nutzt es sogenannte Recipes, also Skripte, welche die einzelnen benötigten Schritte zum Erstellen der Distribution beinhalten. Mithilfe dieser Recipes werden dann auch spezielle Kernels mittels *GIT* automatisch aus dem Internet geladen und dann kompiliert. Zu dem jetzigen Zeitpunkt (November 2011) nutzt das CEL OpenEmbedded nur zum Erstellen des Linux-Kernels für das *Lyrtech SFF-SDR*

5.1 Vorbereiten des Host Systems

Bevor man nun *OpenEmbedded* installieren kann, müssen vorher einige Pakete für das Host-Betriebssystem heruntergeladen und installiert werden. Dies kann aber im Normalfall bequem über den Paket Manager bezogen werden. Es werden hier nur die benötigten Pakete inklusive Terminalbefehl aufgelistet. Für weiter Informationen sei auf die *OpenEmbedded* Homepage verwiesen. [3]

```
> sudo apt-get install python-dev
> sudo apt-get install python-psyco
> sudo apt-get install texi2html
> sudo apt-get install cvs
> sudo apt-get install chrpath
> sudo apt-get install texinfo
```

[3] `http://www.openembedded.org/index.php/Getting_started`

```
> sudo apt-get install quilt
> sudo apt-get install pax-utils
> sudo apt-get install libcurl4-openssl-dev 7.21.6-3ubuntu3
> sudo apt-get install libcurl4-openssl-dev
```

Sind diese Pakete installiert, kann nun mit der Installation fortgefahren werden.

5.2 BitBake

Der Nächste Schritt ist nun *BitBake* zu installieren. Hierzu kann einfach die neuste Version von der Homepage des Herstellers bezogen werden [4]. Diese wird dann einfach in den gewünschten Ordner entpackt. Hier ist es sinnvoll, für *OpenEmbedded* und *BitBake* einen gemeinsamen Ordner zu erstellen, z.B /OEBASE.

5.3 OpenEmbedded

Nun kann OpenEmbedded installiert werden. Der einfachste Weg ist hierbei *OpenEmbedded* über *GIT* zu beziehen.

```
> git clone git://github.com/openembedded/openembedded.git
```

Falls nötig, kann nun noch ein Update bezogen werden. Es wird empfohlen täglich nach Updates für *OpenEmbedded* zu suchen. Dazu wechselt man im Terminal einfach in den Ordner, in welchen man OE geladen hat, und sucht via *GIT* nach Updates

```
> git pull --rebase
```

Sollte ein neues Update vorhanden sein wird des automatisch hinzugefügt. Nun muss *OpenEmbedded* noch zur PATH Variable hinzugefügt werden.

```
> export BBPATH=/stuff/build:/stuff/openembedded
> export PATH=/stuff/bitbake/bin:$PATH
```

Für die Pakete *SCANELF* sowie *OPKG-builder* müssen nun symbolische Links erstellt werden, da OpenEmbedded diese sonst nicht findet.

```
> sudo ln -s /path/to/OE_BASE/openembedded/bin/stage-manager-opkg-build
  /usr/bin/opkg-build
> sudo ln -s /usr/bin/scanelf
  /path/to/OE_BASE/build/tmp/sysroots/x86_64-linux/usr/bin/scanelf
```

Da die Datei *SCANELF* im *tmp* Ordner liegt, muss der Link bei jedem neuen Build neu angelegt werden.

[4] `http://developer.berlios.de/projects/bitbake/`

6 Das Erstellen des Kernels

6.1 Konfigurieren von OpenEmbedded und BitBake

Sind *BitBake* und *OpenEmbedded* auf dem System installiert, sollte dort ein Ordner vorhanden sein, welcher die Unterordner `BitBake`, `openembedded` und `build` enthält. Der `build` Ordner ist unser Hauptordner, in welchem die Distribution erstellt wird. Dazu muss nun zuerst einmal die vorgefertigte *OpenEmbedded* Konfigurationsdatei `/stuff/openembedded/conf/local.conf.sample` in den `build` Ordner kopiert werden. Diese Konfigurationsdatei bildet ein Skript, mit welchem *BitBake* und *OpenEmbedded* gesteuert werden. Diese Datei gilt es nun zu konfigurieren, um die gewünschte Distribution zu erhalten. Dazu öffnet man die Datei mit einem beliebigen Texteditor. Die Folgenden Variablen müssen in `local.conf.sample` geändert werden. Einige Variablen müssen einfach nur auskommentiert werden. Dies geschieht durch Entfernen des # Symbols vor der gewünschten Zeile. Es ist auch darauf zu achten, dass vor dem Variablenname kein Leerzeichen stehen darf.
`DL_DIR` gibt den Download Ordner für all die Dateien an, welche BitBake aus dem Internet laden muss.
`DL_DIR = "${HOME}/sources"`
`BBFILES` gibt den Ordner an, in welchem *BitBake* nach Recipes suchen soll.
`BBFILES = "/stuff/OEBASE/openembedded/recipes/*/*.bb"`
`help2man` wurde vorher schon installiert, und wird deshalb zur `ASSUME_PROVIDED` Variable hinzugefügt.
`ASSUME_PROVIDED += "help2man-native"`
Da wir eine Distribution für das *Lyrtech SFF-SDR* erstellen wollen, wird die Variable `MACHINE` entsprechen definiert.
`MACHINE = "davinci-sffsdr"`
Als Distribution wird die *Angström* Distribution ausgewählt.
`DISTRO = "angstrom-2010.x"`
Um den Kernel und das Dateisystem zu erstellen nutzen wir den *Sourcery CodeBench* Cross-Compiler. Dementsprechend müssen wir OE konfigurerien. `TOOLCHAIN_VENDOR = "-none"`
`TOOLCHAIN_TYPE = "external"`
`TOOLCHAIN_BRAND = "csl"`
`TOOLCHAIN_PATH = "/path/to/codesourcery"`
BitBake soll nun Debug-Messages auf der Konsole ausgeben.
`BBDEBUG = "yes"`
Ist die Host-Plattform mit einem Multicore-Prozessor ausgestattet, kann der Build durch angeben der Prozessorkerne parallelisiert und beschleunigt werden.
`BB_NUMBER_THREADS = "number of cores"`
Ausserdem soll *BitBake* uns den Log des Builds angeben.
`BBINCLUDELOGS = "yes"`
Nun müssen noch einige Zusatzbedingungen erstellt werden. Zum einen wollen wir eglibc2.11 verwenden, zum anderen wollen wir noch MTD-Utils implementieren
`PREFERRED_VERSION_eglibc = "2.11"`
`ANGSTROM_EXTRA_INSTALL="mtd-utils"`

Nun muss man noch die letzte Zeile löschen, danach kann die Distribution erstellt werden.

6.2 Patchen des Kernels

Da der Kernel später einmal das Dateisystem aus dem NAND-Flash laden soll, muss der NAND-Flash noch partitioniert werden. Dies geschieht beim Kompilieren des Kernels. Dafür existiert am CEL bereits ein Patch der auf der Homepage des CELs unter Downloads kostenlos heruntergeladen werden kann. Dieser Patch kann ganz einfach an die jeweiligen Anwendung (Partitionsanzahl, -Größe) angepasst werden[5].

6.3 Der Build

Nun kann die Distribution erstellt werden. Hierfür nutzen wir ein vorgefertigtes Recipe für das SFF-SDR. Dieses wurde bein Installieren von OpenEmbedded mitgeliefert. Der Build-Befehl lautet in diesem Fall `bitbake`. Für weitere Information zu dem Befehl sein auf die BitBake Manual verwiesen [6]. Der Build wird über die Konsole gestartet.

```
bitbake -b //OE_BASE/openembedded/recipes/linux/linux-davinci_2.6.28.bb
```

BitBake legt nun einen Ordner namens `stuff/build/tmp` an, in welchem der Linux Kernel abgelegt wird.

7 Das Rootfilesystem

7.1 Einführung

Das Rootfilesystem beinhaltet die Grundstruktur des Dateisystems für die Distribution. Sie beinhaltet auch die Hierarchie des Dateisystems. Man kann sie somit z.B. mit der `C:\` Struktur von Windows vergleichen. Unter anderem bestimmt das ROOTFS auch das Erscheinungsbild der Distribution (Console-only Distro oder GUI-Desktop-Umgebung) und es beinhaltet schon einige Basis Programme und Befehle, die je nach gewünschter Distribution unterschiedlich mächtig sind. Da wir eine Distribution für das *Lyrtech SFF-SDR* erstellen wollen, beschränken wir uns hier auf reine Konsolen-Distributionen.

7.2 Die Angstrôm Distribution

Das *Angstrôm*-Projekt hat sich auf die Entwicklung von Linux-Distributionen für verschiedene Handheld-Geräte (PDAs, Mini-PCs) spezialisiert. Dafür existieren verschiedene Skripte, welche auch von OpenEmbedded genutzt werden können. In dieses Kompendium werden wir uns aber mit dem Online-Builder von Angstrôm beschäftigen.

[5] hier koennte ihr Link stehen
[6] `http://bitbake.berlios.de/manual/`

7.3 Das Erstellen der Distribution

Wir nutzen hierfür den Online-Builder Narcissicus [7]. Mit diesem kann man Online ein ROOTFS erstellen und dann herunterladen. Dies geschieht einfach durch Auswählen der gewünschten Optionen mittels Drop-Down Menüs.
Als erstes müssen wir die Plattform wählen, für welche wir unsere Distro erstellen wollen. In unserem Fall wählen wir: *dm6446-evm*
Mit der nächsten Option kann man dem ROOTFS einen Namen geben. Dieser ist natürlich vom Nutzer frei wählbar.
Für Option *Complexity* wählt man: *advanced*.
Für die nächste Option *Release-Version* kann zur Zeit nur die Version *2011.3* gewählt werden (Stand: 06.12.2011).
Als *Base-System* wählen wir: *small*
Die nächste Option die wir ändern ist das *SDK*. Hier wählen wir: *simple toolchain*.
Das *Hostsystem* muss natürlich auf die zur Verfügung stehenden Rechner angepasst werden.
Zum Schluss wählen wir als *Environment*: *Console only*.
Nun können dem ROOTFS noch zusätzliche Pakete hinzugefügt werden. Diese richten sich je nach Bedarf und Größe des zur Verfügung stehenden Speicher. Wir empfehlen aber die *TI Gstreamer Plugins*, sowie die *Bootloaderfiles* einzubinden. Je nach Erfahrung mit Embedded-Programming und dem *DM-6446* kann man auch die Beispiele von *TI* hinzufügen.
Dann nur noch auf den Knopf *build me* drücken und sich in aller Ruhe einen Kaffee oder ähnliches besorgen. *Narcissicus* erstellt jetz Zwei Dateien: Die Toolchain und das ROOTFS, welche dann zum Download bereit stellen.

8 Flashen des Lyrtech SFF-SDR

8.1 Flashen des Nand-Speichers

Von Werk aus läuft auf Lyrtech SFF-SDR bereits eine Linux-Distribution von Green Hills Software. Diese muss nun zuerst entfernt werden, bevor wir eine neue, offene Distribution auf dem ARM implementieren können. Zu diesem Zweck gibt es von TI ein Programm, den DV- Flasher (DVF).Gestartet wird der DVF uber die Konsole und verbindet sich dann per RS232 -Schnittstelle mit dem DM6446SoC. Dabei wird der komplette Flash-Speicher des SFF-SDR gelöscht. Dies betrifft natürlich auch die Green Hills Distribution. Lösungen, die Distribution wieder her- zustellen wurden in dieser Arbeit dabei nicht untersucht. Dies sollte man Bedenken bevor man den NAND-Speicher löscht, da dadurch das SFF-SDR erstmal funktionsuntüchtig gemacht wird. Über einen zweiten Befehl kann der DVF nun den UBL sowie die kompilierte Binary von das U-Boot in den DDR2- RAM des SFF-SDRs laden.

```
$ sudo DVFlasher_1_14.exe -enand
$ sudo DVFlasher_1_14.exe -fnandbin u-boot.bin
```

[7] `http://narcissus.angstrom-distribution.org/`

Der DVFlasher ist frei verfügbarbar über die Open SDR Homepage [8].

8.2 Kernel im Nand

Da der Speicher des SFF-SDRs vollständig gelöscht wurde, kann nun der gewünschte Kernel fest in den NAND geschrieben werden. Dies geschieht nun mit dem U-Boot. Wird das SFF-SDR nun angeschaltet, so erscheint die Konsolenoberfläche von U-Boot. Je nach Einstellung versuch das U-Boot einen Autoboot durchzuführen. Dies kann durch drücken einer beliebigen Taste abgebrochen werden. Der Kernel wird nun mittels eines TFTP-Server in den DRAM des Boards geladen, und kann dann mittels U-Boot in den Festspeicher geschrieben werden. Für weitere Informationen zum Einrichten eines TFTP Servers unter Ubuntu/Xubuntu sei auf das Ubuntuusers-Wiki verwiesen[9]. Bevor wir den Kernel in den Flash schreiben können, müssen wir erst seine Größe bestimmen, da der NAND Flash nur ganzzahlige Vielfache der sogenannten Pagelänge (2048 Bytes) schreiben kann. Deshalb im Schreibvorgang die Länge des beschriebenen Bereiches auf das nächst größere Vielfache der Pagelänge aufgerundet werden. Im Folgenden werden nun die benötigten Befehle im U-Boot beschrieben.

```
> tftpboot 0x80700000 uImage
/* läd den Kernel an die angegebene Adresse im DRAM */
> nand write 0x80700000 0x2e00000 0x1fe0000
/*  schreibt die Datei aus dem DRAM an die gewünschte Stelle im NAND */
```

8.3 Einbinden des ROOTFS

Nun muss noch das Dateisystem eingebunden werden. Dazu wird zuerst ein NFS-Server auf dem Hostrechner erstellt. Dazu kann das entsprechende Paket **nfs-kernel-server** aus den Paketquellen installiert werden. Danach erstell man sich einen Ordner, in welchem dann das Dateisystem liegen soll. Als Beispiel verwenden wir hier den Ordner */exports/sffsdr*. Das mit Narcissicus erstellte Dateisystem wird nun in diesen Ordner entpackt. Danach müssen nur die Zugriffsrechte auf diesen Server verwaltet werden. Dies geschieht in der Datei `/etc/exports`. In dieser Datei fügt man nun folgende Zeile hinzu.

```
/exports/sffsdr *(rw,sync,no_root_squash,no_subtree_check)
```

Der NFS-Server ist nun fertig eingerichtet und der Kernel kann das Dateisystem über die Ethernetschnittstelle booten. Dazu muss dem Kernel aber erst einmal mittgeteilt werden, wo das Dateisystem liegt. Das geschieht über die Bootparameter, welche in U-Boot eingestellt werden können. Die Folgenden Schritte finden nun wieder auf dem SFF-SDR in U-Boot statt.

```
> setenv bootcmg 'nboot 0x80700000 0 02e00000;bootm'
> setenv bootargs 'console=ttyS0, 115200n8 root=/dev/nfs rw noinitrd
```

[8] http://www.opensdr.com/node/6
[9] http://wiki.ubuntuusers.de/advanced_TFTP?highlight=tftp

```
  ip=10.10.10.8 nfsroot=10.10.10.1:/exports/sffsdr eth=00:d0:cc:05:01:b7'
> saveenv
```

Der erste Parameter läd den Kernel vor dem Bootvorgang wieder in den DDR2-RAM denn nur hier können Anwendungen ausgeführt werden. Der zweite Parameter übermittelt dem Kernel Informationen, wo das Dateisystem zu finden ist, in diesem Fall auf dem NFS-Server. Über den Befehl boot sollte das Betriebsystem nun starten. Es erscheint der Anmeldebildschirm der Angström-Distribution. Man kann sich nun einfach mit dem Username *root* einloggen.
Der letzte Schritt ist es auch das Dateisystem in den Flash zu schreiben. Dazu startet man das Board ersteinmal per NFS, welches das gewünschte Dateisystem für den Flash bereits beinhaltet. Je nach Kernel-Einstellung sollte der NAND Flash nun partitioniert sein. Der Inhalt der für das Dateisystem vorgesehenen Partition muss nun gelöscht werden.

```
$ flash_eraseall -j /dev/mtdx
```

danach wird ein Ordner erstellt, welcher das spätere Dateisystem beinhalten soll. Dieser wird dann direkt als Blockdevice eingebunden.

```
$ mkdir /mnt/flash|
$ mount -t jffs2 /dev/mtdblockx /mnt/flash|
```

Nun kann das Dateisystem in diesen Ordner entpackt, und der Ordner wieder entbunden werden.

```
$ cd /mnt/flash
$ tar xzf <image dir>/rootfs.tar.gz
$ cd /
$ umount /mnt/flash
```

Das Board muss nun wieder neu gestartet werden. In U-Boot müssen die Bootparameter auf NAND-Boot umgestellt werden.

```
> setenv bootargs
  'console=ttys0, 115200n8 root=/dev/mtdblock1 rootfstype=jffs2 rw'
```

Wenn das Board das nächste mal gestartet wird, läd der Kernel das Dateisystem nun aus dem NAND-Flash.

References

[1] *Codesourcery-Getting Started.*