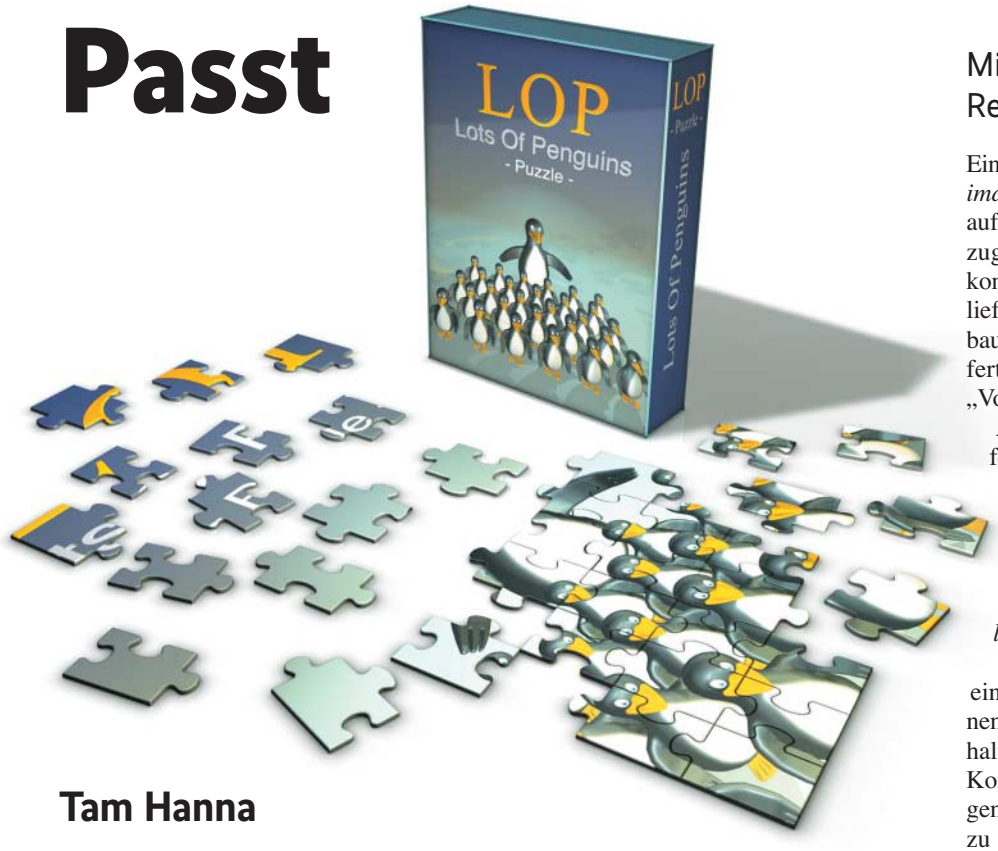
Yocto-Tutorial, Teil 2: Eigene Linux-Distribution für Raspberry Pi

Passt



Tam Hanna

Der zweite Teil des Tutorials zeigt, wie man mit dem Distributionsbaukasten Yocto eine Linux-Distribution für den Raspberry Pi der ersten Generation erzeugt. Entwickler müssen beim Zusammenbauen einiges anpassen. Dadurch erhalten sie ein Betriebssystem, das das GUI Sato und die Entwicklungsumgebung Qt Creator unterstützt.

Mit dem Distributionsbaukasten Yocto können Embedded-Entwickler eine eigene Linux-Distribution erstellen. Der erste Artikel [1] dieses dreiteiligen Yocto-Tutorials hatte gezeigt, wie man ein im Hardware-Emulator *qemu* (Quick EMUlator) lauffähiges Image erzeugt. Dank der in Yocto mitgelieferten Oberfläche Sato war es sogar zur Ausgabe eines grafischen Interface fähig. Als nächster Schritt bietet sich das Portieren auf ein mobiles Gerät an, denn Sato macht nur auf einem Raspberry Pi wirklich Spaß.

Ein weiterer Blick in den inneren Aufbau des Build-Systems erspart dabei lästige Handarbeit, denn Rezepte, Ebenen und Co. warten nur darauf, manuelle Operationen zu ersetzen.

Im ersten Teil der Serie konnten Entwickler das Build-System BitBake mit folgendem Befehl zum Generieren des Image animieren:

```
~/yoctoplace/poky$ source oe-init-build-env
~/yoctoplace/poky/build$ 7
                         bitbake core-image-sato
```

Der Source-Befehl parametriert die Kommandozeile mit diversen Umgebungsvariablen, die für das Kompilieren notwendig sind. Danach folgt ein Aufruf von BitBake, der das Rezept *core-image-sato* bearbeiten soll.

Bei Redaktionsschluss hatte die Datei *poky/meta/receipes-sato/imgaes/core-image-sato.bb* den in Listing 1 gezeigten Inhalt. Dabei beschreibt *DESCRIPTION* das durch das Ausführen des Rezepts entstehende Programm: Dieser Text ist hilfreich bei der Dokumentation und erscheint außerdem in diversen grafischen Yocto-Frontends.

Mit BitBake aus Rezepten Images generieren

Ein Image lässt sich mit dem Rezept *core-image-sato.bb* erzeugen. Hierfür darf es auf die Eigenschaft *IMAGE_FEATURES* zugreifen, die das direkte Einbeziehen komplexerer Subsysteme erlaubt. Zurzeit liefert das Projektteam den Distributionsbaukasten Yocto mit einer Gruppe vorgefertigter Funktionen aus (siehe Tabelle „Vorgefertigte Features“).

Anschließend sollte man die Lizenz festlegen, die für die in diesem Rezept generierten Inhalte gelten soll. Da das Yocto-Projekt von Haus aus mit MIT-lizenziertem Code umgehen kann, findet sich die passende Datei unter *poky/meta/files/common-licenses*.

Normalerweise muss jedes Rezept einen Prüfsummenblock mit Informationen über den Inhalt der Lizenzdatei enthalten. BitBake prüft diese während der Konfiguration, um Nutzer auf Änderungen von Unterkomponenten aufmerksam zu machen. Die im nächsten Schritt folgende *inherit*-Anweisung erklärt, warum das *core-image-sato*-Rezept ohne diesen Block auskommt. In der für die Inklusion angewiesenen Datei *core-image.bbclass* findet sich ganz oben die in Listing 2 aufgeführte Passage.

Board Support Package einbinden

Embedded-erfahrene Entwickler verziehen beim Anblick des Raspberry Pi oft das Gesicht: Der enorme Erfolg des Einplatinencomputers ist ein klassisches Beispiel für Hype, Mundpropaganda und gutes Marketing.

Leider weiß das vorliegende Yocto-Image nichts über den Raspberry Pi. Ändern lässt sich dies durch das Einbinden eines Board Support Package (BSP) – eines Layers, der das Image für die spezifische Hardwareplattform anpasst. Dank der weiten Verbreitung des Einplatinenrechners hostet das Yocto-Projekt ein fertiges BSP (siehe „Onlinequellen“, [a]). Auf der OpenEmbedded-Seite findet sich eine Liste der im BSP enthaltenen Pakete [b].

Da das gesamte Yocto-Projekt auf Git setzt, muss man das BSP – wie im vorhergehenden Teil das Plattform-Image –

auch über das freie Versionsverwaltungssystem beziehen. Nach einem Wechsel in den Ordner *poky* erledigt der Aufruf

```
git clone -b poky git://git.yoctoproject.org/ ⤷
                                meta-raspberrypi
```

das Gewünschte. Im ersten Teil der Serie hatte das Kommando *source oe-init-build-env* einen „Default-Build-Ordner" samt Arbeitsumgebung angelegt. Da der Raspberry Pi eine neue Konfiguration bedeutet, folgt nun der Aufruf mit dem als Bauplatz vorgesehenen Ordner *rpiHouse* als Parameter:

```
~/yoctoplace/poky$ ⤷
                source oe-init-build-env rpiHouse
```

Als Lohn der Mühen findet man sich – wie schon beim letzten Mal – im neuen Build-Ordner *~/yoctoplace/poky/rpiHouse* wieder. Die dort befindliche Konfigurationsdatei *local.conf* muss der Entwickler zunächst mit der neuen Maschine vertraut machen (siehe Listing 3).

Die Definition von *Machine* teilt den im BSP mitgelieferten Rezepten mit, dass die gerade aktive Kompilierung einen Raspberry Pi betrifft und somit für sie relevant ist. *BBMASK* ist ein regulärer Python-Ausdruck, der mit der spezifischen Konfiguration inkompatible Rezepte aus dem Übersetzungsprozess ausschließt.

Da das BSP ein Layer wie jeder andere ist, öffnet man im nächsten Schritt die Datei *rpiHouse/conf/bblayers.conf*, kopiert die letzte Zeile der *BBLAYERS*-Variable und lässt sie wie in Listing 4 zu sehen auf das soeben heruntergeladene Paket *meta-raspberrypi* zeigen. Nach getaner Arbeit ist es abermals Zeit, das Sato-Image zu übersetzen. Der dazu notwendige Befehl sieht so aus:

```
~/yoctoplace/poky/rpiHouse$ bitbake ⤷
                                core-image-sato
```

Bei Redaktionsschluss gibt es eine kleine Unstimmigkeit zwischen dem BSP und der Hauptversion von Yocto. Sie äußert sich in einem für diesen Artikel zwar didaktisch günstigen, aber lästigen Fehler, der auf ein leeres Rezept hinweist. Eine explizite Angabe des Git-Branch „daisy" vermeidet dies.

Beim Entwickeln eines BSP kommt es immer wieder vor, dass man ein Rezept leicht anpassen muss, um es besser auf die Bedürfnisse der Hardware einzustellen. Ungünstig ist es, die *.bb*-Datei zu kopieren, da Änderungen an der Quelldatei nicht automatisch ins BSP wandern.

Yocto löst dies durch *.bbappend*-Dateien und vermischt *gstreamer1.0-plugins-bad_1.4.0.bbappend* während des Übersetzens mit *gstreamer1.0-plugins-bad_1.4.0.bb*: Bei fehlenden oder umbenannten Originaldateien stoppt der Build aus Sicherheitsgründen.

Wer bei Unterschieden zwischen BSP und Basissystem trotzdem weiter kompilieren will, erweitert *local.conf* um folgende Passage:

```
BB_DANGLINGAPPENDS_WARNONLY = "1"
```

Die Entwickler von *meta-raspberrypi* setzen voraus, dass man normalerweise mit der Ebene *meta-oe* kompiliert. Diese lässt sich per Git herunterladen und an den gewohnten Platz kopieren:

```
~/yoctoplace/poky$ git clone -b daisy ⤷
    https://github.com/openembedded/meta-oe.git
~/yoctoplace/poky$ git clone ⤷
     git://github.com/dv1/meta-gstreamer1.0.git
```

Vorgefertigte Funktionen

Name	Funktion
dbg-pkgs	liefert Debuggersymbole für alle im Image enthaltenen Pakete aus
dev-pkgs	versorgt alle im Image enthaltenen Pakete mit Headers und zusätzlichen Bibliotheken, die für Entwickler nützlich sind
doc-pkgs	liefert die Dokumentation der enthaltenen Pakete mit aus
nfs-server	installiert einen NFS-Server
read-only-rootfs	weist den Yocto-Builder an, ein System mit einem nur lesbaren RootFS zu erstellen
splash	System zeigt beim Hochfahren einen Splash-Screen an. Dieser stammt normalerweise aus *psplash*, lässt sich aber durch die Variable *SPLASH* „umleiten".
ssh-server-dropbear	installiert den (vergleichsweise simplen) Dropbear-SSH-Server
ssh-server-openssh	installiert OpenSSH; überschreibt Dropbear, wenn beide gesetzt sind
staticdev-pkgs	liefert alle Pakete mit statischen Bibliotheken aus
tools-debug	installiert *strace* und *gdb*
tools-profile	liefert die Profiler aus
tools-sdk	sorgt dafür, dass das Image mit einem vollwertigen SDK ausgeliefert wird
tools-testapps	liefert diverse Apps aus, die beim Testen der Hardware des Systems hilfreich sind
x11	installiert den X-Server
x11-base	installiert X-Server und grundlegende Features
x11-sato	installiert Sato

Listing 1: *poky/meta/receipes-sato/images/core-image-sato.bb*

```
DESCRIPTION = "Image with Sato, a mobile environment and visual style for \
mobile devices. The image supports X11 with a Sato theme, Pimlico \
applications, and contains terminal, editor, and file manager."

IMAGE_FEATURES += "splash package-management x11-base x11-sato ssh-server-dropbear hwcodecs"

LICENSE = "MIT"

inherit core-image

IMAGE_INSTALL += "packagegroup-core-x11-sato-games"
```

Listing 2: Auszug aus *core-image.bbclass*

```
# Common code for generating core reference images
#
# Copyright (C) 2007-2011 Linux Foundation

LIC_FILES_CHKSUM = "file://${COREBASE}/LICENSE;md5=4d92cd373abda3937c2bc47fbc49d690 \
                    file://${COREBASE}/meta/COPYING.MIT;md5=3da9cfbcb788c80a0384361b4de20420"
```

X-TRACT

- Das Yocto-Team unterstützt Raspberry-Pi-Entwickler durch vorgefertigte Features und ein fertiges Board Support Package.
- Yocto beinhaltet ein Rezept, das das Deployment des Frameworks Qt auf dem Raspberry Pi erleichtert.
- Auch die Entwicklungsumgebung Qt Creator lässt sich für den Einplatinenrechner kompilieren.
- Inklusive GUI und Qt belegt die von Yocto für den Raspberry Pi erzeugte Distribution rund 400 MByte Speicherplatz.

Listing 3: *~/yoctoplace/poky/rpiHouse/local.conf*

```
#
# Machine Selection
#
# You need to select a specific machine to target
# the build with. There are a selection of
# emulated machines available which can boot and run
# in the QEMU emulator:
#

MACHINE ?= "raspberrypi"
BBMASK = "meta-raspberrypi/recipes-multimedia/libav|meta-raspberrypi/recipes-core/systemd"
```

Listing 4: Auszug aus *rpiHouse/conf/bblayers.conf*

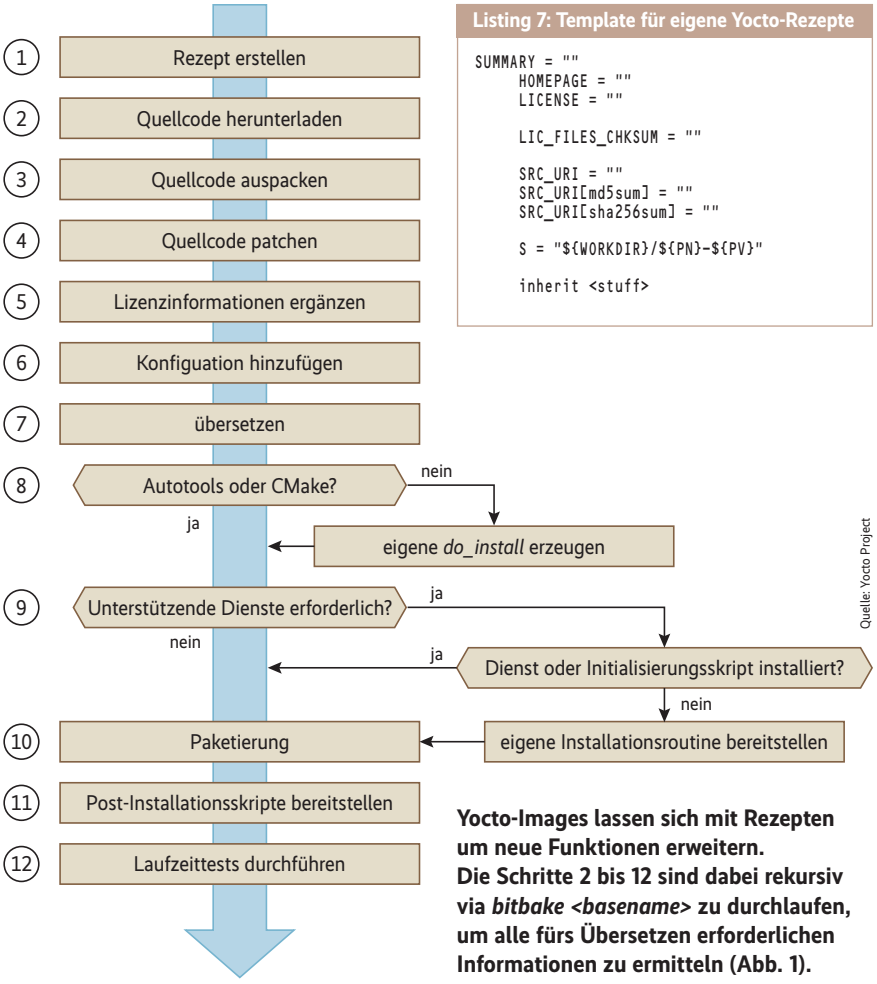
```
BBLAYERS ?= " \
  /home/tamhan/yoctoplace/poky/meta \
  /home/tamhan/yoctoplace/poky/meta-yocto \
  /home/tamhan/yoctoplace/poky/meta-yocto-bsp \
  /home/tamhan/yoctoplace/poky/meta-raspberrypi \
  "
```

Listing 5: *rpiHouse/conf/bblayers.conf*-Erweiterung

```
BBLAYERS ?= " \
[...]
  /home/tamhan/yoctoplace/poky/meta-raspberrypi \
  /home/tamhan/yoctoplace/poky/meta-oe/meta-oe \
  /home/tamhan/yoctoplace/poky/meta-gstreamer1.0 \
  "
```

Listing 6: *poky/meta/conf/bitbake.conf*-Ergänzung

```
SAVANNAH_GNU_MIRROR = "http://download.savannah.gnu.org/releases"
SAVANNAH_NONGNU_MIRROR = "http://download.savannah.nongnu.org/releases"
```



Yocto-Images lassen sich mit Rezepten um neue Funktionen erweitern. Die Schritte 2 bis 12 sind dabei rekursiv via *bitbake <basename>* zu durchlaufen, um alle fürs Übersetzen erforderlichen Informationen zu ermitteln (Abb. 1).

Listing 7: Template für eigene Yocto-Rezepte

```
SUMMARY = ""
      HOMEPAGE = ""
      LICENSE = ""

      LIC_FILES_CHKSUM = ""

      SRC_URI = ""
      SRC_URI[md5sum] = ""
      SRC_URI[sha256sum] = ""

      S = "${WORKDIR}/${PN}-${PV}"

      inherit <stuff>
```

Daraus folgt eine abermalige Modifikation von *bblayers* (siehe Listing 5). Zu guter Letzt müssen Yocto-Entwickler die Datei *poky/meta/conf/bitbake.conf* um zwei globale Variablen ergänzen (siehe Listing 6).

Man sollte sich nicht wundern, wenn das System beim Herunterladen von *0: bcm2835-bootfiles-20140508-r3* für einige Zeit stockt: Es handelt sich dabei um ein rund 3 GByte großes Archiv, das auf nicht sonderlich schnellen Servern liegt.

Wenn das Image fertig ist, muss man es auf eine SD-Karte verfrachten. Leider findet sich im Verzeichnis */tmp/deploy/images/raspberrypi* keine *.img-Datei*: Unter Yocto heißt das gesuchte File *core-image-sato-raspberrypi-<Zeitstempel>.rootfs.rpi-sdimg*.

Die rund 300 MByte kopiert man wie gewohnt per *dd* auf eine SD-Karte, um den Einplatinencomputer anschließend damit neu zu starten. Jetzt sollte nach einigen Sekunden das GUI Sato auf dem Bildschirm erscheinen.

Streng nach Vorschrift: Rezept generiert App

Da das Kommando aufgrund des Cross-Compilers bis zu dreimal länger braucht, ist die Gelegenheit günstig für etwas Theorie:

core-image-sato.bb demonstriert einige wichtige Features des Build-Systems, ist aber letztlich kein besonders gutes Beispiel für die Rolle eines Rezepts. Das liegt daran, dass es selbst keine Übersetzung anstößt, sondern nur weitere Rezepte einbindet.

Yocto arbeitet Rezepte immer anhand des in Abbildung 1 gezeigten Workflows ab. Für das Entwickeln eigener Rezepte ist die in Listing 7 gezeigte Struktur hilfreich. Man sollte sie als eine Art Template nutzen, da sie alle erforderlichen Felder abdeckt.

Nokia ersann den Raspberry Pi unter anderem als Demoplattform für das auf Qt basierende Dumbphone-Betriebssystem Meltemi. Wer einen frisch ausgepackten und mit Raspbian betriebenen RPi mit einer aktuellen Version des Frameworks Qt ausstatten möchte, muss nicht nur wegen der Hardwarekonfiguration einige Hürden überwinden.

Yocto kann an dieser Stelle helfen, wenn auch nur in beschränktem Rahmen. Das Projekt liefert Poky mit einem mehr oder weniger vorbereiteten Rezept aus, das das Deployment von Qt wesentlich erleichtert. Das Übersetzen erfolgt normalerweise vollautomatisch, lediglich ei-

nige Einstellungen für die besonderen Bedürfnisse der Zielhardware fehlen.

Das OpenEmbedded-Projekt stellt einen Layer bereit, der eine mehr oder weniger aktuelle Version von Qt 5 mitbringt. Laut Webseite hängt das Produkt unter anderem von der Ebene *meta-ruby* ab [c].

Als erster Schritt sind die notwendigen Layer mit dem Git-Werkzeug herunterzuladen. *meta-ruby* gibt es nur in Form eines Komplettpakets, das mit dem folgenden Kommando auf die Maschine des Entwicklers gelangt:

```
~/yoctoplace/poky$ git clone -b daisy ↵
  git://git.openembedded.org/meta-openembedded/
Cloning into 'meta-openembedded'...
```

Die für das Erzeugen von Qt 5 nötigen Rezepte lassen sich gezielt herunterladen:

```
~/yoctoplace/poky$ git clone -b daisy ↵
        git://github.com/meta-qt5/meta-qt5.git
Cloning into 'meta-qt5'...
```

Sind die Ressourcen auf den Computer gewandert, baut Yocto sie abermals in *bblayers.conf* ein. Die neue Version von *BBLAYERS* findet sich in Listing 8.

Anschließend ist die Datei *local.conf* wie in Listing 9 gezeigt um die Aufnahme der notwendigen Pakete zu erweitern. *IMAGE_INSTALL_append* fügt den jeweiligen als Parameter übergebenen String direkt in die Liste der zu verarbeitenden Rezepte ein. Wenn der hier abgedruckte Abstand vor den Kommandos fehlt, kommt es zu seltsamen Fehlermeldungen und Verweisen auf nicht vorhandene Pakete oder Abhängigkeiten. Die neu eingestellte Variable bewirkt, dass Yocto die soeben in *bblayers.conf* ergänzten Ebenen auch wirklich ins Projekt einbindet. Danach fehlt nur noch ein weiterer Aufruf von *bitbake*, um eine neue Image-Version zu generieren.

Der Einsatz von WebKit setzt die *icu*-Bibliothek voraus, die man sowohl als normale Ergänzung als auch als Parameter für die *qtbase*-Datei anmelden muss. Unterbleibt dies, entstehen beim Übersetzen alle möglichen Fehler, die das System bösartigerweise WebKit anrechnet. Ebenfalls zu beachten ist, dass die beiden Tasks relativ viel Zeit im Zustand *do fetch* verbringen, da Yocto die Quellcode-Dateien erst beim Übersetzen auf den Rechner herunterlädt.

In der Voreinstellung führt *bitbake* keine weiteren Tasks aus, sobald ein Build-Prozess einen Fehler zurückliefert. Dieses Verhalten ist oft wenig sinnvoll, da ein Gutteil der Pakete nicht voneinander abhängt. Beim unbeaufsichtigten Übersetzen geht so Rechenzeit verloren. Das Übergeben des Parameters *--continue* weist man *bitbake* dazu an, im Fall eines fehlgeschlagenen Pakets trotzdem weitere Tasks auszuführen.

Listing 8: Erweiterte *BBLAYERS*-Definition in *rpiHouse/conf/bblayers.conf*

```
BBLAYERS ?= " \
  /home/tamhan/yoctoplace/poky/meta \
  /home/tamhan/yoctoplace/poky/meta-yocto \
  /home/tamhan/yoctoplace/poky/meta-yocto-bsp \
  /home/tamhan/yoctoplace/poky/meta-raspberrypi \
  /home/tamhan/yoctoplace/poky/meta-oe/meta-oe \
  /home/tamhan/yoctoplace/poky/meta-gstreamer1.0 \
  /home/tamhan/yoctoplace/poky/meta-qt5 \
  /home/tamhan/yoctoplace/poky/meta-openembedded/meta-ruby \
  "
```

Listing 9: Qt5-Ergänzungen für *rpiHouse/conf/local.conf*

```
IMAGE_INSTALL_append = "  qtbase-fonts \
                qtbase-plugins qtbase-tools "
IMAGE_INSTALL_append = "      qtbase-examples"
IMAGE_INSTALL_append = "      qtdeclarative "
IMAGE_INSTALL_append = "      qtdeclarative-
                                plugins "
IMAGE_INSTALL_append = "      qtdeclarative-
                                  tools "
IMAGE_INSTALL_append = "      qtdeclarative-
                                qmlplugins "
IMAGE_INSTALL_append = "      qtmultimedia "
IMAGE_INSTALL_append = "      qtmultimedia-
                                plugins "
IMAGE_INSTALL_append = "      qtmultimedia-
                                qmlplugins "
IMAGE_INSTALL_append = "      qtsensors "
IMAGE_INSTALL_append =        qtsvg "
IMAGE_INSTALL_append = "      qtsvg-plugins "
IMAGE_INSTALL_append = "      qtimageformats-
                                plugins "
IMAGE_INSTALL_append = "      qtsystems "
IMAGE_INSTALL_append = "      qtsystems-tools"
IMAGE_INSTALL_append = "      qtsystems-
                                qmlplugins "
IMAGE_INSTALL_append = "      qtscript "
IMAGE_INSTALL_append = "      qt3d "
IMAGE_INSTALL_append = "      qt3d-qmlplugins
"
IMAGE_INSTALL_append = "      qt3d-tools "
IMAGE_INSTALL_append = "      qttools-plugins"
IMAGE_INSTALL_append = "      qtgraphical
                            effects-qmlplugins "
IMAGE_INSTALL_append = "      qtconnectivity-
                                qmlplugins "
IMAGE_INSTALL_append = "      qtbase"

IMAGE_INSTALL_append = " icu "

IMAGE_INSTALL_append = "      qtwebkit "
IMAGE_INSTALL_append = "      qtwebkit-
                             examples-examples "
IMAGE_INSTALL_append = "  qtwebkit-
                                qmlplugins "
PACKAGECONFIG_append_pn-qtbase = " icu"
```

Ab in den Qt-Creator

Ein auf Qt basierendes Image ist nur die halbe Miete. Wirklich komfortabel lässt sich das Abbild nur nutzen, wenn es sich aus dem Qt Creator [d] heraus ansprechen lässt. Aus leidvoller Erfahrung hier zu allererst ein Hinweis: Wer nicht die aktuelle Version 3.3 von Qt Creator einsetzt, bittet um Schmerzen. Daher sollte man sie herunterladen und im Versionsdialog prüfen, ob diese auch im Einsatz ist: Die meisten Linux-Distributionen bringen ältere Versionen der Entwicklungsumgebung mit, die besonders beim ohnehin fehleranfälligen Cross-Kompilieren zu Kinkerlitzchen neigen. Ubuntu 14.04 setzt beispielsweise Version 3.0.x ein.

Wie im ersten Teil der Serie erzeugt BitBake die Toolchain. Leider enthält das Rezept *packagegroup.qt5-toolchain-target.bb* die Zeile *„qttools-plugins \“* für ein leeres Paket, was den Erstellungsprozess blockiert. Daher muss man diese löschen oder in *local.conf* die Zeile

```
ALLOW_EMPTY_qttools-plugins = "1"
```

ergänzen, damit der auszuführende Befehl – *bitbake meta-toolchain-qt5* – auch durchläuft. Er generiert eine rund 700 MByte große *.sh*-Datei in *yoctoplace/poky/rpiHouse/tmp/deploy/sdk*. So kann man den Cross-Compiler installieren:

```
~/yoctoplace/poky/rpiHouse/tmp/deploy/sdk$ ↵
     ./poky-eglibc-x86_64-meta-toolchain-qt5- ↵
                    armv6-vfp-toolchain-1.6.2.sh
Enter target directory for SDK ↵
                  (default: /opt/poky/1.6.2):
You are about to install the SDK ↵
          to "/opt/poky/1.6.2". Proceed[Y/n]?Y
. . .
SDK has been successfully set up ...
```

Im nächsten Akt folgt der Start von Qt Creator. Hierzu sollte man die Version sicherheitshalber noch mal im About-Dialog prüfen: Wer nicht mindestens mit Version 3.2 arbeitet, dürfte im folgenden Schritt Probleme bekommen.

Ein Klick auf „Tools –> Options –> Build and Run –> Compilers“ öffnet den Dialog zur Compiler-Verwaltung. Mit „Add –> GCC“ kann man eine neue Toolchain anlegen – der Pfad zum Compiler *(path)* lautet normalerweise */opt/poky/1.6.2/sysroots/x86_64-pokysdk-linux/usr/bin/arm-poky-linux-gnueabi/arm-poky-linux-gnueabi-gcc*.

Nach einem Wechsel in das Qt-Versions-Tab lässt sich eine neue Qt-Version

Yocto in drei Teilen

Teil 1: Wie man am PC ein im Emulator lauffähiges Image erzeugt

Teil 2: Ein Image für den Raspberry Pi entsteht

Teil 3: Der Weg auf den Arduino Galileo

Glossar

ADT: Eine von Yocto erzeugte Cross-Compiler-Toolchain, die auf die Bedürfnisse des vorliegenden Image zugeschnitten ist. Wird im Android-Umfeld als Abkürzung für ein nicht verwandtes Eclipse-Plug-in verwendet.

BitBake: Build-System, das Yocto aus dem Projekt OpenEmbedded übernommen hat.

BSP: Board Support Package, ein Layer, der hardwarespezifische Module enthält.

Kit: Eine in QT Creator beim Erstellen einer Applikation eingesetzte Kombination aus Compiler und Qt-Version.

Layer: Design-Pattern, mit dem OpenEmbedded und Yocto komplexe Systeme unterteilen. Beschreibt eine Ansammlung von Rezepten, die gemeinsam eine Funktion realisieren.

Paket: Resultat von Rezepten im Yocto-Projekt.

***qemu* (Quick EMUlator):** Mit dem Emulator *qemu* kann man ein mit Yoco erstelltes Image auf dem PC laufen lassen.

Qt Creator: Integrierte Entwicklungsumgebung (IDE) für GNU/Linux, Mac OS X und Microsoft Windows.

Rezept: Fortgeschrittenes Makefile für Softwareprodukte im Yocto-Projekt. Ein Rezept beschreibt die Herkunft des Codes, sein Übersetzen und die Distribution des Resultats.

Sato: Im Yocto-Projekt enthaltenes GUI-System für Embedded-Systeme.

anlegen: Hierfür wählt man die unter */opt/poky/1.6.2/sysroots/x86_64-pokysdk-linux/usr/bin/qt5/* befindliche Version von *qmake* aus – die Entwicklungsumgebung erkennt automatisch, dass es sich um Qt für Embedded Linux handelt.

Zu guter Letzt muss der Nutzer ein sogenanntes Kit anlegen, eine Kombination aus Compiler und Qt-Version, die beim Erstellen einer Applikation zum Einsatz kommt. Qt Creator braucht für den Cross-Compiler eine lokale Kopie des Root-Verzeichnisses des Zielsystems – das SDK hat sie freundlicherweise unter */opt/poky/1.6.2/sysroots/armv6-vfp-poky-linux-gnueabi* bereitgestellt.

An dieser Stelle kann man die Toolchain testen, indem man ein beliebiges Programm zur Kompilation freigibt. Wirft Qt Creator eine Fehlermeldung nach dem Schema *make: c: Command not found* aus, findet das von Qt erzeugte Makefile die notwendigen Dateien nicht. Ein Blick auf den Qt-Kompilationsprozess (Abbildung 2) hilft, das Problem zu verstehen.

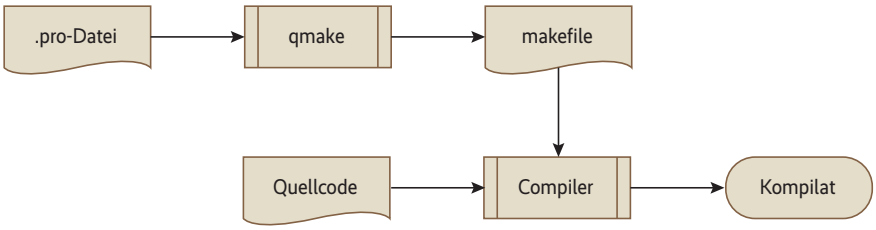
Wer das Makefile im *build*-Verzeichnis des Projekts ansieht, stellt fest, dass die Datei auf einige mit *OE_* beginnende Umgebungsvariablen verweist. Diese finden sich in Rohform in der unter */yocto-place/poky/rpiHouse/tmp/work/armv6-vfp-poky-linux-gnueabi/qtbase/5.3.2-r0/build* liegenden Datei *config.summary*; der relevante Block beginnt mit *qmake vars*.

Nun gilt es, die im Installationsverzeichnis von *qtcreator* befindliche Datei *qtcreator.sh* um die Deklaration der Variablen zu ergänzen, die man durch das Voranstellen von *export* einleitet. Die auf der Maschine des Autors funktionierende Beispieldatei erspart das Abtippen und Formatieren – sie kann zumindest als Vorlage dienen und steht wie *bblayers.conf* und *local.conf* über den *iX*-Listing-Service zum Download bereit [e].

Außerdem sollten Entwickler darauf achten, den in vielen Kommandos enthaltenen *Sysroot*-Befehl an den im Kit festgelegten anzugleichen. Unterbleibt dies, moniert Qt Creator während des Übersetzens das Fehlen diverser Bibliotheken.

Danach lässt sich die IDE durch einen Aufruf des Skripts neu starten. Wenn die Kompilation an dieser Stelle immer noch fehlschlägt, so sind zusätzliche Umgebungsvariablen nachzuladen. Diese finden sich unter */opt/poky/<version>* – die Dateien hören auf den Namen *./environment-setup-<arch>*.

Fertige Kompilate lassen sich von Hand auf die SD-Karte verschieben und danach auf der Konsole ausführen. Dank des in *core-image-sato* integrierten Dropbear-SSH-Servers kann man Qt Creator zum Remote Deployment animieren. Das Yocto-System verhält sich dabei wie jedes andere Embedded Linux Device.



Aus einer Projektdatei (.pro*) generiert *qmake* ein Makefile, das der Compiler anschließend abarbeitet (Abb. 2).**

Fazit

Am Raspberry Pi macht Yocto eine gute Figur: Das in diesem Artikel erzeugte Embedded-Betriebssystem kommt mit wesentlich weniger Speicher aus, nicht zuletzt, weil es auf diverse für einen dedizierten Clusternode nicht benötigte Funktionen verzichtet. Das mit Qt ausgelieferte Image kommt – samt GUI-System – mit rund 400 MByte Platz aus.

Im dritten Teil wird es um kniffligere Probleme gehen. Vor allem das Portieren auf den Arduino Galileo hält aufgrund der noch eingeschränkteren Hardware einige zusätzliche Stolpersteine bereit. (avr)

Onlinequellen

[a]	Board Support Package für Raspberry Pi	git.yoctoproject.org/cgit/cgit.cgi/meta-raspberrypi/
[b]	Liste der im BSP enthaltenen Pakete	layers.openembedded.org/layerindex/branch/master/layer/meta-raspberrypi/
[c]	meta-ruby auf OpenEmbedded	layers.openembedded.org/layerindex/branch/master/layer/meta-ruby/
[d]	Qt Creator	qt-project.org/wiki/Category:Tools::QtCreator
[e]	*iX*-Listing-Service	ftp.heise.de/pub/ix/ix_listings/2015/03/

Tam Hanna

ist Gründer der Tamoggemon Holding k.s. und beschäftigt sich seit 2004 mit der Entwicklung und Anwendung mobiler Geräte.

Literatur

[1] Tam Hanna; Linux nach Maß; Yocto-Tutorial, Teil 1: Embedded-Distributionen selbst bauen *iX* 2/2015; S. 44 ff

Alle Links: www.ix.de/ix1503146