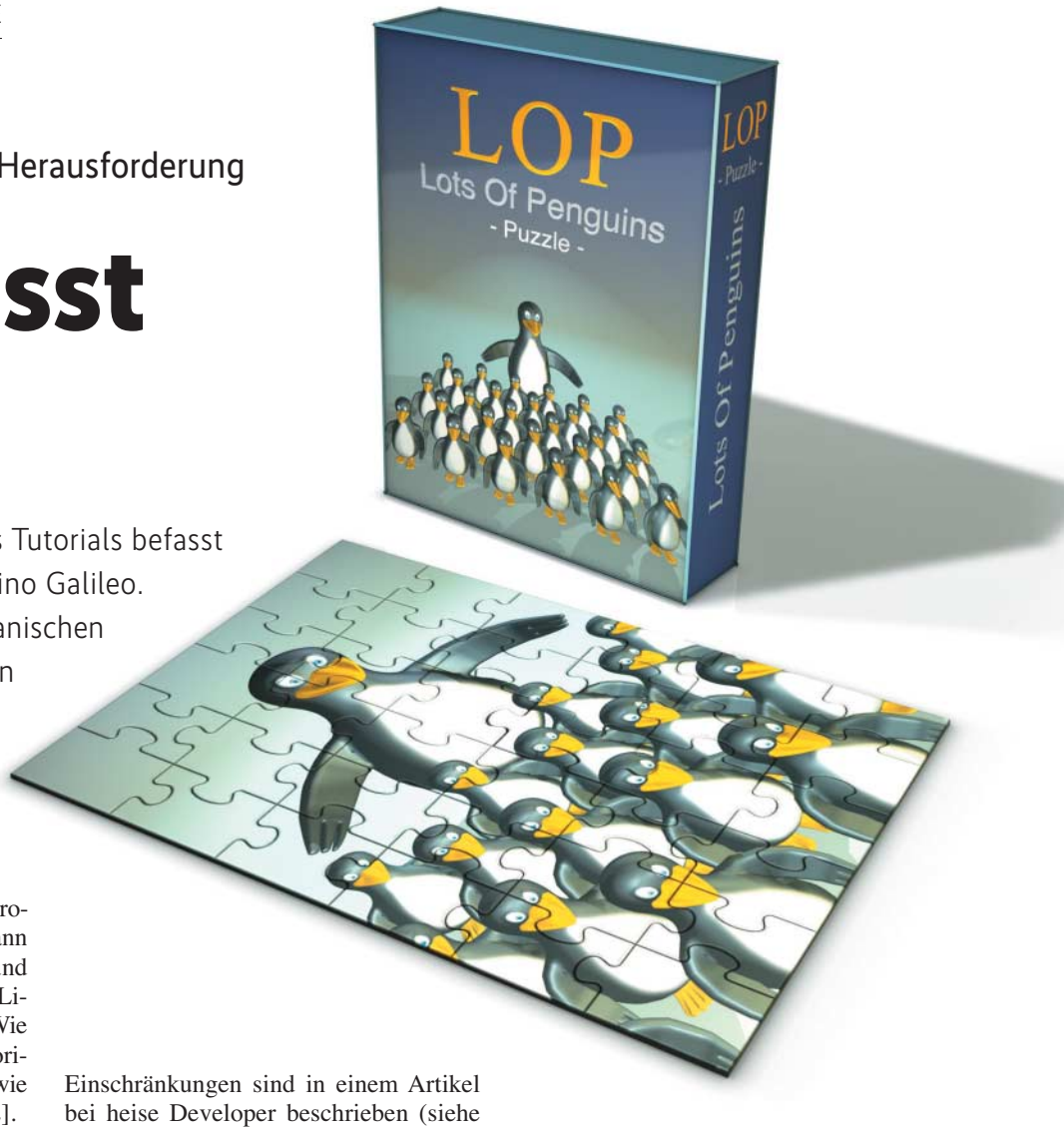
Yocto-Tutorial, Teil 3:
Intels Arduino Galileo als Herausforderung

# Angepasst

**Tam Hanna**




Im dritten und letzten Teil des Tutorials befasst sich der Autor mit Intels Arduino Galileo. Dafür sind aufgrund der spartanischen Hardware einige Anpassungen sowie Strategien zur Fehlersuche erforderlich.

Embedded-Systeme leiden unter chronischem Platzmangel. Daher kann eine eigene, auf die Hardware und den Zweck des Systems abgestimmte Linux-Distribution gute Dienste leisten. Wie in den ersten beiden Teilen dieses Tutorials zu lesen war, liefert ein Baukasten wie Yocto hier eine gute Hilfestellung [1, 2].

So hatte das Build-System BitBake beim Portieren eines Yocto-Image auf den Raspberry Pi die Änderung der Prozessorarchitektur mehr oder weniger komplett eigenständig erledigt. Im dritten und letzten Artikel dieser Serie geht es um schwierigere Aufgaben, denn nun ist Intels Arduino Galileo an der Reihe – eine in vielerlei Hinsicht seltsame Plattform. Ihr auf der x86-Architektur basierender Prozessor befindet sich trotz 256 MByte Arbeitsspeicher auf dem Stand eines Pentium I. Mit einem weitgehend Arduino-kompatiblen IO-Subsystem kann man beliebige Hardware anschließen. Seine Einschränkungen sind in einem Artikel bei heise Developer beschrieben (siehe „Onlinequellen“, [a]).

Trotz dieser Spezifikationen nach Art anderer Einplatinenrechner konkurriert der Galileo nicht mit Raspberry Pi und Co. Das liegt daran, dass er keine Videoausgabe mitbringt. Man kann zwar einen externen VGA-Controller anschließen, aber wegen des recht hohen Preises der Hardware und der vor allem dem I/O-Subsystem geschuldeten mauen Gesamtperformance ergibt ein kommerzielles Deployment der Platine dann nur noch sehr selten Sinn.

Abgesehen davon hat Intel den Galileo abgekündigt, und auch das Yocto-Projekt wartet die Plattform anscheinend nicht weiter. Dennoch kann der Versuch, ein Yocto-Image dafür zu bauen, als Trainingsgelände zum Entwickeln von Fehlersuchmethoden dienen.

## iX-TRACT

- Intel und das Yocto-Team unterstützen Entwickler für den Arduino Galileo durch vorgefertigte Features, Rezepte und Board Support Packages.
- Zwar kann die mitgelieferte grafische Benutzerschnittstelle Hob die Einstiegshürde senken, allerdings ersetzt sie kein Expertenwissen.
- Ein Einsatz auf dem Galileo-Board erfordert diverse Anpassungen und Korrekturen.
- Schwierigkeiten bereiten vor allem neue Versionen der beteiligten Systeme, da Intel und das Yocto-Team das Galileo-System nicht weiter pflegen.

## Ein Rezept für eine Qt-Anwendung

Vor dem Portieren lohnt sich ein kurzer Blick auf ein Rezept (siehe Glossar), das eine in der Distribution enthaltene Qt-Anwendung kompiliert. Wenn man die für das eigene Board benötigten Applikationen auf diese Art und Weise bereitstellt, wandern sie automatisch von Image zu Image – ein erheblicher Vorteil beim Deployment.

Listing 1 zeigt die Datei *quicky_0.4.bb*; sie definiert einen primitiven Texteditor, der Qt 4 benötigt, aber sonst keine besonderen Anforderungen an seine Umgebung stellt. Am Anfang stehen die diversen Deklarationen mit Informationen über Programmfunktion und Lizenz. Während des Übersetzens beschafft sich das Build-System BitBake den Quellcode und greift da-

bei direkt auf das *tar*-Archiv von *quicky* zu. Änderungen am Online-Quellcode erkennt es anhand diverser Prüfsummen.

Für Qt 4 bindet BitBake entweder *qt4x11* oder *qt4e* ein. Beide Distributionen enthalten einen Verweis auf die Datei *qmake_base.bbclass*, die das Werkzeug *qmake* und damit die eigentliche Kompilation anstößt. Für *quicky* fällt die Wahl per *inherit qt4x11* auf Ersteres. Den Abschluss bilden die im Rahmen der Installation auszuführenden Befehle (*do_install()*…).

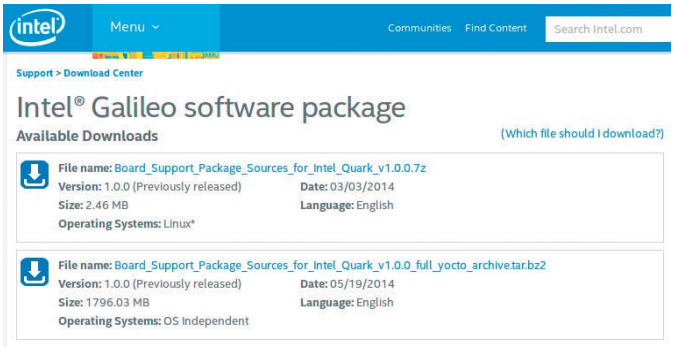
## Board Support Package passt Build-Prozess an

Wie schon beim Raspberry Pi brauchen Entwickler auch diesmal ein Board Support Package. Es handelt sich dabei um eine Yocto-Ebene, die den Build-Prozess an die vorliegende Hardware-Plattform anpasst. Intel bietet die für den Arduino Galileo notwendigen Ressourcen zum Download an [b]. Neben einem abgespeckten Board Support Package gibt es ein komplettes Archiv, das sämtliche für das Erstellen eines lauffähigen Image benötigten Informationen enthält (s. Abb. 1).

Wer das von Intel angebotene Board Support Package in die im Rahmen der Artikelserie erstellte Arbeitsumgebung implantiert, fängt sich an einigen Stellen Ärger ein: Die diversen Dateiversionen geraten sich in die Haare, was zu schwer behebbaren zirkulären Abhängigkeiten führt.

Aber ein anderer Weg führt zum Ziel: Nach dem Herunterladen der rund 2,2 MByte großen Basisdatei sollte man das darin enthaltene Archiv *meta-clanton** extrahieren und das daraus entstandene Verzeichnis an einer gut zugänglichen Stelle im Dateisystem sichern – im Artikel als */home/tamhan/yoctoplace/galileo*. Nach einem Wechsel in das Verzeichnis stößt der Aufruf *./setup.sh* den Download einer Galileo-kompatiblen Yocto-Version an.

Anschließend aktiviert der Befehl *source poky/oe-init-build-env yocto_build* das neue Build-Verzeichnis. Das unterscheidet sich von seinem Vorgänger aus den beiden ersten Teilen durch eine veränderte Struktur. Einige Layer befinden sich direkt in *galileo*, während andere im Unterverzeichnis *poky* liegen. Dies ist insofern kein Problem, als die Umgebungsvariable *BBLAYERS* mit absoluten Pfaden arbeitet – auf der Maschine des Autors sieht die initiale Konfigurationsdatei *yocto_build/conf/bblayers.conf* wie in Listing 2 aus. Im Verzeichnis *meta-clanton-bsp* befinden sich die eigentlichen Board-Support-Dateien, während *meta-clanton-distro* vor allem wegen der Image-Rezepte interessant ist. Aufgrund des fehlenden Grafikchips eignet sich das bisher verwendete *core-image-sato* nicht für den Galileo. Aber die Dateien *image-full* und *image-full-galileo* sind ein würdiger Ersatz.




**Intel bietet zwei Versionen des Board Support Package an (Abb. 1).**

## Grundlegendes fürs eigene Yocto-Image

Da die Leser das Image dieses Mal ändern müssen, sollten sie jetzt eine neue BitBake-Datei namens *meta-clanton-distro/recipes-core/images/heiseimage-full.bb* erstellen. Ihr Inhalt sieht anfangs so aus:

```
require image-full-galileo.bb
IMAGE_ROOTFS_SIZE = "1207200"
```

Rezepte und Images verhalten sich in Yocto wie Klassen in einer objektorientierten Programmiersprache. Das Heise-Image basiert im Moment auf *image-full-galileo.bb*, ändert aber die Variable *IMAGE_ROOTFS_SIZE* zwecks Vergrößerung des im Image verfügbaren Speicherplatzes.

Intel bietet Entwicklern an, ihre Programme entweder auf *eglibc* oder auf *uClibc* aufzubauen. Erstere ist eine mittlerweile in der Abkündigung befindliche abgespeckte Abart der *glibc*, während es sich bei *uClibc* um eine dedizierte C Library für Embedded-Systeme handelt.

Aufgrund der besseren Verträglichkeit – *eglibc* ist teilweise mit *glibc* binärkompatibel – geht es hier mit *eglibc* weiter. Das erfordert das Umstellen der Variablen *DISTRO* in *yocto_build/conf/local.conf* auf *clanton-full*:

```
[...]
MACHINE = "clanton"
DISTRO ?= "clanton-full"
EXTRA_IMAGE_FEATURES = "debug-tweaks"
[...]
```

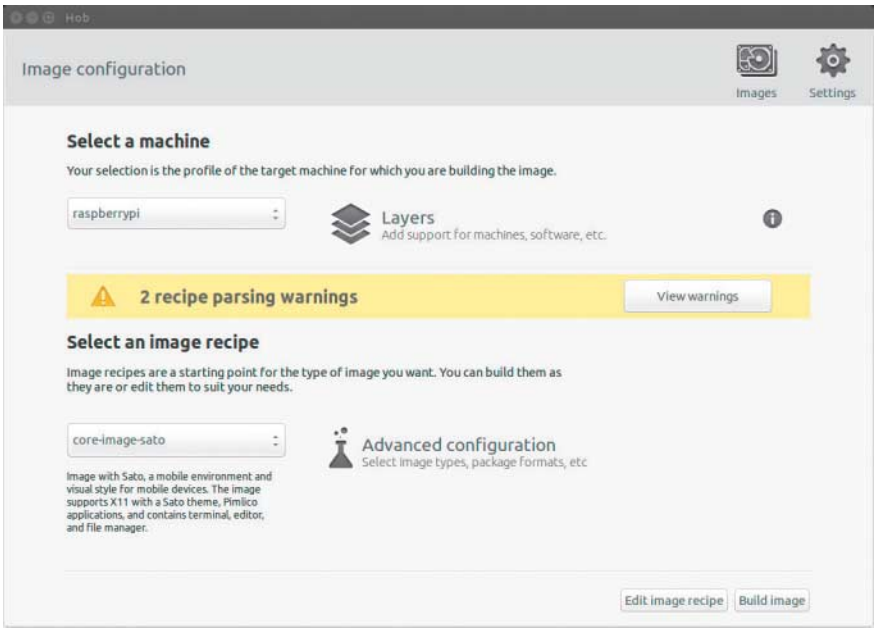
Wer keinen Achtkerner hat, sollte die von Intel fix verdrahteten Werte für die Thread-Anzahlen anpassen. Für Images, die Arduino-Sketches ausführen müssen, ist allerding *uClibc* die richtige Wahl: In diesem Fall bleibt *DISTRO* auf *clanton-tiny*.

**Listing 1: *poky/meta/receipes-qt/qt-apps/quicky_0.4.bb***

```
SUMMARY = "A simple note-taking application with Wiki-style syntax and behaviour"
HOMEPAGE = "http://qt-apps.org/content/show.php/Quicky?content=80325"
LICENSE = "GPLv2+"
LIC_FILES_CHKSUM = "file://version.h;endline=19;md5=878bdaff438dab86298301fd1a210e14"
SECTION = "x11/apps"

PR = "r2"

SRC_URI = "http://qt-apps.org/CONTENT/content-files/80325-quicky-0.4.tar.gz"

SRC_URI[md5sum] = "824d9e477ee9c4994f73a3cb215161d9"
SRC_URI[sha256sum] = "9c66376e0035d44547612bf629890769a6178c3e7eafbcf95f1c6207ac0f352a"

inherit qt4x11

do_install() {
  install -d ${D}${bindir}
  install -m 0755 ${S}/${BPN} ${D}${bindir}
}
```

**Listing 2: Initiale *bblayers.conf***

```
LCONF_VERSION = "6"
BBPATH = "${TOPDIR}"
BBFILES ?= ""
BBLAYERS ?= " \
  /home/tamhan/yoctoplace/galileo/poky/meta \
  /home/tamhan/yoctoplace/galileo/poky/meta-yocto \
  /home/tamhan/yoctoplace/galileo/poky/meta-yocto-bsp \
  /home/tamhan/yoctoplace/galileo/meta-intel \
  /home/tamhan/yoctoplace/galileo/meta-oe/meta-oe \
  /home/tamhan/yoctoplace/galileo/meta-clanton-distro \
  /home/tamhan/yoctoplace/galileo/meta-clanton-bsp \
  "
```




**Hob, das grafische Frontend des Yocto-Build-Systems, erleichtert das Erstellen vorkonfigurierter Images (Abb. 2).**

Den eigentlichen Build-Prozess übernimmt wie immer BitBake. Der in *yocto_build* aufgerufene Befehl lautet diesmal *bitbake heiseimage-full*. Im ersten Schritt erstellt BitBake einige fehlende Build-Komponenten, um danach mit dem Übersetzen zu beginnen.

Da einige Komponenten in *7zip*-Archiven vorliegen, sollte man eventuell die fehlende Komponente durch folgenden Befehl nachinstallieren:

```
sudo apt-get install build-essential p7zip-full
```

Auf aktuellen Distributionen kommt es mitunter zu einem Fehler, der auf eine ungültige *tar*-Datei hinweist:

```
tar: --same-order option cannot be used with -c
Try 'tar --help' or 'tar --usage' ⤷
                          for more information.
tar: This does not look like a tar archive
tar: Exiting with failure status ⤷
                         due to previous errors
```

Das GNU-Projekt hat die Kommandozeilenparameter von *tar* mit Version 1.27 verändert. Das als Basis für Intels Paket dienende Yocto 1.5 kommt mit der neuen Syntax nicht zurecht [c]. Diese Hürde lässt sich durch das Herunterladen einer älteren *tar*-Ausgabe beheben. Eine von Intel vorgegebene Kommandofolge löst das am einfachsten (s. Listing 3). Allerdings installiert sie systemweit eine potenziell unsichere *tar*-Version. Besser ist es daher, als Installationsziel das Verzeichnis für die Yocto-Skripte anzugeben, das in *PATH* vor den Systemverzeichnissen steht:

```
./configure --prefix=/<PFAD-zu-galileo>/ ⤷
                    poky/scripts --exec-prefix=
```

Der erste *make*-Aufruf scheitert mit einer Fehlermeldung. Um die Ursache zu beheben, muss man die eben generierte Datei *tar-1.26/gnu/stdio.h* öffnen und folgende Passage entfernen:

```
_GL_WARN_ON_USE (gets, "gets is ⤷
         a security hole - use fgets instead");
```

Danach geht das Kompilieren wie gewohnt weiter:

```
make
sudo make install
```

Zurück im Verzeichnis *galileo/yocto_build* ist es ratsam, das Übersetzen des Images durch den Parameter *--continue* zu ergänzen. Wer nicht auf Ubuntu 12.04 arbeitet, kann die jetzt auftretenden Fehler anhand der im Weiteren vorgestellten Informationen „en bloc“ abarbeiten. Unter Ubuntu 12.04 tritt – je nach Aktualität des Systems – nur ein Teil der Fehler auf.

Yocto-Images entstehen in einem Build-Verzeichnis, das normalerweise fünf Unterordner enthält. Der *conf*-Folder enthält die beiden in der Vergangenheit mehrfach besprochenen Einstellungsdateien, die das Verhalten des Image-Generators beeinflussen.

## Das passiert im Hintergrund

Die diversen im Rahmen von *do_fetch* heruntergeladenen Archive liegen im Ordner *downloads*. Er dient als eine Art Zwischenspeicher für Quellcode-Pakete, damit man sie bei einem eventuellen Neuübersetzen nicht neu beschaffen muss.

Im Verzeichnis *sstate-cache* lagern als *shared state* bezeichnete Versionsinformationen, eine weitere Optimierungsmaßnahme, die das erneute Abarbeiten von Jobs auf Arbeitsschrittebene erlaubt. Dazu ein kurzes Gedankenspiel: Ein Entwickler eines vergleichsweise aufwendigen Projekts ändert das Paketformat. Das macht die Resultate der Paketierung ungültig, nicht aber die der Übersetzung. Aus ressourcentechnischer Sicht wäre es sinnvoll, nur das Paketieren zu wiederholen. Yocto analysiert dazu die Ein- und Ausgabewerte der diversen Schritte eines Rezepts: Wenn sich in den Eingabewerten nichts geändert hat, gilt die Ausgabe als unverändert.

Normalerweise beansprucht das *tmp*-Verzeichnis den meisten Platz auf der Festplatte. BitBake nutzt es als Zwischenablage für all jene Informationen, die beim Kompilieren anfallen. Neben den als Resultat vorgesehenen Images befinden sich dort auch Zwischendateien. Unter Platzmangel leidende Entwickler können den Inhalt des Ordners entsorgen – wenn sie das Image nochmals benötigen, muss BitBake eben alles neu übersetzen.

## Fehler aufspüren und beheben

Für diesen Artikel ist *build/tmp/work/* von besonderer Bedeutung. Dieser Ordner enthält die von den diversen Paketen beim Kompilieren hinterlassenen Artefakte: Im vorigen Teil der Serie lag dort

**Listing 3: Ältere *tar*-Version einspielen**

```
wget http://ftp.gnu.org/gnu/tar/tar-1.26.tar.gz
tar -xf tar*.gz
cd tar*
./configure --prefix=/usr --exec-prefix=
make
```

**Listing 4: *OpenCV*-Fehlermeldungen**

```
ERROR: Function failed: Fetcher failure for URL:
   'http://downloads.sourceforge.net/opencvlibrary/opencv-unix/2.4.3/
          OpenCV-2.4.3.tar.bz2'. Unable to fetch URL from any source.
ERROR: Logfile of failure stored in:
                  /home/tamhan/yoctoplace/galileo/yocto_build/tmp/work/
              i586-poky-linux/opencv/2.4.3-r2/temp/log.do_fetch.28765
```

die Datei mit den diversen Einstellungen für Qt Creator. Diesmal dienen die in der BitBake-Konsole rot hervorgehobenen Fehlermeldungen zur Suche nach den betreffenden Logdateien. Als Erstes kommt *OpenCV* an die Reihe. Dazu finden sich in der Konsole die in Listing 4 aufgeführten Fehlermeldungen.

Das Öffnen der Logdatei informiert darüber, dass der Fetcher auch nach mehreren Versuchen keine Version des Archivs finden konnte. Dies liegt daran, dass die Entwickler OpenCV-2.4.3 von ihren Servern verbannt haben. Leider ist die Nachfolgeversion nicht mehr mit dem von Intel angebotenen Image kompatibel.

Erfreulicherweise steht die Datei zum Zeitpunkt der Drucklegung zum Download bereit [d]. Zur Lösung des Problems gilt es, alle Rezepte zu finden, die auf die alte URL hinweisen. Im Verzeichnis *meta-oe/meta-oe/receipes-support/opencv/* ist sowohl *opencv-_2.4.3.bb* als auch *opencv-samples_2.4.3.bb* in einem Editor zu öffnen. Die problematische Zeile sieht so aus:

```
SRC_URI="${SOURCEFORGE_MIRROR}/opencvlibrary/ ↵
       opencv-unix/${PV}/OpenCV-${PV}.tar.bz2 \
```

*PV* ist eine von mehreren Hundert Variablen, die den Aufwand beim Erstellen von Rezepten reduzieren. BitBake ersetzt sie beim Ausführen durch die vorliegende Version des Pakets. Eine Liste aller von BitBake selbst verwalteten Variablen gibt es auf den Webseiten des Yocto-Projekts [e]. Die korrigierte Version der Zeile lautet:

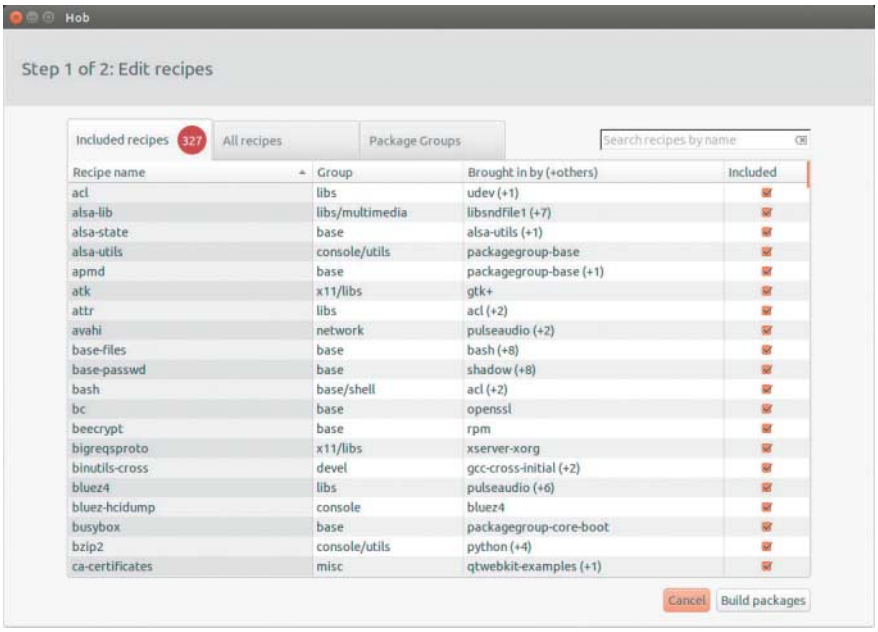
```
SRC_URI = "ftp://ftp.netbsd.org/pub/pkgsrc/ ↵
              distfiles/OpenCV-${PV}.tar.bz2 \
```

Anschließend müssen Entwickler den Build-Prozess abermals ausführen: Die Zahl der Fehler hat sich reduziert. Nun erfordert der von *git* geworfene Fehler im x264-Paket Aufmerksamkeit: Wartungsarbeiten im *git*-Tree des Projekts haben die Hashes der verschiedenen Commit-Stände verändert. Daraus folgt, dass die Prüfsumme „1cffe9f406cc54f4759fc9eeb8559 8fb8cae66c7“ nicht mehr auf ein nutzbares Image verweist. Im Rezept *meta-oe/meta-oe/receipes-multimedia/x264/x264_git.bb* sieht die betroffene Passage so aus:

```
SRCREV = ↵
     "1cffe9f406cc54f4759fc9eeb85598fb8cae66c7"
```

In der Praxis spräche nichts dagegen, diesen Hash einfach durch die korrekte Version zu ersetzen. Aus didaktischen Gründen ist es sinnvoller, dieses in Form eines *bbappend*-Files zu erledigen.

Dazu muss man im Verzeichnis *meta-oe/meta-oe/recipes-multimedia/x264* eine Datei namens *x264_git.bbappend* anlegen. Sie enthält folgende Zeile mit dem korrekten Hash:




**Pakete lassen sich in Hob fein abgestuft an- und abwählen (Abb. 3).**

```
SRCREV = ↵
     "bfed708c5358a2b4ef65923fb0683cefa9184e6f"
```

Die Einstellungen in den *bbappend*-Dateien überschreiben die Inhalte ihrer Mutterdatei – zumindest so lange, wie es sich nicht um ein durch *IMAGE_FEATURES+=* eingepflegtes Paket handelt. Die Variable *SRCREV* aus der *bbappend*-Datei überschreibt die im Paket befindliche Version, wodurch BitBake auch *x264* erfolgreich zusammenbaut.

Bisher war die Arbeit insofern einfach, als die bei der Fehlersuche ausgegebenen Informationen in den Cache des durchschnittlichen Terminals passten. Bei komplexen Kompilationsprozessen ist dies oft nicht der Fall. Das Problem in den *elfutils* sowie der Fehler in Wayland lassen sich nur anhand der Logdateien finden. Als Beispiel dient der Fehler, den die in Listing 5 gezeigte Passage im Logfile beschreibt.

Es handelt sich hierbei um eine von Red Hat eingeführte Änderung, die die Kompatibilität mit älteren Computersystemen verbessern sollte. Leider kommen moderne Compiler damit nicht zurecht, weshalb der Format-String anzupassen ist. In mehreren *addr2line.c*-Files finden sich Zeilen, die dem im Logfile monierten Schema entsprechen. Die korrigierte Version ersetzt *%a* durch *%m*:

```
%$:if (sscanf (string, "(%m[^)])%" PRIiMAX ↵
                    "%n", &name, &addr, &i) == 2
```

Da Intel eine für das Übersetzen von Yocto-Images optimierte virtuelle Maschine samt Ubuntu 12.04 anbietet, beendet der Artikel die Arbeit an dieser Stelle, denn das Beheben weiterer Fehler steigert den Lerneffekt nicht mehr. Intels nur mehr in Restbeständen verfügbarer Galileo I ist für die gebotene Leistung sehr teuer – wer auf den MiniPCIe-Slot verzichten kann, ist anderswo sowieso besser aufgehoben. Das scheinen auch zumindest einige der beteiligten Intel-Entwickler so zu sehen. So findet sich in SUSE Studio zwar noch das Image einer vorkonfigurierten Galileo Build Appliance [f]. Diese basiert allerdings auf dem von SUSE Studio nicht mehr unterstützten openSUSE 12.3 und steht daher nicht mehr zum direkten Download bereit.

## Yocto in drei Teilen

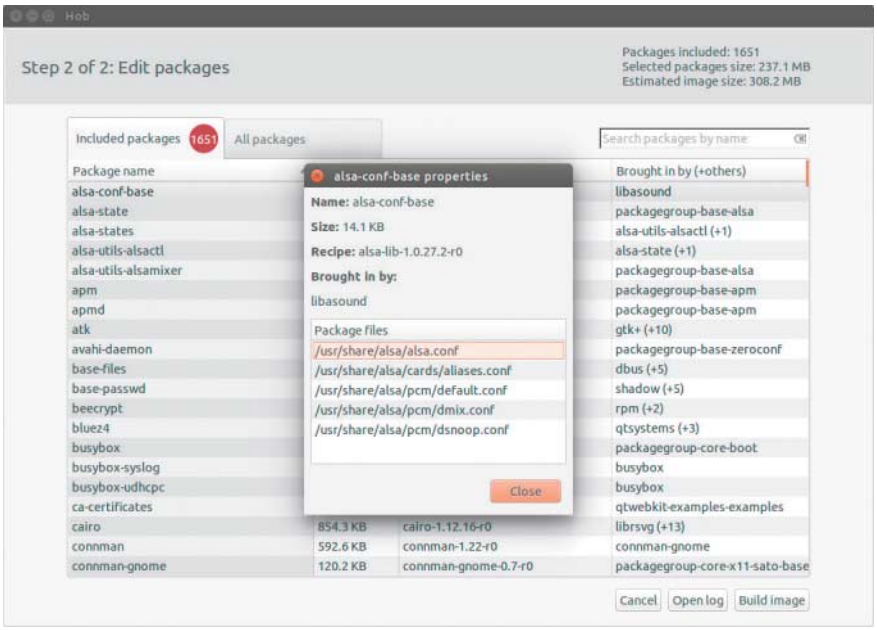
Teil 1: Wie man am PC ein im Emulator lauffähiges Image erzeugt

Teil 2: Ein Image für den Raspberry Pi entsteht

**Teil 3: Der Weg auf den Arduino Galileo**

## Image mit dem GUI Hob generieren

Bislang hat der Artikel gezeigt, wie man die Images von Hand konfiguriert und die zu verwendenden Ebenen und Rezepte mit einem Texteditor zusammenstellt. Diese Vorgehensweise ist insofern suboptimal, als kleine Syntaxfehler zu Problemen beim Kompilieren führen können.




**Das Paket *alsa-conf-base* trägt vergleichsweise wenige Dateien bei (Abb. 4).**

Hier verspricht das mitgelieferte grafische Frontend Hob Linderung.

Im Moment synchronisiert BitBake die in der Liste gezeigten Eigenschaften, die es aus *local.conf* und *bblayers.conf* entnimmt und in diese zurückschreibt:

```
MACHINE
PACKAGE_CLASSES
DISTRO
DL_DIR
SSTATE_DIR
SSTATE_MIRRORS
PARALLEL_MAKE
BB_NUMBER_THREADS
CONF_VERSION
BBLAYERS
LCONF_VERSION
```

Wer das grafische Frontend Hob nutzen will, braucht die Pakete Gtk+ 2.20.0 und PyGtk 2.21.0 – letzteres muss man unter Ubuntu gegebenenfalls per *apt-get install python-gtk2-dev* nachträglich einspielen. In der Regel funktionieren mit der Distribution gelieferte höhere Versionen der Komponenten. Zum Aktivieren von Hob ist es ratsam, im Terminal vorher *source* aufzurufen. Die ideale Kommandofolge zum Programmstart sieht so aus:

```
poky$ source oe-init-build-env
poky/build$ hob
```

Nach dem GUI-Start muss der Anwender im ersten Schritt das Zielsystem auswählen. Hob richtet sich hier nach den schon heruntergeladenen Ebenen. Bei der Geräteauswahl beginnt sofort ein mehrminütiger Parsing-Prozess, in dem das System alle vorhandenen Rezepte analysiert. Wer Hob in einem noch leeren Build-Verzeichnis ausführt, muss die zu verwendenden Ebenen über das Menü *Layers* einpflegen. Dabei erweitert sich sukzessive die im Feld *Machines* angebotene Auswahl.

Im nächsten Schritt ist das zu erstellende Image festzulegen (siehe Abbildung 2).

**Listing 5: Auszug einer Logdatei**

```
Argument of type 'float *', but argument 3 has type 'char **' [-Werror=format=]
|        if (sscanf (string, "(%a[^)])%" PRIiMAX "%n", &name, &addr, &i) == 2
|            ^
| /home/tamhan/yoctoplace/galileo/yocto_build/tmp/work/x86_64-linux/elfutils-native/
                    0.148-r11/elfutils-0.148/src/addr2line.c:453:7: error: format '%a' expects
                argument of type 'float *', but argument 3 has type 'char **' [-Werror=format=]
|        switch (sscanf (string, "%a[^-+]%n%" PRIiMAX "%n", &name, &i, &addr, &j))
|            ^
```

**Listing 6: Fehlerfall *qttools-plugins***

```
Loading cache...
Updating cache...               ######################################## [100%]

Computing transaction...error: Can't install packagegroup-qt5-toolchain-target-1.0-r0@all:
                                                        no package provides qttools-plugins

DEBUG: Python function do_populate_sdk finished
ERROR: Function failed: do_populate_sdk
```

Ein Klick auf den Button „Build Image“ startet einen BitBake-Lauf; die Anzahl der Threads sowie andere Compile-Parameter lassen sich über das Settings-Symbol bei Bedarf konfigurieren.

Hob zwingt nicht dazu, vorliegende Images eins zu eins zu kompilieren. Das „Edit Image Recipy“-Fenster öffnet den in Abbildung 3 gezeigten Dialog, der eine Liste aller am System befindlichen Rezepte und Pakete anbietet. Ein anschließender Klick auf „Build Packages“ startet die erste Stufe der Übersetzung. Hob lädt dabei alle auf dem System noch nicht vorhandenen Pakete herunter und kompiliert sie mit BitBake.

Nach diesem ersten Schritt kann der Anwender die auszuliefernden Pakete in einem weiteren Dialog auswählen. Das Anklicken eines Pakets öffnet das in Abbildung 4 gezeigte Fenster, das Informationen über die in das Root-Dateisystem eingebrachten Dateien anbietet. Es handelt sich hierbei um eine gekürzte Auswertung der Build History, die beim Deselektieren nicht benötigter Archive hilfreich ist.

Ein anschließender Klick auf den Button „Build Image“ sorgt dafür, dass das Image die gewünschten Formen bekommt. Wer Hob zum Erstellen eines Image vom Typ *Iso* oder *hddimg* anweist, den unterstützt ein kleiner Assistent beim Deployment. Yocto-Images für *qemu* lassen sich stattdessen direkt in den Emulator ausliefern.

## Tipps für die Fehlersuche

Kundendienst ist im Embedded-Bereich seit jeher eine knifflige Angelegenheit. Wenn Hardware und Software von zwei verschiedenen Anbietern stammen, schieben sich die beiden Unternehmen nur zu gern gegenseitig die Schuld zu.

Für einen Entwickler besteht der erste Schritt zur Lösung eines Problems darin, Teile der Fehlermeldung in der Suchmaschine einzugeben. In vielen Fällen hatten andere eine identische oder zumindest ähnliche Frage: Abschauen mag die nach Konfuzius niederste Form des Lernens sein, erweist sich aber als höchst wirksam.

Nützlich ist außerdem, das kritische Element im gesamten Yocto-Verzeichnis zu suchen. Beim in Listing 6 gezeigten Fall wäre dies *qttools-plugins* – je nach persönlichem Geschmack kann man über die Kommandozeile oder per GUI recherchieren. Rezepte höherer Ordnung weisen gern in der Praxis irrelevante Abhängigkeiten auf. Im Fall dieses Beispiels – aufmerksame Leser erkennen es

eventuell aus dem zweiten Teil – ließen sie sich ohne negative Folgen auskommentieren.

Wenn durch die Suche keine Lösung zu finden ist, sollte man sich im nächsten Schritt an die Yocto-Entwickler wenden mit einer Anfrage an eine Mailingliste [g]: Antworten gibt es normalerweise binnen weniger Stunden. Für eilige Probleme steht ein bei Freenode gehosteter IRC-Kanal zur Verfügung, der allerdings nicht mit der Reichweite der Mailinglisten mithalten kann.

Zu guter Letzt sei darauf hingewiesen, dass die Wartung von Embedded-Linux-Derivaten eine Aufgabe ist, die sich auch mit Yocto nicht nebenbei erledigen lässt. Das Pflegen eines aus Quellcode entstehenden Betriebssystem-Image erfordert enormes Wissen über die Plattform: Es ist keine Schande, von Zeit zu Zeit die Hilfe eines Beratungsunternehmens in Anspruch zu nehmen.

## Achtung, Versionswahnsinn!

Das Yocto-Team hat sich auf einen sechsmonatigen Release-Takt geeinigt. Für Entwickler bedeutet dies eine an Qt erinnernde Situation: Es gibt regelmäßig Updates mit diversen Änderungen. Während der Arbeit an dieser Artikelserie erschienen die Versionen 1.7 und 1.7.1. Ihr Spitzname Dizzy weist auf den Geisteszustand hin, den ein Entwickler beim Nachvollziehen der rapiden Änderungen einnimmt.

Aus technischer Sicht bringt Dizzy Updates an diversen Komponenten. Statt *eglibc* kommt eine Vollversion der *glibc* zum Einsatz und der GCC ist nun in der Version 4.9 dabei. Darüber hinaus bezieht sich das im Rahmen dieses Tutorials nicht besprochene Selbsttestsystem nicht mehr auf eigene erstellte Images: Auf kompatibler Hardware kann die Workstation das im Einsatz befindliche Image herbeiholen.

Ein als „Toaster“ bezeichnetes Webinterface soll die hier kurz vorgestellte Hob-Oberfläche zukünftig ersetzen. Leider wurde Toaster für Dizzy nicht rechtzeitig fertig.

Bei dem am 16. Januar 2015 ausgelieferten Yocto 1.7.1 handelt es sich – schon ob der vergleichsweise kleinen Inkrementierung der Seriennummer – um eine klassische Maintenance Release. Die verwendeten Kernels bekamen kleine Aktualisierungen, für diverse Bibliotheken hat das Yocto-Team Sicherheitsupdates und Bugfixes eingefügt.

Wenn das Embedded-System nicht mit dem Internet verbunden ist, müssen die Leser die Änderungen nicht unbedingt nachvollziehen. Yocto ist – anders als Qt – absolut nicht abwärtskompatibel. Das Vorwärtsportieren eines Board-Support-Pakets artet daher schnell in nur noch für Experten verwaltbare Arbeit aus. Wer kein Kernel-Hacker ist, geht an dieser Stelle über kurz oder lang k. o.

## 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.

**Hob:** Die grafische Benutzerschnittstelle des Yocto-Build-Systems BitBake.

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

**Paket:** Im Yocto-Projekt Bezeichnung für das Resultat von Rezepten.

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

**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.

## Fazit

An dieser Stelle endet die Reise durch die Welt von Yocto Linux. Leser haben sich mit den drei Ausgaben der Artikelserie hoffentlich reichlich Informationen über den Aufbau und die Nutzung des Build-Systems auf verschiedenen Plattformen erarbeitet.

Mit den hier gesammelten Erfahrungen kann man bei Bedarf im Internet weitere Informationen anfordern. Falsche Scham ist dabei nicht angebracht: Schließlich hat ein durchschnittlicher Programmierer in der Regel von Unix genauso viel Ahnung wie ein durchschnittlicher Systemadministrator von der Sprache C++.

Die Arbeit mit Yocto setzt enormes Wissen im Bereich Build-Systeme und Unix voraus. Grafische Frontends wie Hob können zum Lernerfolg nur wenig beitragen: Wenn zwei Rezepte nicht zusammenpassen, kann auch das GUI den Fehler nicht „erriechen“. (avr)

### 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.

[2] Tam Hanna; Passt; Yocto-Tutorial, Teil 2: eigene Linux-Distribution für Raspberry Pi *iX* 3/2015; S. 146 ff.

### Onlinequellen

| | | |
|---|---|---|
| [a] | Galileo unter der Lupe | heise.de/-2252539 |
| [b] | Board Support Package für Arduino Galileo | downloadcenter.intel.com/Detail_Desc.aspx?DwnldID=23171 |
| [c] | *tar* auf Version 1.27 updaten | lists.yoctoproject.org/pipermail/yocto/2013-October/016540.html |
| [d] | Download OpenCV-2.4.3 | ftp.netbsd.org/pub/pkgsrc/distfiles/OpenCV-2.4.3.tar.bz2 |
| [e] | Liste der von BitBake verwalteten Variablen | www.yoctoproject.org/docs/current/ref-manual/ref-manual.html#ref-variables-glos. |
| [f] | Galileo Build Appliance | susestudio.com/a/Ng4RLG/galileo-build-appliance-gnome |
| [g] | Yocto-Entwickler-Mailinglisten | www.yoctoproject.org/tools-resources/community/mailing-lists |

Alle Links: www.ix.de/ix1504148