Yocto-Tutorial, Teil 1:
Embedded-Distributionen selbst bauen

Linux nach Maß

Tam Hanna

Mit dem Distributionsbaukasten Yocto können Embedded-Entwickler eine eigene Linux-Distribution erstellen, die auf die Anforderungen ihres Systems zugeschnitten ist. Der erste Part des dreiteiligen Tutorials erklärt, wie man ein lauffähiges Image für den Emulator *qemu* erzeugt.

Embedded-Systeme und Einplatinencomputer leiden unter chronischem Speicherplatzmangel. Wer eine Applikation für ein solches Gerät entwickeln möchte, beginnt normalerweise mit einer Standarddistribution, um sie anschließend um unnötige Pakete zu erleichtern – ein Weg mit Tücken: Erstens führt das Abspecken nicht zum ideal kleinen System, zweitens lassen sich die Instruktionen nicht ohne Weiteres auf anderen Geräten wiederverwenden.

Hier setzt das Projekt Yocto an (siehe „Onlinequellen“, [a]), ein Build-Werkzeug, mit dem Entwickler eigene und für ihr Computersystem optimierte Distributionen samt den dazugehörenden Cross-Compilern erstellen können. An Yocto arbeiten über 40 Unternehmen mit [b]. Einen allgemeinen Überblick über Baukästen dieser Art gibt der Artikel „Montiert“ ab Seite 36.

Dieser erste Teil des Tutorials zeigt, wie man ein im Emulator lauffähiges Image erzeugt. Die Teile zwei und drei beschreiben, wie man es auf Raspberry Pi und Arduino Galileo einsetzen kann.

Ein paar Yocto-Grundlagen

Mit dem Build-Werkzeug Yocto können Embedded-Entwickler eigene Linux-Distributionen erstellen. Abbildung 1 zeigt alle Elemente, die beim Erstellen von Yocto-Images zum Einsatz kommen können.

Einen eigenen Betriebssystemkernel entwickelt Yocto nicht: Die realisierten Images basieren auf dem Projekt OpenEmbedded [c]. Es trägt zwei Komponenten bei: Teil eins bildet ein als BitBake bezeichnetes Build-System – der hellbraune Teil in der Mitte des Diagramms von Abbildung 1. Der zweite und nicht weniger wichtige Teil ist eine als *meta-oe* bezeichnete Sammlung von Rezepten, die grundlegende Systemkomponenten erstellt.

Das Build-System verarbeitet die in Form von Metadaten vorliegenden Informationen. BitBake lädt auf Wunsch automatisch Code-Dateien aus dem Internet und kommt mit allen verbreiteten und quelloffenen Archivformaten zurecht.

Yocto-Entwickler unterteilen Metadaten normalerweise in drei Gruppen. Die mit der Endung *.conf* versehenen Konfigurationsdateien beeinflussen das Verhalten des Build-Systems als Ganzes. Sie definieren Grundlegendes wie die zu verwendende Zielplattform oder notwendige Optimierungen.

Quelldateien für Softwareprodukte enden auf *.bb*. Diese sogenannten „Rezepte“ enthalten eine Gruppe von Befehlen, die das von ihnen beschriebene Modul erzeugen. Im Yocto-Konzept heißt das Resultat von Rezepten „Paket“. Das Team bittet in der Dokumentation darum, die Ausdrücke „Rezept“ und „Paket“ nicht synonym zu

iX-TRACT

- Mit dem Distributionsbaukasten Yocto können Embedded-Entwickler eigene Linux-Distributionen erstellen, die auf die Bedürfnisse der Geräte abgestimmt sind.
- Als Build-System nutzt Yocto das aus dem OpenEmbedded-Projekt stammende BitBake.
- Die Option, die Entwicklungsumgebung Eclipse in Yocto zu integrieren, bietet Entwicklern den Komfort einer professionellen IDE.

Yocto-Befehle

Befehl	Einsatzzweck
sudo apt-get install gawk wget git-core diffstat unzip texinfo gcc-multilib build-essential chrpath	essenzielle Pakete, ohne die das Kompilieren von Yocto-Images nicht möglich ist
sudo apt-get install libsdl1.2-dev xterm	Pakete, die auf Build-Systemen mit Grafikbildschirm hilfreich sind (lies: eigentlich immer)
sudo apt-get install make xsltproc docbook-utils fop dblatex xmlto	Werkzeuge zum Übersetzen der Yocto-Dokumentation
sudo apt-get install autoconf automake libtool libglib2.0-dev	Unterstützungsbibliotheken für ADT

Listing 1: Ausgabe von *source oe-init-build-env*

```
You had no conf/local.conf file. This configuration file has therefore been created for you
with some default values. You may wish to edit it to use a different MACHINE (target hardware)
or enable parallel build options to take advantage of multiple cores for example. See the file
for more information as common configuration options are commented.
. . .
### Shell environment set up for builds. ###

You can now run 'bitbake <target>'

Common targets are:
    core-image-minimal
    core-image-sato
    meta-toolchain
    adt-installer
    meta-ide-support

You can also run generated qemu images with a command like 'runqemu qemux86'

tamhan@TAMHAN14:~/yoctoplace/poky/build$
```

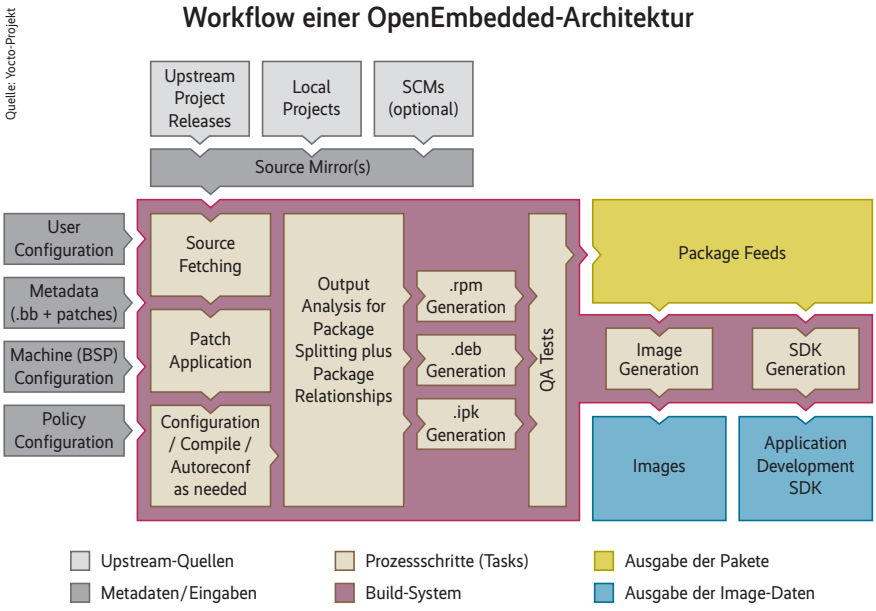
verwenden. Klassen dienen als Abstraktionsschicht für häufig verwendete Teile eines Rezepts. Darüber lässt sich doppelter Code vermeiden und das Ergebnis besser warten.

Embedded-Projekte können rasch auf einige Hundert Funktionen anwachsen. In diesem Fall führt kein Weg daran vorbei, sie in Layer zu zerlegen. Dabei handelt es sich um ein von OpenEmbedded übernommenes Design-Pattern zum Unterteilen komplexer Systeme: Ein Layer bezeichnet eine Gruppe von Metadaten, die in Zusammenhang zueinander stehen.

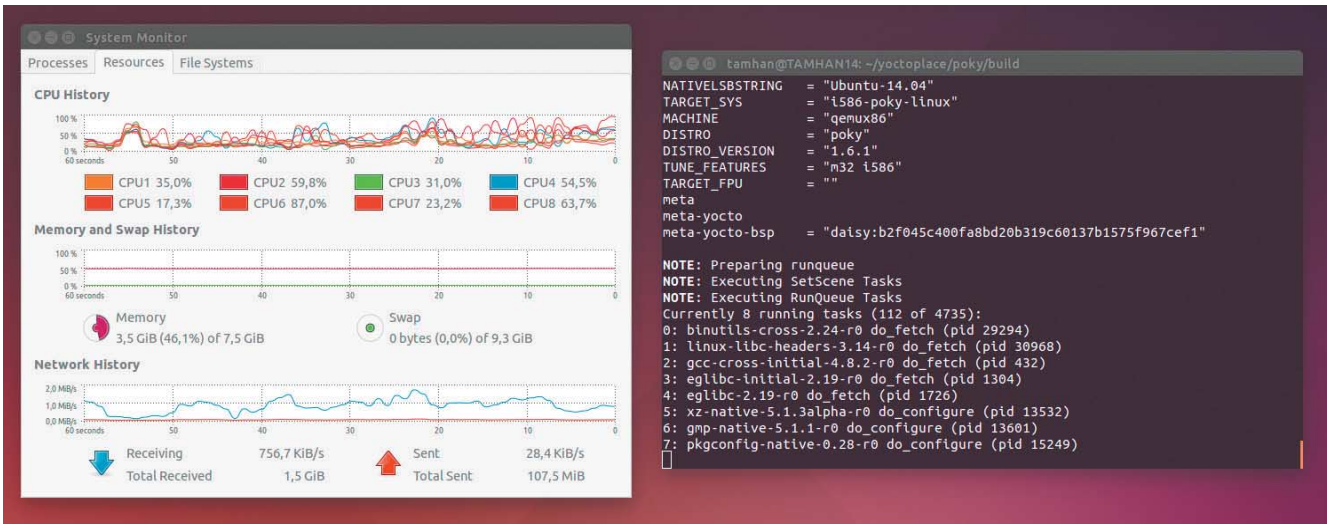
Vor dem Anlegen einer eigenen Ebene sollte man die Liste von mittlerweile einigen Hundert vorgefertigten Layers prüfen [d]. Sie lassen sich mit einem einzigen Befehl in das Projekt einbinden.

Das braucht Yocto

Diese Artikelserie basiert auf einer 64-Bit-Version von Ubuntu 14.04, einem achtkernigen AMD 8520 mit 8 GByte RAM und einer Solid State Disk (SSD). Yocto stellt hohe Anforderungen an die Hardware: Neben 50 GByte (!) Plattenplatz sollten Entwickler pro logischem Kern 1 GByte RAM einplanen. Das Build-System erfordert folgende Komponenten:
– Git 1.7.5 oder höher,
– tar 1.24 oder höher,



Yocto-Images durchlaufen bei ihrer Entstehung eine komplexe Werkzeugkette (Abb. 1).



Beim Kompilieren erreicht das System die volle Auslastung der CPU erst nach dem Download aller Quelldaten (Abb. 2).

Ein Blick auf das erzeugte Image für *qemu*; die Anzeige erfolgt über das X-basierte GUI Sato (Abb. 3).

– Python 2.7.3 oder höher; Ausnahme: Python 3.x funktioniert nicht.

Nicht so wählerisch zeigt sich Yocto bei der Linux-Distribution des Hosts: Von CentOS bis Ubuntu sind alle weitverbreiteten Derivate willkommen [e].

Zusätzlich benötigt Ubuntu noch einige Pakete. Das Yocto-Team unterteilt sie in die vier in der Tabelle „Yocto-Befehle“ genannten Gruppen: Wer ohne das als ADT bezeichnete Cross-Plattform-SDK (Software Development Kit) oder ohne Dokumentation auskommt, überspringt die damit befassten Passagen.

Wenn das für diesen Artikel verwendete Build-System die oben erwähnten Werkzeuge nicht enthält, kann man das Paket „buildtools-Tarball“ herunterladen. Unter Ubuntu 14.04 entfällt dies erfreulicherweise. Daher geht es im nächsten Schritt weiter mit dem Download des Build-Systems.

Das Yocto-Team bittet darum, dass Nutzer den Code über Git beziehen [f]. Die notwendigen Befehle hierfür:

```
tamhan@TAMHAN14:~$ mkdir yoctoplace
tamhan@TAMHAN14:~$ cd yoctoplace
tamhan@TAMHAN14:~/yoctoplace$ git clone ↵
   -b daisy git://git.yoctoproject.org/poky.git
```

Nach dem erfolgreichen Herunterladen der Dateien enthält der neue Ordner ein Unterverzeichnis namens *poky*. Dort erzeugt das Kommando *source oe-init-build-env* eine leere Einstellungsdatei, Listing 1 zeigt die Antwort des Systems.

Das *source*-Kommando setzt die Einstellungsdatei im Verzeichnis *build* des Image ab. Dessen Unterverzeichnis *conf* enthält die Datei *local.conf*, in der man den Übersetzungsprozess einstellen kann. Da Leser dieser Zeilen an dieser Stelle ein Image für den Emulator *qemu* (Quick EMUlator) erzeugen wollen, können sie die Datei ohne Änderungen übernehmen – die Zuweisung mit den zwei Fragezeichen legt „qemux86“ als Standardwert fest:

```
# Machine Selection
. . .
# This sets the default machine to be qemux86 ↵
                  if no other machine is selected:
MACHINE ??= "qemux86"
```

Die vorliegende Konfiguration lässt sich nun übersetzen, was auf schnellen Stationen locker eine gute Stunde in Anspruch nimmt. Dazu dient der Aufruf des Kommandos *bitbake core-image-sato* im *build*-Verzeichnis des Image.

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

Wegen der durch den Download entstehenden Latenz bleibt die CPU trotz optimaler Einstellung der Thread-Settings am Anfang gering ausgelastet (siehe Abbildung 2). Nach dem Abschluss aller I/O-Operationen geht die Maschine normalerweise auf 100 Prozent.

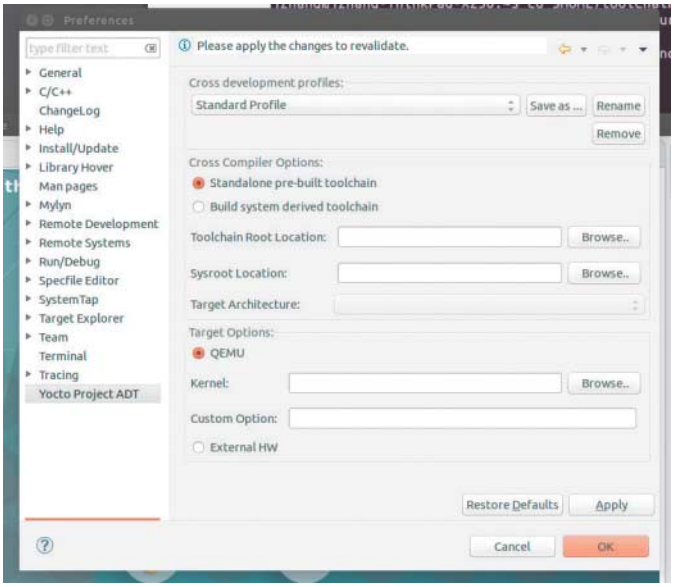
Im Anschluss an den erfolgreichen Build-Prozess lässt sich das Image mit dem Befehl *runqemu qemux86* ausführen. Das dadurch angestoßene Skript fragt mitunter nach dem Root-Passwort. Den Lohn der Mühen zeigt Abbildung 3.

Yocto legt die entstandenen Image-Dateien im Ordner */build/tmp/deploy/images/<plattform>* ab. Erst der nächste Teil der Serie wird sich mit dem Deployment auf echte Hardware befassen – im Moment reicht es aus, wenn sich das Image im Emulator *qemu* ausführen lässt.

Yoctos Entwicklerwerkzeug

Cross-Compiler waren in der Anfangszeit der Informatik ein höchst seltenes Gut, Programme entstanden meist auf dem System, auf dem sie später laufen sollten. Im Laufe der Zeit benötigten Entwickler wegen immer kleinerer Zielsysteme Cross-Compiler: Ein AVR-Mikrocontroller lässt sich nicht dazu vergattern, ein C-Programm zu übersetzen. Auf Wunsch erzeugt Yocto eine ADT genannte bezeichnete Toolchain, die auf die Bedürfnisse des System-Image zugeschnitten ist. Aufgrund der enormen Flexibilität des Produkts warten an dieser Stelle einige Besonderheiten auf die angehenden Distributoren.

Die Datei *Local.conf* enthält die folgende Passage, die die Auswahl des x86-Zielsystems für die Binärdateien erleichtert. Ein auf einem 64-Bit-Host laufendes



Das Yocto-Plug-in von Eclipse benötigt Informationen über seine Arbeitsumgebung (Abb. 4).

Yocto kann auf Wunsch eine Werkzeugkette für 32-Bit-Prozessoren generieren:

```
# SDK/ADT target architecture
. . .
# Supported values are i686 and x86_64
#SDKMACHINE ?= "i686"
```

Wer schon einmal Code für den Raspberry Pi kompiliert hat, kennt sicher den Fall mit dem dafür einzuhängenden, als *sysroot* bezeichneten Karten-Image. Es ist ein exaktes Abbild der Root-Partition, die dem Cross-Compiler die Arbeit erleichtert. Die mit der Verwendung eines selbst auf Basis eines Zielsystems erstellten *sysroot* einhergehenden Unannehmlichkeiten – zusätzliche „mitzuschleifende" 4 GByte Daten sowie das obligatorische Mounten – lassen sich am einfachsten umgehen, indem man das Verwalten von *sysroot* und Co. an Yocto abtritt. Dazu genügt es, BitBake durch den folgenden Befehl *bitbake adt-installer* zum Erstellen eines Installationsmediums zu animieren

Direkt nach dem Erzeugen des Abbilds ausgeführt, sollte der Befehl nur wenige Sekunden benötigen. Yocto enthält einen hocheffizienten Cache, der einmal kompilierte Binärdateien vorhält – doch dazu mehr in einem der folgenden Teile.

Der ADT-Installer liegt dann im Verzeichnis */build/tmp/deploy/sdk*. Das Auspacken des Archivs legt die in ihm enthaltene Konfigurations- (*adt_installer.conf*) und Installationsdatei frei. Dort kann man die zu konfigurierenden Werkzeuge sowie die jeweils herunterzuladenden Images wählen – von Haus aus ARM und X86. Das Eingrenzen der Architekturen reduziert die Installationszeit erheblich, da der Download der rund 500 MByte großen Images entfällt (siehe Listing 2). Auf den Rechner gelangt das ADT durch folgenden Befehl:

```
tamhan@TAMHAN14:~/yoctoplace/poky/build/tmp/ ⤸
      deploy/sdk/adt-installer$ ./adt_installer
Please enter the install location (default: ⤸
                                  /opt/poky/1.6.1):
```

Während des Installierens kommt es mitunter zu Fehlermeldungen wie unten dargestellt, die darauf hinweisen, dass

Listing 2: *adt_installer.conf*

```
# The following are for system wide setup

# Target architectures that you want to setup host cross dev environment for
# valid values are: arm, x86, x86_64, ppc, mips with space separation
# between entries
YOCTOADT_TARGETS="arm x86"
# Whether install qemu or not, valid entries are: Y/N
YOCTOADT_QEMU="Y"
# Whether install user-mode nfs or not, valid entries are: Y/N.
# If you want to use Yocto Eclipse plug-in as your dev IDE,
# you need to select both qemu and NFS
YOCTOADT_NFS_UTIL="Y"
```

Listing 3: ADT installieren

```
Successfully installed selected native ADT!
Installing target sysroot for arch: arm, rootfs type: sato-sdk, location:
                                                  /home/tamhan/test-yocto/qemuarm
Creating directory /home/tamhan/test-yocto/qemuarm...
Extracting rootfs: core-image-sato-sdk-qemuarm.tar.bz2, using pseudo...
Installing target sysroot for arch: x86, rootfs type: sato-sdk, location:
                                                  /home/tamhan/test-yocto/qemux86
Creating directory /home/tamhan/test-yocto/qemux86...
Extracting rootfs: core-image-sato-sdk-qemux86.tar.bz2, using pseudo...
############################################################
# Yocto ADT has been successfully installed.
############################################################
```

Es ist so weit: „Hello World" ist im Sato-Terminal lauffähig (Abb. 5).

Dateien umgezogen sind. Bei Bedarf muss man das Installationsskript anpassen:

```
HTTP request sent, awaiting response...  ↵
                          301 Moved Permanently
Location: ↵
       http://adtrepo.yoctoproject.org/1.6.1/ ↵
         [following] --2014-09-13 14:22:48-- ↵
        http://adtrepo.yoctoproject.org/1.6.1/
Reusing existing connection to ↵
                     adtrepo.yoctoproject.org:80.
HTTP request sent, awaiting response... 200 OK
Length: 1097 (1,1K) [text/html] ↵
                               Saving to: '1.6.1'
```

Nach der Eingabe des Root-Passworts erzeugt die Installationsroutine einen Block mit Informationen, die für die weitere Arbeit mit Yocto wichtig sind. Listing 3 zeigt, wie das auf der Maschine des Autors aussah.

Eclipse installieren

Puristen entwickeln ihre Programme mit der lokalen Installation des ADT. Als mit Visual Studio aufgewachsene Person empfindet der Autor dieser Zeilen die Arbeit mit einer „richtigen IDE" als bequemer: Die Eclipse-Integration erweist sich aus diesem Blickwinkel gesehen fast als ein Gottesgeschenk.

Ärgerlicherweise gilt auch hier, dass nichts so einfach ist, wie es auf den ersten Blick scheint, denn das Yocto-Team bietet seine Plug-ins im Moment nur für die Releases 4.2 (Juno) und 4.3 (Kepler) an. Vom Eclipse-Projekt kann man die für die jeweilige Arbeitsumgebung passende Version der freien Entwicklungsumgebung herunterladen [g] und wie gewohnt installieren. Danach führt der Klick auf „Help –> Install new Software" im gestarteten Eclipse zu einem Dialog, in dem man „Work With Kepler http://download.eclipse.org/releases/kepler" auswählt, um die diversen verfügbaren Erweiterungen anzuzeigen. Die Arbeit mit Yocto setzt folgende Module voraus:

– Linux Tools
– LTTng (Linux Tracing Toolkit)
– Mobile and Device Development
– C/C++ Remote Launch (erfordert RSE Remote System Explorer)
  – Remote System Explorer End-user Runtime
  – Remote System Explorer User Actions
  – Target Management Terminal
  – TCF Remote System Explorer Add-in
  – TCF Target Explorer
– Programming Languages
– C/C++ Autotools Support
– C/C++ Development Tools

Manche Versionen von Eclipse enthalten einige der hier erwähnten Elemente schon von Haus aus – in diesem Fall entfällt eine Neuinstallation. Im Anschluss an den obligatorischen Neustart von Eclipse öffnet man das „Help –> Install new Software"-Fenster abermals und weist dem Feld „Work With" diesmal die entsprechende Yocto-URL http://downloads.yoctoproject.org/releases/eclipse-plugin/1.6/kepler zu.

Nach dem Auswählen und Herunterladen aller dort befindlichen Plug-ins ist erneut ein Eclipse-Neustart erforderlich. Die in Abbildung 4 gezeigten Einstellungen für das Yocto-Plug-in finden sich in Eclipse unter „Windows –> Preferences –> Yocto Project ADT".

Da Embedded-Entwickler oft mit verschiedenen Hardwarekomponenten parallel arbeiten, können sie mit dem Toolkit ADT verschiedene Profile anlegen. Für dieses Artikelprojekt reicht es, die Standardversion an die konkrete Situation anzupassen.

Endlich: Hello World!

Die vom ADT-Installer erzeugte Toolchain liegt im Verzeichnis */opt/poky/1.6.1*, während sich das *sysroot* im Verzeichnis *~/test-yocto/qemux86* befindet. Die Option „Standalone pre-built toolchain" bleibt vorerst selektiert, im Architekturfeld ist „i586-poky-linux" auszuwählen.

Als Laufzeitumgebung dient *qemu*. Mit dem Browse-Button kann man den zu verwendenden Kernel festlegen. Danach ist im Ordner */poky/build/tmp/deploy/images/qemux86* die Datei *bzImage-qemux86.bin* auszuwählen. Das für Hardwareeinstellungen vorgesehene Feld „Custom Option" bleibt leer, da die Default-Werte in Ordnung sind.

Damit ist Yocto konfiguriert. Anschließend sollte man ein neues Projekt vom Typ „C / C++ – C Project" anlegen, als Untertyp dient „Yocto Project ADT Autotools Project – Hello WorldANSI C Autotools Project". Die restlichen Einstellungen sind eins zu eins aus den Vorgaben zu übernehmen.

Ein Klick auf „Run –> External Tools –> <Imagename>" – im Fall dieses Artikels „qemu-i586-poky-linux" – startet den eigentlichen Emulator. Wenn Eclipse dabei nicht abstürzt (was im Setup des Autors durchaus passierte), so erscheint ein Terminalfenster am Bildschirm. Dieses fordert zum Eingeben des Root-Passworts auf, da es den Emulator in den Netzwerkstack der Workstation einbinden muss. Auf machen Maschinen erscheint zudem eine Aufforderung, den Portmapper *rpcbind* einzuspielen, der man nachkommen muss. Ab diesem Zeitpunkt lässt sich der Emulator aus Eclipse heraus starten.

In der Rubrik „All" bietet der Launcher ein Terminalfenster an. Mit dem dort

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.

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 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.

eingegebenen Befehl *ifconfig* erhält man die lokale IP-Adresse des Abbilds. Sie lautet normalerweise 192.168.7.2, kann sich aber aus diversen Gründen ändern.

Da die hier benutzte Entwicklungsumgebung auf Eclipse basiert, fehlt noch eine Konfiguration für die Ausführung. Unter „Run –> Debug Configurations“ wartet eine beim Anlegen des Projekts generierte Vorlage auf Parameter. Das geschieht durch die Wahl des „New“-Buttons neben der Connection-Combobox. Als Verbindungstyp ist hier „SSH Only“ zu wählen, die IP-Adresse wandert in das Feld „Host Name“, ein Klick auf „Next“ startet die nur bei laufendem Emulator erfolgreich durchführbare Verifikation. Die restlichen Parameter des Verbindungsassistenten kann man eins zu eins übernehmen.

Debugge mich!

Im Fenster mit der neuen Debug Configuration erscheint die soeben erzeugte Verbindung im Feld „Connection“. Der Eintrag in „Remote Absolute Path“ legt den am Zielsystem zu verwendenden Pfad fest: Yocto-Entwickler wählen normalerweise */usr/bin/<appname>*. Ein Klick auf „Debug“ öffnet den Dialog zur Eingabe des Passworts. Frisch generierte Images lassen sich mit dem User „root“ ansprechen, das Password-Feld darf leer bleiben. Dann erscheint die Frage nach der Authentizität des Schlüssels, und der Deployment-Prozess für die Applikation ist beendet.

Endlich Hello World: Das Programm lässt sich im Terminal anzeigen und ausführen. Auf Wunsch kann der Nutzer auf die Funktionen des in Eclipse integrierten Debuggers zurückgreifen, um nach Programmfehlern zu jagen.

Manche Images sind für den Emulator nicht ansprechbar. Applikationen lassen sich in diesem Fall nicht direkt im Emulator ausführen. Das liegt meist daran, dass Entwickler ihre jeweiligen Distributionen ohne die für die Kommunikation mit Eclipse notwendigen Werkzeuge generiert haben.

Es spricht aus technischer Sicht nichts dagegen, das Image in *qemu* zu booten und die fehlenden Komponenten „on the fly“ zu installieren. Auf die Dauer erscheint dies aber nicht sinnvoll, da sonst die bisherigen Schritte bei jedem neuen System abzuarbeiten sind.

Viel intelligenter wäre es, wenn der Image-Generator die notwendigen Komponenten gleich beim Zusammenbau an ihren rechten Platz schiebt. Einstellen lässt sich das in der Datei *local.conf*. In der Rubrik „EXTRA_IMAGE_FEATURES“ kann man die *debug-tweaks* zuweisen und so eine einfache Debugger-Unterstützung aktivieren. Fortgeschrittenere Optionen beschreibt die über dem Schalter befindliche Dokumentation.

Onlinequellen

[a]	Yocto-Projekt	www.yoctoproject.org
[b]	Teilnehmer an Yocto	www.yoctoproject.org/ecosystem/yocto-project-participants
[c]	OpenEmbedded	www.openembedded.org/wiki/Main_Page
[d]	Layer aus OpenEmbedded	layers.openembedded.org/layerindex/branch/master/layers/
[e]	Systemanforderungen	www.yoctoproject.org/docs/1.6.1/ref-manual/ref-manual.html#intro-requirements
[f]	Download Yocto	www.yoctoproject.org/downloads
[g]	Download Eclipse	www.eclipse.org/downloads/packages/eclipse-ide-cc-developers/keplersr2

Fazit

Seit jeher ist das Entwickeln von Embedded-Software eine anspruchsvolle Aufgabe, die sich Interessierten oft erst auf den zweiten Blick erschließt. Der Artikel vermittelt ein grundlegendes Verständnis der Struktur des hinter Yocto stehenden Build-Systems und der dazugehörenden Werkzeuge.

Zurzeit ist das generierte Image nur im Emulator lebensfähig. Der nächste Teil der Serie wagt dann den Sprung auf den Raspberry Pi. Außerdem ändert er einige für den Zusammenbau des Image verwendete Rezepte. Der dritte und letzte Teil portiert das Image auf den Arduino Galileo und stellt einige Tricks und Besonderheiten vor.

Bis dahin: Gut Linux! (avr)

Tam Hanna

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

Alle Links: www.ix.de/ix1502044