Embedded Linux

danke Yocto und OpenEmbedded

Von Marco Israel

Dezember 2019

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

DTB Device Tree Blob

1 Einleitung

1.1 Über Yocto, OpenEmbedded, poky, meta-daten und Bitbake

Das Yocto Project bezeichnet eine Community Gruppe welche sich zur Aufgabe gemacht hat, dass erstellen von Linux Distributionen für eingebettet System zu vereinfachen.

Zusammen mit der OpenEmbedded Community pflegt das Yocto Project eine Software Build Umgebung mit dem Namen Bitbake, bestehend aus Python- und Shell-Scripten, welches das Erstellen von Linux Distributionen koordiniert. Parallel zu Bitbake entstehen und wachsen innerhalb der Yocto Community verschiedene tools wie z.B. Wrapper, welchen das Arbeiten mit Bitbake vereinfachen sollen, indem diese Tools beispielsweise einzelne Workflows abbilden.

Des Weiteren pflegen beiden Communities Dateien mit Metadaten (Metadatendatei die in Form von Regeln beschreiben wie Software Pakte innerhalb für unterschiedliche Distribution und Hardware durch Bitbake gebaut werden müssen. erforderlich sind.

Diese Regeln werden Recipies genannt. Regeln bzw. diese Metadaten und somit auch das Build-System Bitbake arbeiten nach einen Schichten Modell. Dabei beschreiben Meta-Daten auf unterster Schicht allgemeine grundlegende Anleitungen zum Übersetzten der wichtigsten Funktionen eines Betriebssystems. Höhere Schichten erweitern (detaillieren) oder überschreiben diese grundlegenden Rezepte in immer höheren Schichten. So setzt auf Beschreibungen der Hardware (BSP, Board Support packed Schichten) Schichten der Software- und Anwendungsschicht auf. Ein solches Schichtenmodell ermöglicht es, einzelne Schichten auszutauschen oder darunterliegende Einstellungen abzuändern. Die Yocto community stellt Meta-Daten / Recipes bereit, um ein minimalistisches Betriebssystem mit grundlegenden Linux Tools unter der Virtualisierungsumgebung QEMU für verschiedene Architekturen starten zu

können. Die OpenEmbedded Community stellt Meta-Daten Rezepte bereit, die auf dieses minimalistische Betriebssystem aufsetzten und genutzt werden können um ein Betriebssystem nach eigenen Wünschen gestalten (zusammen setzten) und für Hardware Plattformen konfigurieren zu können. Hierzu gehören sowohl Hardware als auch open Source Software Beschreibungen. Beide Communities pflegen und erweitern das Build Umgebung „Bitbake“, welches verschiedene Aufgaben in geregelter Reihenfolge im Multicore Betrieb durchführt:

- Herunterladen (fetchen) und Sammeln von Source Dateien (z.B. Git Repositorien, ftp Servern oder lokalen Dateien.)
- Übersetzten, konfigurieren, patchen, installieren, verifizieren usw. von Paketen auf mehreren Prozessor Kernen
- Bereitstellen von Entwicklung und Verwaltungstools

Das Minimalisten Betriebssystem der Yocto Community wird „poky“ genannt.

1.2 Docker

Bitbake benötigt, neben einem aktuellen Python (2) Interpreter, verschiedene Tools. So u.a. Git zum Herunterladen von Source Detain. Alle diese Abhängigkeiten wurden in einem Docker Container zusammengefasst. Bitbake kann unter unschädlichen Betriebssystemen und Plattformen auf gleiche Weise genutzt werden, ohne dass es und seine Abhängigkeiten neu konfiguriert werden müssen. Im Gegensatz zu anderen Virtualisierungstechniken hat Docker trotz Virtualisierungstechniken keinen großartigen Performance Verlust. Von einem Gebrauch von klassischen Virtualisierungstechniken ist aus Performance Gründen dringend abzuraten.

2 Setup Eclipse

2.1 Installation, Einrichtung und Plugins

Zur Entwicklung von C/C++ kann Eclipse CDT verwendet werden.

Zudem sind zum Cross compilieren und Remote Debuggen sowie zum remote Deployen nachfolgende Plugins bzw zusätzliche Eclipse-Softwaremodule nötig: (*Help -> Install new Software*)

- C/C++ Remote (Over TCF/TE) Run/Debug Launcher.
- Remote System Explorer User Actions
- TM Terminal via Remote System Explorer
- TCF Target Explorer

Des weiteren stellt die Yocto download Seite downloads.yoctoproject.org ein SDK Plugin bereit, welches das Konfigurieren von Remote Einstellungen für jeweilige Entwicklungsprojekt vereinfacht und zusammenfasst. Zum installieren muss beispielsweis zu den Installationsquellen in eclipse **http://downloads.yoctoproject.org/releases/eclipse-plugin/2.6.1/oxygen/** hinzu gefügt werden.

2.2 Cross Compile und Remote Debugging Einstellungen

TODO

3 Commands Bitbake

NOTE: Ein *Image* ist in der Bitbake welt ebenso ein *recipie* und wird entsprechend gleich behandelt. Alle befehle die für ‚einfache'recipes gelten (z.b. Kernel Modules oder Applikationen), gelten ebenso für image-recipes.

3.1 Bitbake Quellen

Eine die offizielle Dokumentation zu *Bitbake* und *Bitbake-Layers* mitmitsamt Kommandoist zu finden unter: . Eine Übersicht über die Kommandos von Bitbake-Layers ist zu finden unter [Gon18, 156] und im Anschluss in Anwendung gezeigt.

3.2 Neuer *meta-layer*

Nachfolgende möglichkeiten stehen bereit, um einen neuen Meta-daten layer zu erstellen:

- yokto-layer create <myLayer> (**Empfohlen**)
- bitbake-layers create-layer <mylayer>

Wie sich einer neuer meta-layer zum Build hinzufügen lässt, ist nachzulesen unter 3.3, Seite 5.

3.3 meta-layer zum build hinzufügen

Um einen neuen Metadaten-layer zum Buildsystem hinzu zu fügen, muss dieser in die datei */conf/bblaysers.conf* Bitbake bekant gegeben werden. Hierzu stehen nachfolgende Möglichkeiten bereit:

- Manueles Eintragen der des Pfades zum meta-layer (**Empfohlen**)
- bitbake-layers add-layer <layername>.

3.4 Active meta-layer auflisten

Um alle aktiven meta-layer aufzulisten die im build einbezogen werden, stehen nachfolgenden Möglichkeiten bereit:

- Manuelles Einsehen der Datei con/bblayers.con
- bitbake-layers show-layers (**Empfohlen**)

3.5 Anzeigen aller recipes - beispielsweise aller images

Recipies die während eines Builds aktive sind und ausgeführt werden lassen sich wie folgt anzeigen:

- bitbake-layer show-recipes [<recipes>]
- bitbake-layer show-recipes [<recipes>] | grep image

3.6 Anzeigen der Tasks eines recipes

Jedes Recipes besitzt eine eine Anzahl von standard Tasks die es direkt oder indirekt implementiert hat oder erbt. Alle Task die eine recipe besitzt und somit während eines builds durchlaufen werden lassen sich anzeigen durch:

- bitbake -c listtasks <recipes>

Wie gezielt einzelne bzw. bestimmte Task eines Recipes ausgeführt werden ist unter 3.7, Seite 6 beschrieben.

3.7 Ausführen bestimmter Tasks eines Recipes

Der Parameter *bitbake -c* (bitbake -command) ermöglicht das Ausführen bestimmter Tasks eines Recipes:

- bitbake -c <cmd> <recipes>

3.8 Umgebungsvariablen eines Recipes

Die Umgebungsvariablen welche eine Recipe dauerhaft oder temporär setzt, erweitert oder löscht, lassen sich wie folgt anzeigen.

- bitbake -e <recipes>

Es kan geziehlt nach einzelnen Variablen gefiltert werden:

- bitbake -e <recipes> | grep ^<ENVVARIABLE>

3.8.1 Häufig benötigte Umgebungsvariablen

- bitbake -e virtual/kernel | grep **^SRC_URI=**
- bitbake virtual/kernel -e | grep **„PREFERRED_PROVIDER_virtual/kernel“**

3.9 Erzeugen von temporären entwicklungs-Shells

Bitbake ist in der Lage temporäre, gekapselte und vorkonfigurierte Entwicklungs-Shells zu erzeugen und zu öffnen:

- bitbake -c devshell <recipes>
- bitbake -c devpyshell <recipes>

Dabei unterscheidet bitbake zwischen einer normalen Bash-Shell und einer Phython-Shell (Die beiden Skriptsprachen welche innerhalb bitbake Anwendung finden. Beispielsweise wahlweise Python als auch Bash-Shell commandos (kombiniert) innerhalb eines Recipe verwenden.

4 Lösung für bekannte Fehler

4.1 ERROR: Fehler beim Bauen eines recipes>

4.1.1 lösung

1. Run *bitbake -c cleanall <recipes>*
2. If this not work, clean (deleate) all:
   - *sstate* folder
   - *deploy* folder
   - *tmp* folder
   - everythink else in the project folder, exapting / but not the conf folder

5 Source code development und debugging

5.1 Cross development

Es existieren verschiedene Wege um Anwendungen für Zielplattformen zu entwickeln. Yocto setzt dabei auf die beiden nachfolgenden; zum einen die Bereitstellung einer gekapselten Entwicklungsumgebung in Form eines SDKs, zum anderen die Entwicklung unter Zuhilfenahme der Buildumgebung Bitbake mitsamt seiner Tools.

Nachfolgend beide Wege im Überblick:

5.1.1 Entwicklung mit dem Yocto SDK

Dieser Weg dient der normalen Entwicklung von Softwarekomponenten. Das Yocto SDK liefert eine gekapselte Shell-Umgebung mit vordefinierten Umgebungvariablen sowie allen nötigen Header Dateien, Libraries usw. welche auf der Zielplattform vorhanden sein werden. Hierzu bildes es die Ordnerstruktur (das s.g. rootfs) der Zielplattform in einem Ordner ab.

1. **Shell Umgebungsvariablen laden** Um die SDK Umgebung zu nutzen ist es nötig, eine Konfigurationsdatei in die Shell zu laden.

Listing 5.1: Einrichten der SDK Umgebung

```
1 [user@host]/yoctopath $: source ./enviroment-setup-*.sh
```

2. **Eclipse starten** Abhängig vom Anwendungfall, anschließend in der selben Shell Eclipse oder QTCreator starten.

Listing 5.2: Start von Eclipse in der zuvor konfiguierten Umgebung

```
1 [user@host]/yoctopath $: eclipse &
```

5.1.2 Entwicklung mit Yocto Tools

Diese Software Entwicklungsart dient mehr dazu Änderungen an existierendem Sourcecode durchzuführen. Hierbei stehen zwei tools zur Verfügung:

devtool <cmd> Zum einen steht das Yocto-Tool **devtool** mit verschiedene Kommandos bereit. Eine Übersicht über Kommandos und Anwendungsbeispiele liefert die Yocto Webseite oder *devtool –help*.

bibake -c devshell <recipie> Erzeugt bzw. öffnet eine vorkonfigurierte Shell ensprechend den Metadaten eines *recipie* zu einer Source Datei.

5.2 Remote Debugging

6 Ausblick

6.1 Nächste Schritte

6.1.1 Entwicklungswerkzeuge als Docker Container

Es wäre sinnvoll die Entwicklungstools in einem oder mehrere Docker container vorkonfiguiert zur verfügug zu stellen. Hierzu zählen unter anderem:

Eclipse Vorkonfiguriertes Eclipse inclusieve Plugins, Cross-Compile und Remote debugging Einstellungen.

QT5 Vorkonfiguiertes QT mit Crosscompile und Remot Debugging

TFTboot und NFSROOT Server in einem Container vorkonfiguriert bereitstellen

6.1.2 Erweiterung der Scripte

Das *run.sh* script sollte so erweitert werden, das es ‚post'oder ‚pre'Aufgaben vor oder nach dem aufrufen der *dockerjobs.sh* durchführt oder andere postbuild oder prebuild scripte aufruft. Denkbar wären:

- Kopieren des zImages und Device Tree Blob (DTB) in das *TFT-boot* Verzeichnis
- Kopieren und extrahieren des rootfs in das nfs-rootfs Verzeichnis

Das *run.sh* script erzeugt das *dockerjobs.sh* Script sollte dem run.sh script parameter übergeben werden. Anschließend startet das run.sh script den Docker container in definierter version (gesetzt über Parameter oder direkt innerhalb des run.sh scripts). Das dockerjobs.sh wird innerhalb docker durch das image aufgerufen und enthält alle aufgaben welche durch den Container in batchmode erfüllt werden sollen. Das dockerjobs.sh script lässt sich manuell erweitern / erstellen. Es wird nur überschrieben, wenn dem run.sh Ausführungsbefehle übergeben werden.

Literaturverzeichnis

[Gon18] Gonzalez, Alex: *Embedded Linux Development Using Yocto Project Cookbook* -. 2nd Revised edition. Birmingham : Packt Publishing, 2018. – ISBN 978–1–788–39921–0

addbibresourcebibtex