Embedded Linux mit Yocto

in 2026

Michael Estner

Linux-Infotag Augsburg
May 2, 2026

Agenda

1. Yocto/Openembedded
2. bitbake
3. meta-layer
4. Demotime
5. bitbake-setup und kas
6. Nützliche Tools

Whoami

- Name: Michael Estner
- Linux Architekt @ Elektrobit
- Embedded Linux, Secure boot, Netzwerk, Kernel
- Nachts: Master Fernuni Hagen und Yocto Projekt

Yocto/Openembedded

Was ist Embedded Linux?

- **Embedded Linux** bezeichnet den Einsatz des Linux-Betriebssystems in eingebetteten Systemen (Embedded Systems).
- Ein **Embedded System** erfüllt eine dedizierte Funktion
- Typische Einsatzbereiche:
  - Industrieanlagen und Automatisierung
  - Automotive (z.B. Infotainment, Steuergeräte)
  - Consumer Electronics (z.B. Smart TVs, Router)
  - IoT-Geräte und Edge-Devices
- **Warum Linux?**
  - Open Source und hochgradig anpassbar
  - Große Hardware-Unterstützung (Architekturen, Treiber)
  - Stabiles und bewährtes Ökosystem

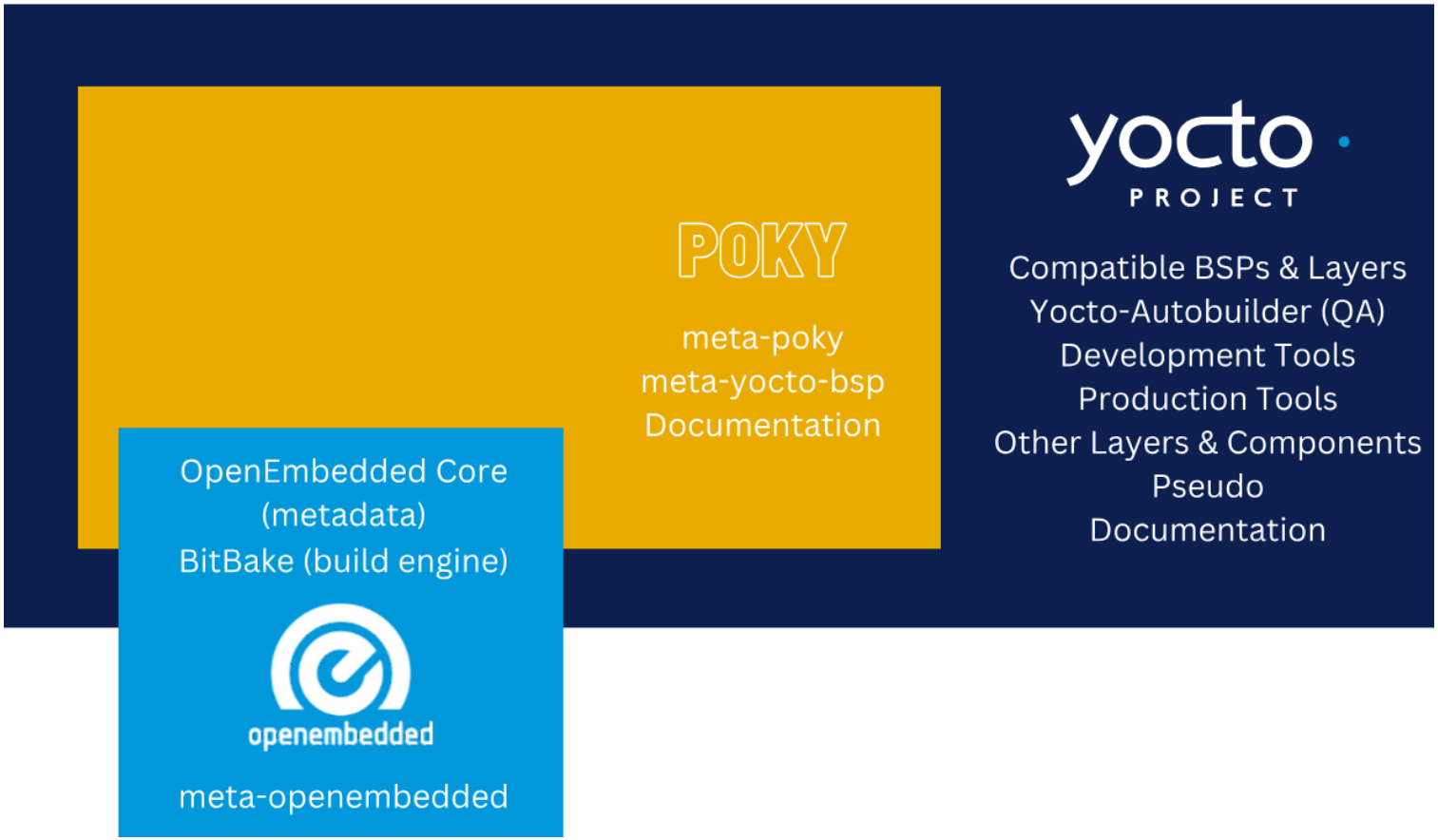
OpenEmbedded: Wurzeln

- **Beginn Anfang 2000**
  - Community getriebenes Projekt für Embedded Linux
  - Ziel: Flexibles und Wiederverwendbares Buildsystem
- **Motivation:**
  - Cross-compilation für viele Hardware Architekturen
  - Manuelle Systemintegration vermeiden
- **Kernkonzept:**
  - Recipes (.bb files) beschreiben wie Software gebaut wird
  - `BitBake` als Build engine
  - Openembedded stellt die meta-layer bereit
- $\Rightarrow$ Grundlage für moderne Yocto Embedded Linux Systeme

Yocto Project: Grundlagen

- **Gegründet:**
  - 2010
  - Projekt der Linux Foundation
  - Kooperation zwischen der Industrie (Intel, NXP, usw.)
- **Zweck:**
  - Bereitstellung eines Frameworks zum Erstellen maßgeschneiderter Linux-Distributionen
  - Bereitstellung einer Referenzdistribution: poky
  - Bereitstellung des Projektökosystems und Dokumentation
- **Kernidee:**
  - Keine out of the box Distribution
  - Werkzeugkasten zur Erstellung deines Embedded Linux
- $\Rightarrow$ Standardisierter, skalierbarer Ansatz zur Entwicklung von Embedded-Linux-Systemen

Yocto/Openembedded



[7]

bitbake

BitBake: Die Yocto-Build-Engine

- BitBake dient als Task-Scheduler
- Es parst textbasierte Dateien wie .bb, .conf und .bbclass
- Hauptaufgabe ist die Ausführung einer Abfolge von Tasks
- BitBake ist von GNU Make inspiriert

Task-Ausführung und Abhängigkeiten

- BitBake zerlegt den Build-Prozess eines Rezepts in atomare, diskrete Tasks wie `do_fetch`, `do_compile` und `do_install`.
- Abhängigkeiten werden auf **Task-Ebene**verwaltet, wodurch BitBake Tasks aus verschiedenen Rezepten parallel ausführen kann.
- Zur Leistungsoptimierung speichert der **Shared State Cache (Sstate)** Task-Ergebnisse zur Wiederverwendung
- Es erstellt eigene ”native” Versionen von Host-Tools und verwendet diese (wie Compilern)

Praktische BitBake-Nutzung

- Vor der Nutzung muss die Build-Umgebung mit dem Skript `oe-init-build-env` initialisiert werden.
- **Standard-Build:** Verwenden Sie `bitbake <target>` (z.B. `bitbake core-image-minimal`)
- **Ausführung spezifischer Tasks:** Das `-c`-Flag ermöglicht das Ausführen bestimmter Tasks, z.B. `bitbake -c menuconfig virtual/kernel` zur Kernel-Konfiguration.
- **Inspektionswerkzeuge:**
  - `bitbake -s`: Listet alle verfügbaren Rezepte und deren Versionen auf.
  - `bitbake -e <recipe>`: Gibt die vollständige BitBake-Umgebung und Variablenexpansionen aus.
  - `bitbake -g -u taskexp <target>`: Öffnet einen grafischen Abhängigkeitsbaum-Explorer.

meta-layer

Die Meta-Layer Struktur

- Metadaten im Yocto Project sind in **Layern** organisiert
- **Konvention:** Layer-Namen beginnen mit `meta-`
- Ein Layer ist eine Sammlung von Rezepten, Konfigurationsdateien und Klassen, die einen gemeinsamen Zweck erfüllen
- Jeder Layer **muss** ein Verzeichnis `conf/` mit der Datei **`layer.conf`** enthalten =¿ Einstiegspunkt für BitBake

Layer-Hierarchie und Prioritäten

- Layer können übereinander gestapelt werden; die Variable **`BBFILE_PRIORITY`** entscheidet bei Konflikten, welches Rezept gewinnt
- **Das wichtigste Yocto-Gesetz:** Modifizieren Sie niemals Core-Layer
- Eigene Anpassungen erfolgen immer in einem **Custom-Layer**
- Das Tool `bitbake-layers` hilft beim Verwalten und Erstellen neuer Layer

Machine Configuration (.conf)

- Die Machine-Definition beschreibt die **Hardware-Eigenschaften**des Zielgeräts
- Diese Dateien befinden sich unter `conf/machine/<machine>.conf` innerhalb eines Layers
- **Zentrale Aufgaben:**
  - Festlegung der Ziel-Architektur (`TARGET_ARCH`)
  - Auswahl des Kernels und Bootloaders (`PREFERRED_PROVIDER`) Definition von Hardware-Treibern und Device Trees

Machine Features und Tuning

- **`MACHINE_FEATURES`** listet Hardware-Funktionen wie `wifi` oder `bluetooth`
- Steuert ob Rezepte entsprechende Pakete oder Treiber automatisch in das Image aufnehmen
- Include-Dateien (`.inc`) definieren spezifische Prozessor-Optimierungen

Distribution Layer (Distro)

- Die Distribution legt die **Software-Policies** fest, die für das Projekt gelten
- **Typische Entscheidungen:**
  - Wahl der C-Bibliothek (glibc vs. musl)
  - Wahl des Init-Managers (systemd vs. SysVinit)
  - Wahl des Paketformats (rpm, deb, ipk)
- Poky ist die Referenz-Distribution, muss aber für die Produktion durch eine eigene Distro ersetzt werden

Anpassung der Distribution

- Konfigurationsdateien liegen unter `conf/distro/<distro>.conf`
- **Pflicht-Variablen:** `DISTRO_NAME` und `DISTRO_VERSION`
- **`DISTRO_FEATURES`** definiert globale Software-Unterstützung (z.B. `ipv6`, `wayland`)
- Eigene Distros ermöglichen es Policies sauber von Hardware-Definitionen zu trennen

BitBake Rezepte (.bb)

- Rezepte sind das Herzstück; sie beschreiben, wie eine einzelne Softwarekomponente gehandhabt wird
- Sie enthalten Anweisungen zum **Abrufen (do_fetch)**, **do_unpack**, **do_patch**, **do_compile** und **do_install**
- Der Output eines Rezepts ist typischerweise ein Satz von Binärpaketen
- Rezepte definieren sowohl Build-Zeit- (`DEPENDS`) als auch Laufzeit-Abhängigkeiten (`RDEPENDS`)

Anatomie eines Rezepts

Ein Rezept lässt sich in drei Teile gliedern:

1. **Header:** Beschreibung, Homepage und Lizenz (`LICENSE`)
2. **Quellen:** Woher kommt der Code? (`SRC_URI`, `SRCREV`)
3. **Tasks:** Auszuführende Funktionen (z.B. `do_compile`, `do_install`)

BitBake Classes (.bbclass)

- Klassen bieten eine **Abstraktion für gemeinsamen Code**, der in mehreren Rezepten wiederverwendet werden kann
- Rezepte nutzen Klassen mit dem Keyword **`inherit`**
- **Wichtige Standard-Klassen:**
  - `base.bbclass`: Wird automatisch von jedem Rezept geerbt
  - `kernel.bbclass`: Enthält Logik zum Bauen von Linux-Kernel
  - `autotools.bbclass`: Automatisiert Rezepte für Projekte mit Autoconf/Automake
- Sie verhindern Redundanz und vereinfachen die Wartung erheblich

Variablenzuweisung in Yocto/OpenEmbedded

- Variablen sind ein zentrales Konzept zur Steuerung des Build-Verhaltens in Rezepten (.bb) und Konfigurationen.
- **Grundlegende Zuweisung:**
  - `VAR = "value"`: Setzt den Wert
  - `VAR := "value"`: Sofortige Zuweisung
- **Schwache Zuweisung:**
  - `VAR ?= "value"`: Setzt nur, wenn noch nicht definiert.
  - `VAR ??= "value"`: Setzt einen Default-Wert mit sehr niedriger Priorität.
- **Override-Syntax:**
  - `VAR:machine = "value"`: Gilt nur für bestimmte Maschinen.
  - `VAR:append:class-target = " value"`: Kontextabhängige Erweiterung.
- Das Verständnis der Evaluationsreihenfolge ist wichtig

Append, Prepend und Remove

- Yocto bietet Mechanismen zur **inkrementellen Modifikation** von Variablen.
- **Append (Anhängen):**
  - `VAR:append = " value"`: Fügt Variable hinzu
  - `recipe.bbappend`: Konfiguration/Ergänzung eines Recipes in anderen layern
- **Prepend (Voranstellen):**
  - `VAR:prepend = "value "`: Fügt Variable am Beginn ein
- **Remove (Entfernen):**
  - `VAR:remove = "value"`: Entfernt spezifische Einträge aus einer Variablen.
- Diese Mechanismen sind essenziell für das Layering und das saubere Überschreiben von Metadaten ohne direkte Modifikation bestehender Rezepte.

Demotime

bitbake-setup und kas

Yocto Setup Tools: kas vs. bitbake-setup

- Moderne Werkzeuge zur Initialisierung von Yocto-Projekten
- Fokus: Reproduzierbarkeit, Automatisierung, Wartbarkeit
- Vergleich zweier Ansätze:
  - kas (externes Tool)
  - bitbake-setup (offizielles Tool)

kas – Konzept

- Tool zur Beschreibung kompletter Yocto-Builds in einer Datei [3]
- Verwendet YAML als Konfigurationsformat
- Ziel: ”One-command build”
- Kümmert sich um:
  - Layer-Checkout
  - Konfiguration
  - Build-Ausführung
- Optional: Containerisierung (kas-container)

kas – Stärken

- Sehr gute Reproduzierbarkeit (inkl. Container)
- YAML:
  - gut lesbar
  - unterstützt Kommentare
- Unterstützt Layering von Konfigurationen
- Automatische Generierung von local.conf
- Weit verbreitet und ausgereift

bitbake-setup – Konzept

- Neues offizielles Tool des Yocto-Projekts[1]
- Stark in BitBake integriert
- Verwendet JSON für Konfiguration
- Fokus:
  - Standardisierung
  - Vereinfachte Projektstruktur
- Nutzt Registry für mehrere Projekte

bitbake-setup – Konzept

- Neues offizielles Tool des Yocto-Projekts
- Stark in BitBake integriert
- Verwendet JSON für Konfiguration
- Fokus:
  - Standardisierung
  - Vereinfachte Projektstruktur
- Nutzt Registry für mehrere Projekte

bitbake-setup – Eigenschaften

- Nutzt ”Layer Fragments” statt local.conf
- Trennt:
  - globale Konfiguration
  - lokale Anpassungen
- Kein automatischer Build (nur Setup)
- Keine Container-Isolation
- Noch in früher Entwicklungsphase

Vergleich: kas vs. bitbake-setup

- **Integration**
  - kas: extern
  - bitbake-setup: Teil des Yocto-Ökosystems
- **Konfiguration**
  - kas: YAML
  - bitbake-setup: JSON
- **Build**
  - kas: Build + Setup
  - bitbake-setup: nur Setup
- **Reproduzierbarkeit**
  - kas: sehr hoch (Container)
  - bitbake-setup: abhängig vom Host
- **Reifegrad**
  - kas: etabliert
  - bitbake-setup: noch in Entwicklung

Nützliche Tools

devshell: Die interaktive Entwicklungsumgebung

- Die **devshell** ist ein spezieller BitBake-Task, der eine interaktive Shell für ein bestimmtes Rezept öffnet
- **Befehl:** `bitbake -c devshell <recipe>`
- **Funktionsweise:**
  - BitBake entpackt und patcht den Quellcode
  - Ein neues Terminal öffnet sich direkt im Quellverzeichnis des Rezepts
  - Die Umgebungsvariablen (wie `$CC`, `LDFLAGS`) sind korrekt für das Cross-Compiling gesetzt

devshell: Praxisbeispiel

Innerhalb der devshell können Standard-Build-Befehle manuell ausgeführt werden:

```
# In der devshell fuer z.B. connman:
$ ./configure --enable-feature-x
$ make
$ $CC mytest.c -o mytest
```

- **Vorteil:** Man muss nicht den kompletten BitBake-Workflow abwarten, um kleine Codeänderungen zu testen
- **Erweiterung:** `pydevshell` bietet eine interaktive Python-Umgebung

devtool: Der Eckpfeiler des eSDK

- **devtool** ist eine Sammlung von Dienstprogrammen, die den Compile-Test-Debug-Zyklus massiv beschleunigen
- Es unterstützt drei Haupt-Workflows:
  1. **add:** Erstellt automatisch ein neues Rezept aus einem Quell-Tarball oder Repository.
  2. **modify:** Bereitet ein existierendes Rezept für Änderungen vor.
  3. **upgrade:** Aktualisiert ein Rezept auf eine neuere Upstream-Version
- **Workspace:** devtool nutzt einen eigenen (`workspace`), um Änderungen isoliert zu verwalten.

devtool: Der iterative Workflow

Ein typischer Entwicklungszyklus mit devtool:

```
$ devtool modify <recipe>
# Code im Verzeichnis workspace/sources/<recipe> bearb
$ devtool build <recipe>
$ devtool deploy-target <recipe> root@192.168.7.2
$ devtool finish <recipe> meta-custom-layer
```

- `deploy-target` installiert Binärdateien via SSH direkt auf die Hardware, ohne das gesamte Image neu zu flashen
- Nach dem Abschluss (`finish`) wird der Workspace aufgeräumt und ein `.bbappend` oder ein Patch erstellt

recipetool: Das Werkzeug hinter den Kulissen

- **recipetool** bildet die technische Basis für viele Funktionen von `devtool`.
- Es ist darauf spezialisiert, Metadaten in Rezepten zu analysieren und zu manipulieren
- **Kernfunktionen:**
  - Automatisches Erstellen von Rezepten aus Quellcode (Erkennung von Lizenzdateien und Build-Systemen)
  - Hinzufügen oder Ändern von Variablen in bestehenden Rezepten
  - Erstellen und Aktualisieren von `.bbappend`-Dateien.

Was ist mit dem CRA?

Links

- GitHub: kas
- kas Dokumentation (Übersicht)
- kas Getting Started
- Yocto Project Technical Overview
- BitBake User Manual (bitbake-setup)
- Yocto Training (Rootcommit)
- Yocto Training (Bootlin)
- Blog: kas vs. bitbake-setup

Kontakt

Beispiel meta-layer:

- Github Repo

Kontakt

- Linkedin
- michaelestner@web.de

Fragen?

- 3 Setting Up The Environment With bitbake-setup — Bitbake dev documentation.
- The Evolving Landscape of Yocto Project Setup: bitbake-setup vs. KAS.
- Getting Started — kas 5.2 documentation.
- Project Configuration — kas 5.2 documentation.
- Project Configuration — kas 5.2 documentation.
- siemens/kas: Setup tool for bitbake based projects.
- Technical Overview - The Yocto Project.
- Welcome to the kas documentation, a setup tool for bitbake based projects — kas 5.0 documentation.
- Yocto Project and OpenEmbedded development training – Bootlin.
- Yocto Project and OpenEmbedded training – Root Commit.

siemens/kas, November 2025.
original-date: 2017-06-13T12:11:57Z.