Short Dive into Building Automotive ECUs with Yocto

First Steps into Yocto

Andreas Cord-Landwehr

August 12th, 2018

Akademy 2018, Vienna

Introduction

About Me & the Talk

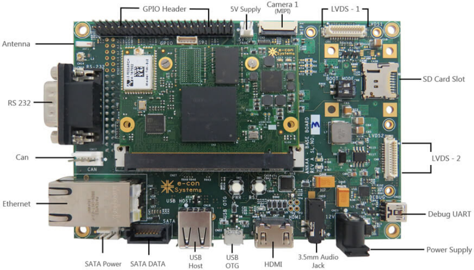
About Me

- IRC-nick: CoLa
- KDE developer since 2010; mostly KDE-Edu
- did PhD in algorithmic game theory
- now working as software developer at CLAAS E-Systems and creating terminals for big agriculture machines
  - could be called “enterprise embedded” development
  - key areas: Qt, C++, embedded Linux, Yocto
  - several Yocto-based ARM devices are at my desk

The Next 23 Minutes:

1. Introduction into Yocto
2. Using Yocto to build images and SDKs
3. How to create KDE-power devices with Yocto

Embedded Devices in Scope of this Talk



source: https://www.e-consystems.com

custom hardware: purpose-tailored hardware, e.g. ARM CPU, CAN interface, 100BaseT1 Ethernet, NAND memory, GPIOs for different purposes

hardware evolves: hardware and software are often developed at the same time and only a limited number of prototype devices is available

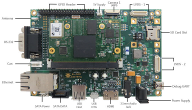
custom flashing process: devices must be flashed by developers, updated by technicians on field and provisioned at end of the manufacturing line

cross-building: hardware architecture different to x86 development machine

operating system: in this talk we only look at Linux (note: no hard real-time)

Part 1: Yocto Basics

The Basic Issue: Creating a System Image



Issue Statement

For a given target device, we want to generate a system image:

- image is a root file-system that is ready to be flashed to persistent memory
- image contains applications and libraries that work on target device's architecture
- image generation is reproduceable and well-definend (= no magic happens)
- image shall be created on an x86 developer machine

In the remainder of this talk, I will outline how Yocto attempts to solve this task.

Part 1: Yocto Basics

About the Yocto Project

> *The Yocto Project is an open source collaboration project that provides templates, tools and methods to help you create custom Linux-based systems for embedded and IOT products, regardless of the hardware architecture.*
>
> https://www.yoctoproject.org/about/

This means that Yocto. . .

1. strives for being an ecosystem that makes device creation simple
2. aims to provide all tools needed for doing this job
3. ensures reusability and vendor independence by defining general rules

Basic Notions:

Recipe: build and packaging instructions for compiling a source code package

Layer: set of recipes and/or modifications of other recipes

Image: complete root file system that is flashed onto a device

SDK: set of cross-compiled libraries, header files and all cross-compilation tools needed to cross-compile code for a target device

About the Yocto Project

> *The Yocto Project is an open source collaboration project that provides templates, tools and methods to help you create custom Linux-based systems for embedded and IOT products, regardless of the hardware architecture.*
>
> https://www.yoctoproject.org/about/

This means that Yocto...

1. strives for being an ecosystem that makes device creation simple
2. aims to provide all tools needed for doing this job
3. ensures reusability and vendor independence by defining general rules

Basic Notions:

Recipe: build and packaging instructions for compiling a source code package

Layer: set of recipes and/or modifications of other recipes

Image: complete root file system that is flashed onto a device

SDK: set of cross-compiled libraries, header files and all cross-compilation tools needed to cross-compile code for a target device

Part 1: Yocto Basics

Building Block 1: OpenEmbedded-Core

Yocto & OpenEmbedded

- OpenEmbedded is a build automation framework and cross-compile environment
- OpenEmbedded community was formally established in 2003; Yocto in 2010
- Yocto uses and co-maintains OpenEmbedded tools (BitBake, OE-Core)

OpenEmbedded-Core = metadata & build instructions

- defines how basic tasks are performed (reused in recipes)
  1. download source code
  2. configure source code
  3. setup build dependencies
  4. compile source code with the respective build system (CMake, QMake, Make. . .)
  5. populate cross-building sysroot and create packages
  6. perform QA checks
- provides a big initial set of recipes for core libraries and applications
- supported platforms: ARM, MIPS, PowerPC, x86, QEMU
- repository: `http://git.openembedded.org/openembedded-core/`

Part 1: Yocto Basics

Building Block 1: OpenEmbedded-Core

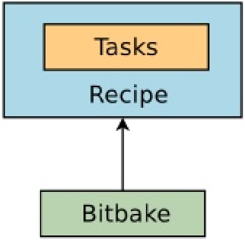
Yocto & OpenEmbedded

- OpenEmbedded is a build automation framework and cross-compile environment
- OpenEmbedded community was formally established in 2003; Yocto in 2010
- Yocto uses and co-maintains OpenEmbedded tools (BitBake, OE-Core)

OpenEmbedded-Core = metadata & build instructions

- defines how basic tasks are performed (reused in recipes)
  1. download source code
  2. configure source code
  3. setup build dependencies
  4. compile source code with the respective build system (CMake, QMake, Make...)
  5. populate cross-building sysroot and create packages
  6. perform QA checks
- provides a big initial set of recipes for core libraries and applications
- supported platforms: ARM, MIPS, PowerPC, x86, QEMU
- repository: `http://git.openembedded.org/openembedded-core/`

Building Block 2: BitBake



BitBake = build system

- BitBake recipes specify how a particular package is built (by using basic OE-Core tasks)
- overall BitBake execution:
  1. parses all available layers and their recipes
  2. prepares build tools and cross-build tools by configuring their environments
  3. schedules all basic tasks of all to be built packages by their defined dependencies (e.g. first download, then configure, then build; e.g. first build `qtbase` then `karchive`)
  4. performs the specified basic tasks
- BitBake configuration consists of two parts:
  1. environment configuration that is sourced by a script
  2. a folder `conf/` in the build directory that contains list of all layers (`bblayers.conf`) and build machine specific general configuration (`local.conf`)
- repository: `http://git.openembedded.org/bitbake/`

Part 1: Yocto Basics

Building Block 3: Poky

Poky = reference & quick-start distro

- reference distribution of the Yocto Project
- contains BitBake: build system, task scheduler and executor
- contains metadata (global definitions, build logic, packaging, etc.):
  1. OpenEmbedded-Core (OE-Core)
  2. Yocto Project-specific metadata (meta-yocto)
  3. Yocto Project-specific board support package (meta-yocto-bsp)
- Repository: `https://git.yoctoproject.org/cgit/cgit.cgi/poky/`

Poky contains everything to start a new project or to be used as a blueprint.

Part 1: Yocto Basics

SDKs for your Yocto Image

The Need for SDKs

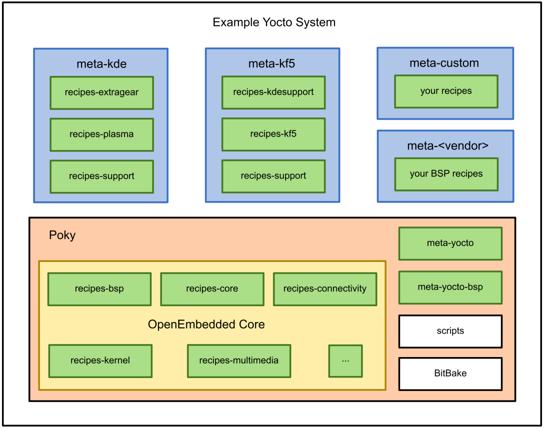
- building Yocto is complex and extremly time & space consuming
- split responsibilities in a team: development vs. integration
- in industry, you might not want to disclose your source codes or build configuration to contractors

Yocto's Standard SDK

- allows compiling for target device without building the Yocto system first
- can be generated alongside with image (and ALWAYS must be ABI compatible with image)
- is a self-extracing (gigantic) shell script
- contains all needed cross-building tools and cross-built libraries
- can be integrated with your IDE (e.g. KDevelop, QtCreator)
- simply generate with: `bitbake -c do_populate_sdk`

Part 2: Using Yocto

Contents of a Typical Yocto System



- always needed: OE-Core, BitBake
- distribution: Poky or some custom distribution
- board support package (e.g. meta-ti, meta-fsl-arm)
- custom layers with your own recipes (e.g. meta-kf5)

Part 2: Using Yocto

KF5-powered Devices with Yocto: meta-kf5 & meta-kde

Additional layers can bring additional libraries/applications to the Yocto world:

meta-kf5

- provides build recipes for latest release of KF5
- provides all non-standard dependency recipes
- can easily be integrated into any (recent) Yocto project
- Repository: `https://cgit.kde.org/yocto-meta-kf5.git/`
  → thanks to Johan Thelin & Volker Krause

meta-kde

- all recipes for building a full Plasma Desktop
- Repository: `https://cgit.kde.org/yocto-meta-kde.git/`
  → again, thanks to Volker!

And now?!

How to start with Yocto and KF5?

This talk has (by far) not enough time to do that, but I would start as follows:

1. start with the Yocto quick start guide, setup your system and try with QEMU: `https://www.yoctoproject.org/docs/2.5/brief-yoctoprojectqs/brief-yoctoprojectqs.html`
2. get a real development device (e.g. Raspberry, BeagleBone, i.MX6) and run your test system there
   → you will need a BSP layer for that...
3. integrate `meta-qt5` and run a simple full-screen QML test application (or a console application, if you do not have a display)
4. integrate `meta-kf5` and (if you want) `meta-kde`

Further Reading & References

- Yocto Project Documentation
  https://www.yoctoproject.org/docs/
- BitBake User Manual
  https://www.yoctoproject.org/docs/2.5/bitbake-user-manual/bitbake-user-manual.html
- Yocto Reference Manual
  https://www.yoctoproject.org/docs/2.5/ref-manual/ref-manual.html
- OpenEmbedded Wiki
  http://www.openembedded.org/wiki/Main_Page
- Qt for Embedded
  http://doc.qt.io/qt-5/embedded-linux.html

Thank you for your attention!

Andreas Cord-Landwehr

E-mail: cordlandwehr@kde.org