bbv
MAKING VISIONS WORK.

Yocto: Das geht auch Automatisch!

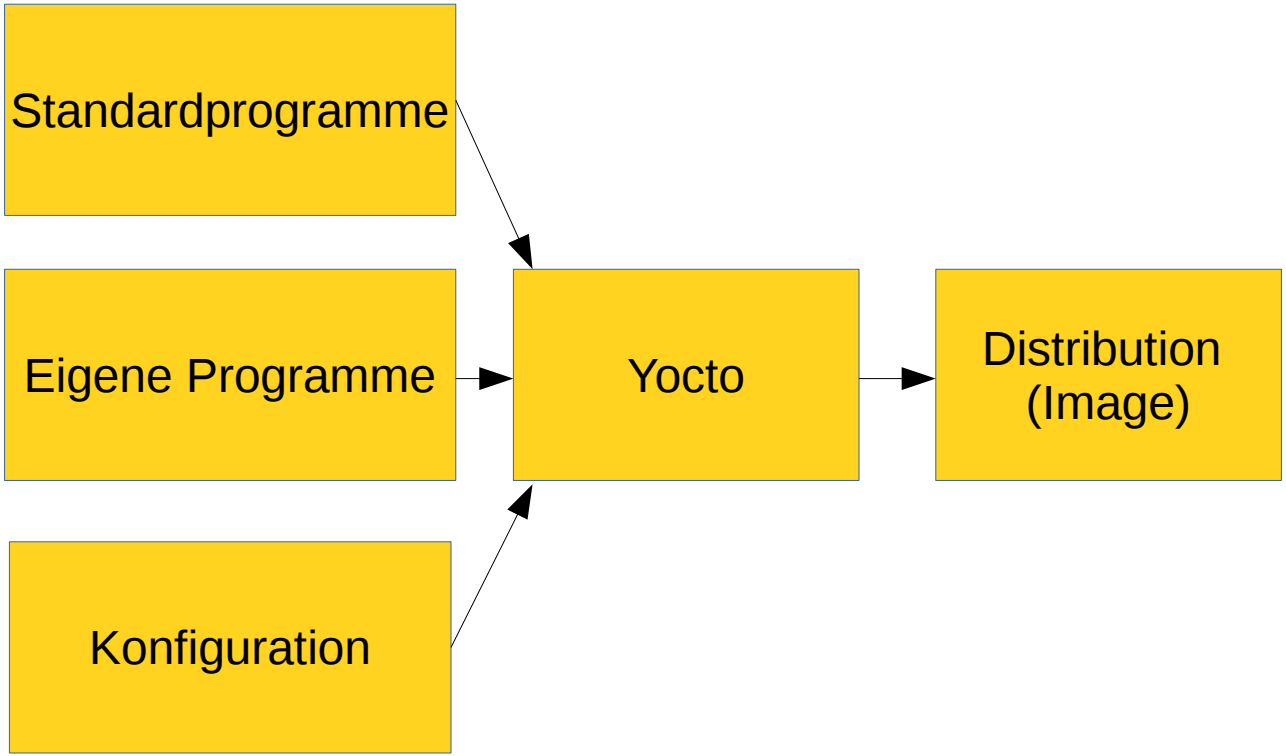
Yocto

It's not an embedded Linux distribution

– it creates a custom one for you

Yocto Project

Yocto Build


Standardprogramme
Eigene Programme
Konfiguration
Yocto
Distribution
(Image)

Generierung einer Distribution

Generierung einer Raspberry Pi Distribution

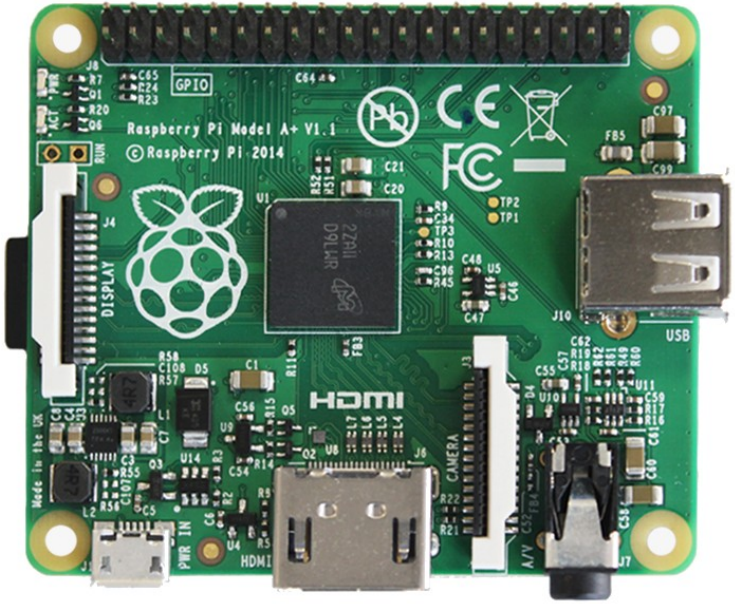
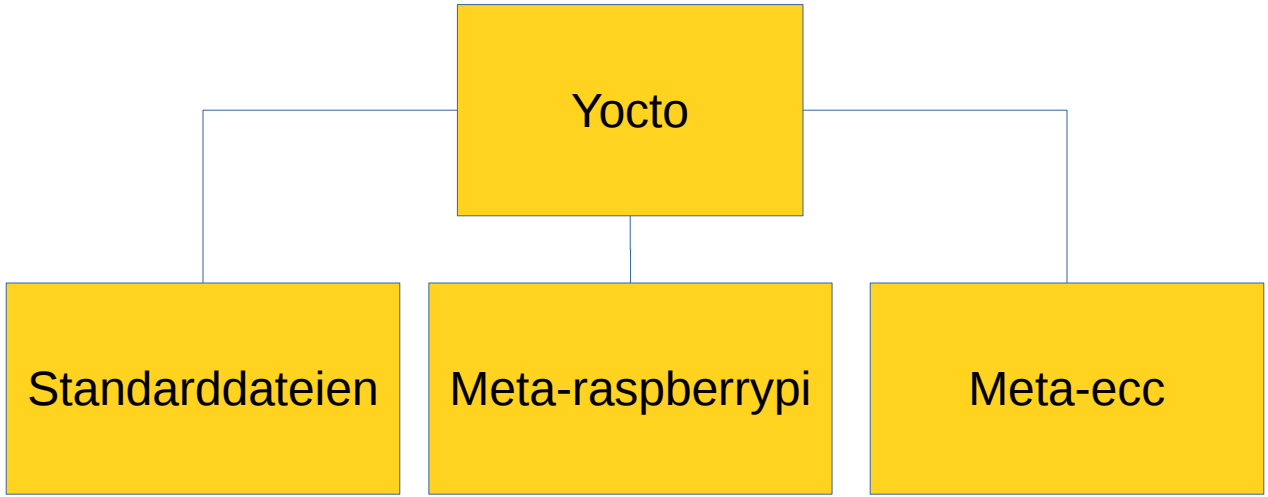


Photo: Florian Frankenberger

Benötigte Schritte zur Generierung

- Yocto herunterladen
- Meta-raspberrypi Layer herunterladen
- Eigenen Layer erstellen
  - Programm erstellen / patchen
- Layer in Yocto einbinden
- Maschine auswählen
- Image Generierung

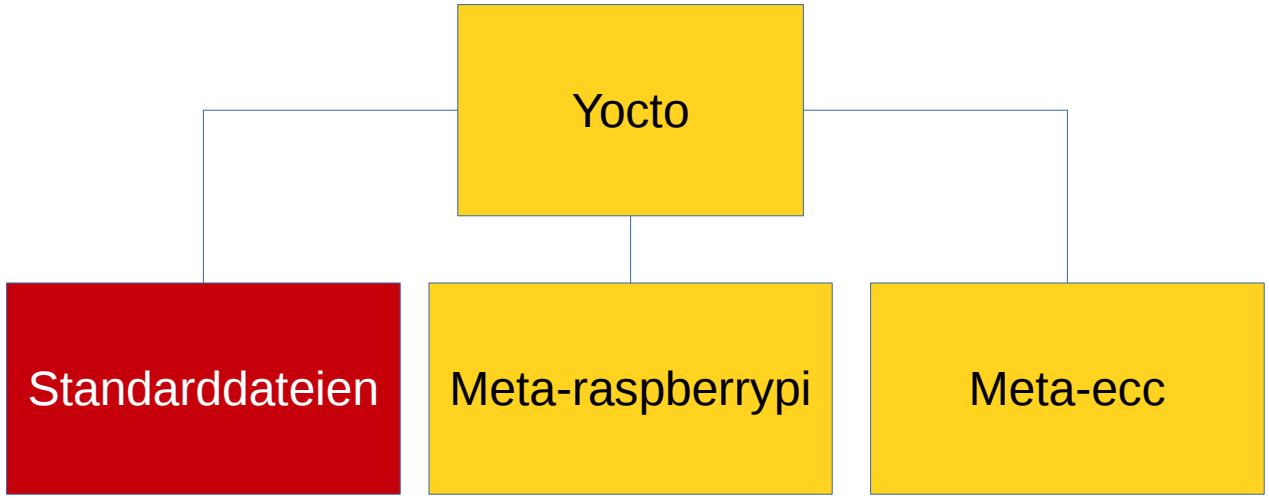
Yocto und seine Layer



Bitbake

- Arbeitstool um Code effizient und parallel abzuarbeiten
- Ähnlich wie “make”, aber weiterführend
  - Arbeitet mit Metadaten
  - Kann Dateien vom Internet herunterladen
  - Abhängigkeiten herausfinden und abarbeiten

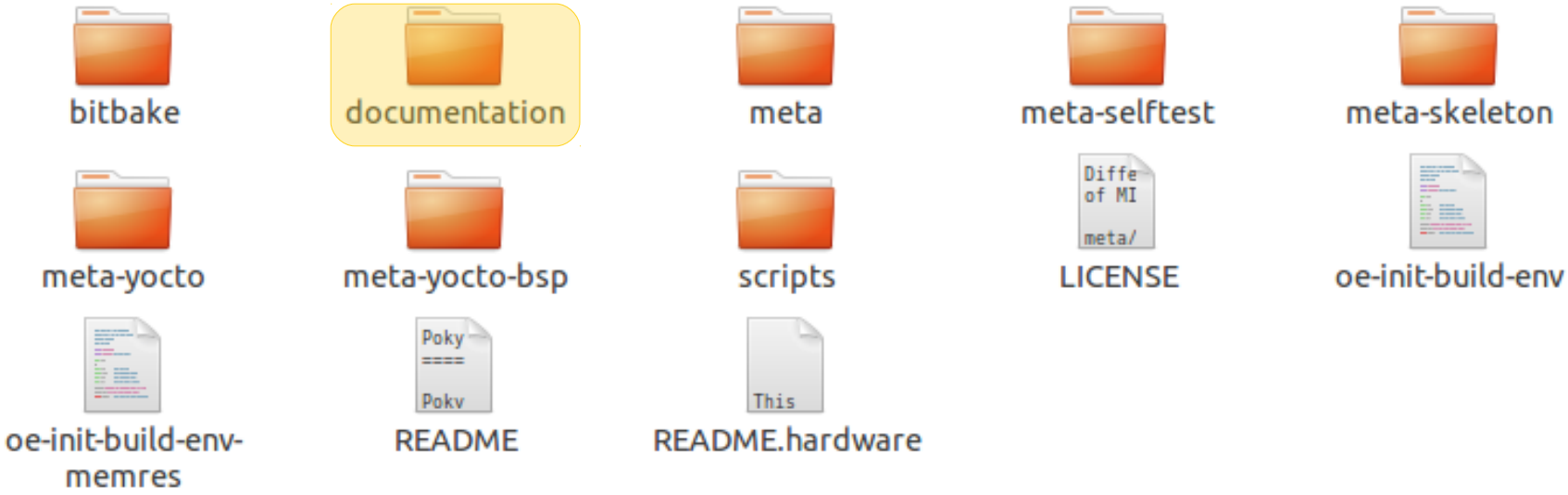
Yocto und seine Layer - Standarddateien



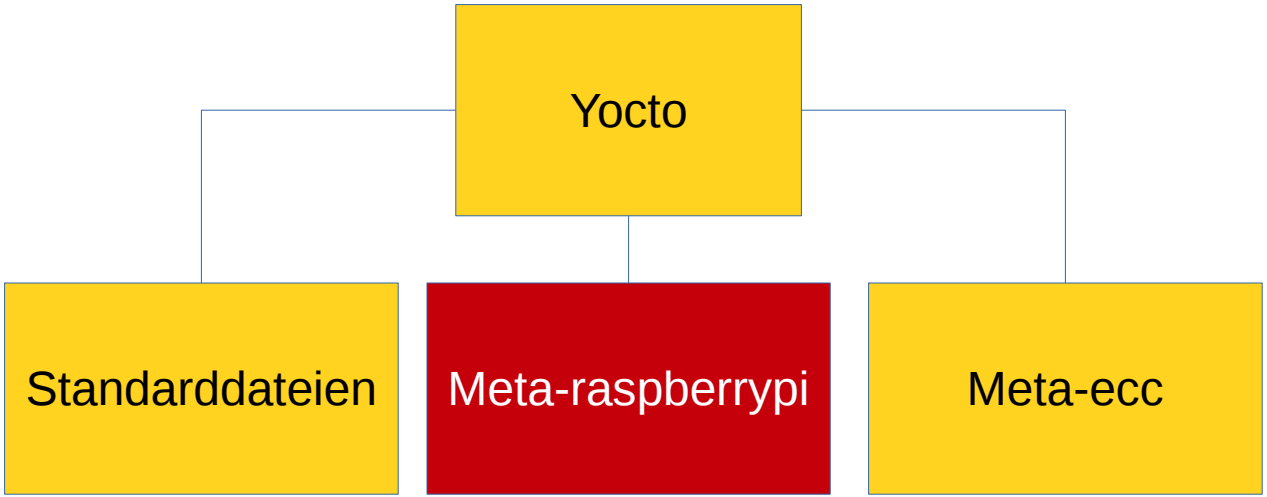
Poky

- Platformunabhängig, „cross-compiling“ Integrations Layer
- Basisrezepte
- Tools
- Metadaten

Poky Layout

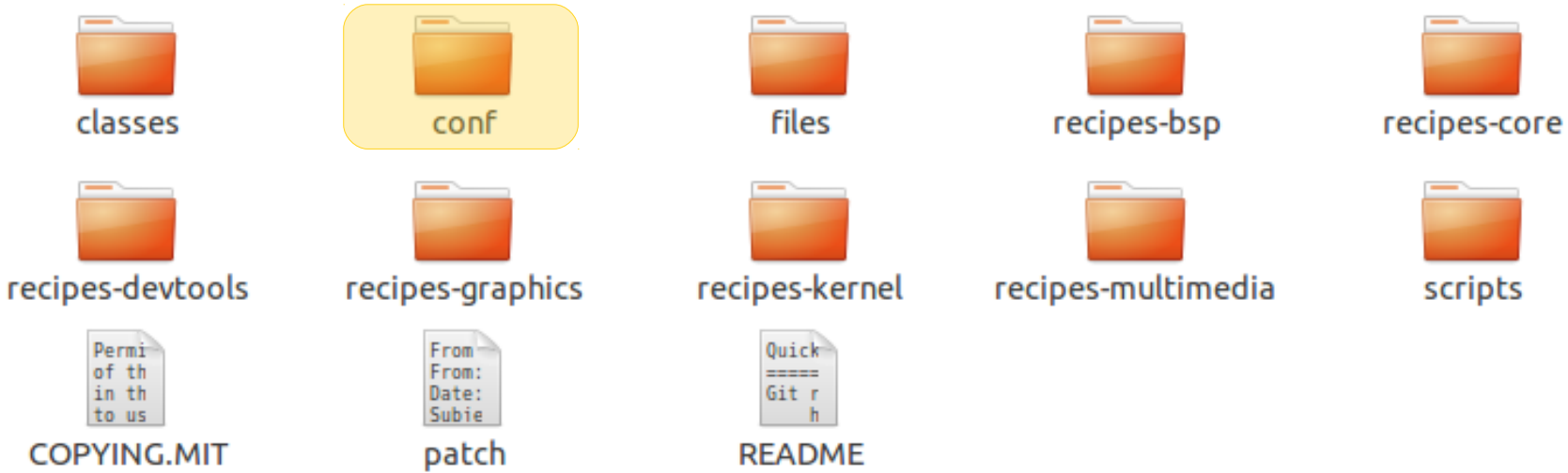

bitbake
documentation
meta
meta-selftest
meta-skeleton
meta-yocto
meta-yocto-bsp
scripts
Diffe
of MI
meta/
LICENSE
oe-init-build-env
oe-init-build-env-memres
Poky
====
Poky
README
This
README.hardware

Yocto und seine Layer – Hardware-Layer



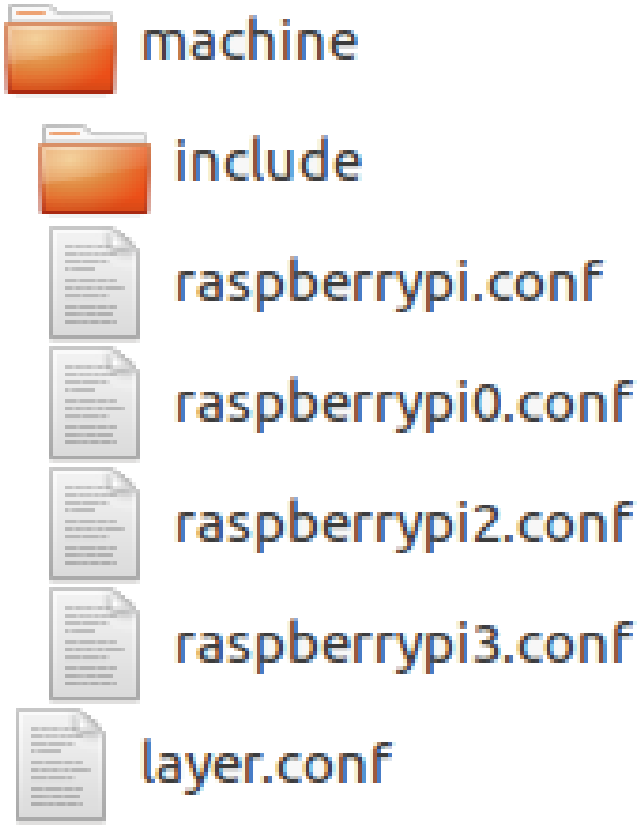
Meta-Raspberrypi

- BSP-Layer für Raspberry Pi
- Metadaten, angepasste Rezepte und Images

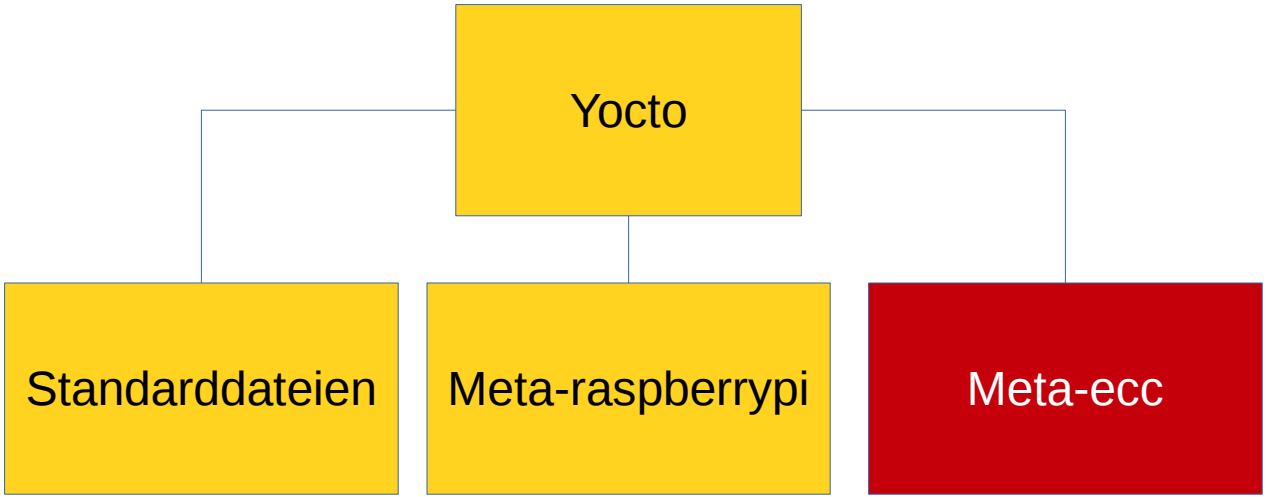


Meta-Raspberrypi

- BSP-Layer für Raspberry Pi
- Metadaten, angepasste Rezepte und Images



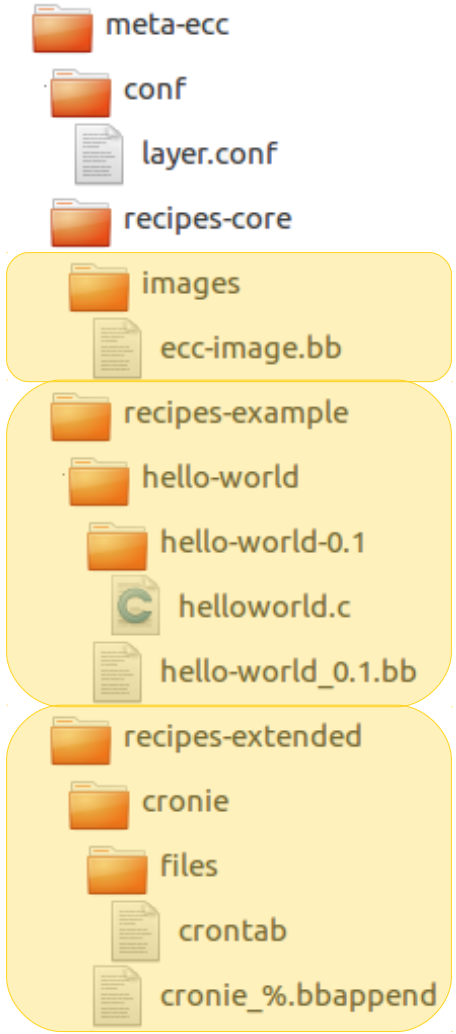
Yocto und seine Layer – Eigener Layer



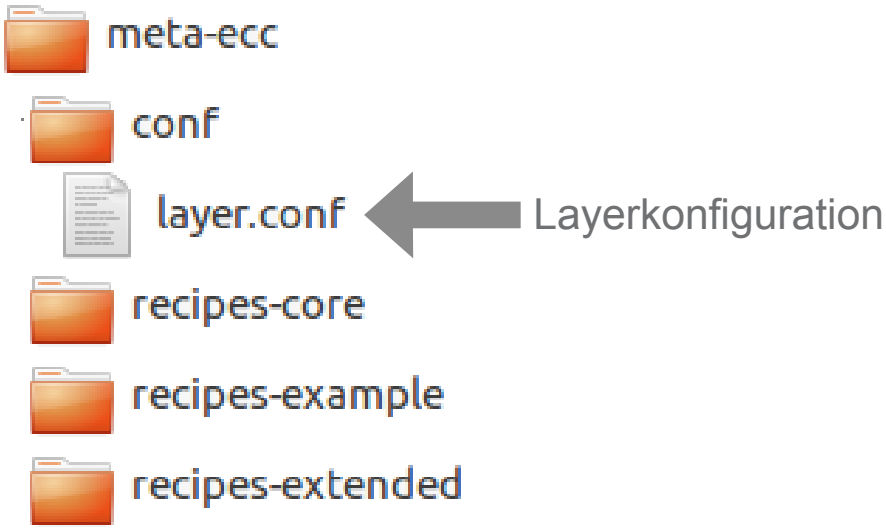
Meta-ecc

- Selbst erstellter Beispiel-Layer
  - Ein eigenes Rezept
  - Ein gepatchtes Rezept (cronie / crontab)
  - Beispiel-Image

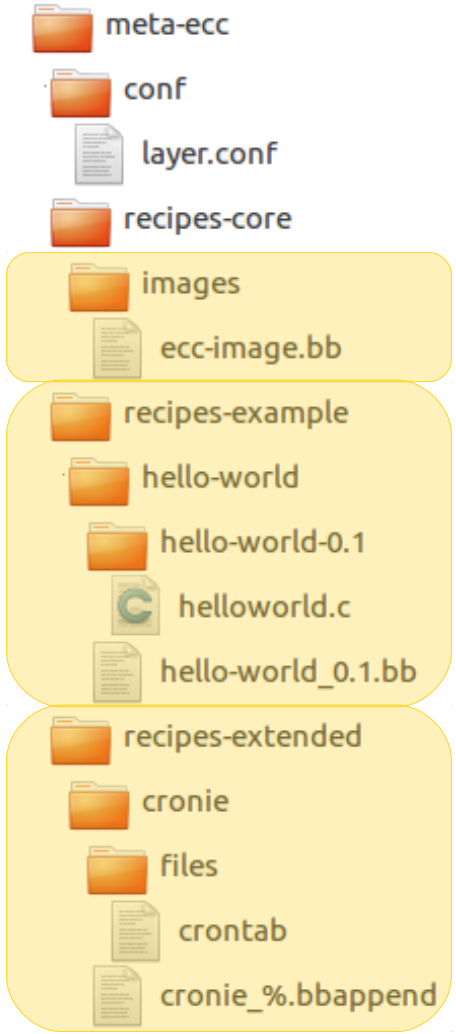
Meta-ecc


meta-ecc
conf
layer.conf
recipes-core
images
ecc-image.bb
recipes-example
hello-world
hello-world-0.1
helloworld.c
hello-world_0.1.bb
recipes-extended
cronie
files
crontab
cronie_%.bbappend

Meta-ecc - Konfiguration



Meta-ecc


meta-ecc
conf
layer.conf
recipes-core
images
ecc-image.bb
recipes-example
hello-world
hello-world-0.1
helloworld.c
hello-world_0.1.bb
recipes-extended
cronie
files
crontab
cronie_%.bbappend

Meta-ecc - Konfiguration

- Meta-ecc/conf/layer.conf

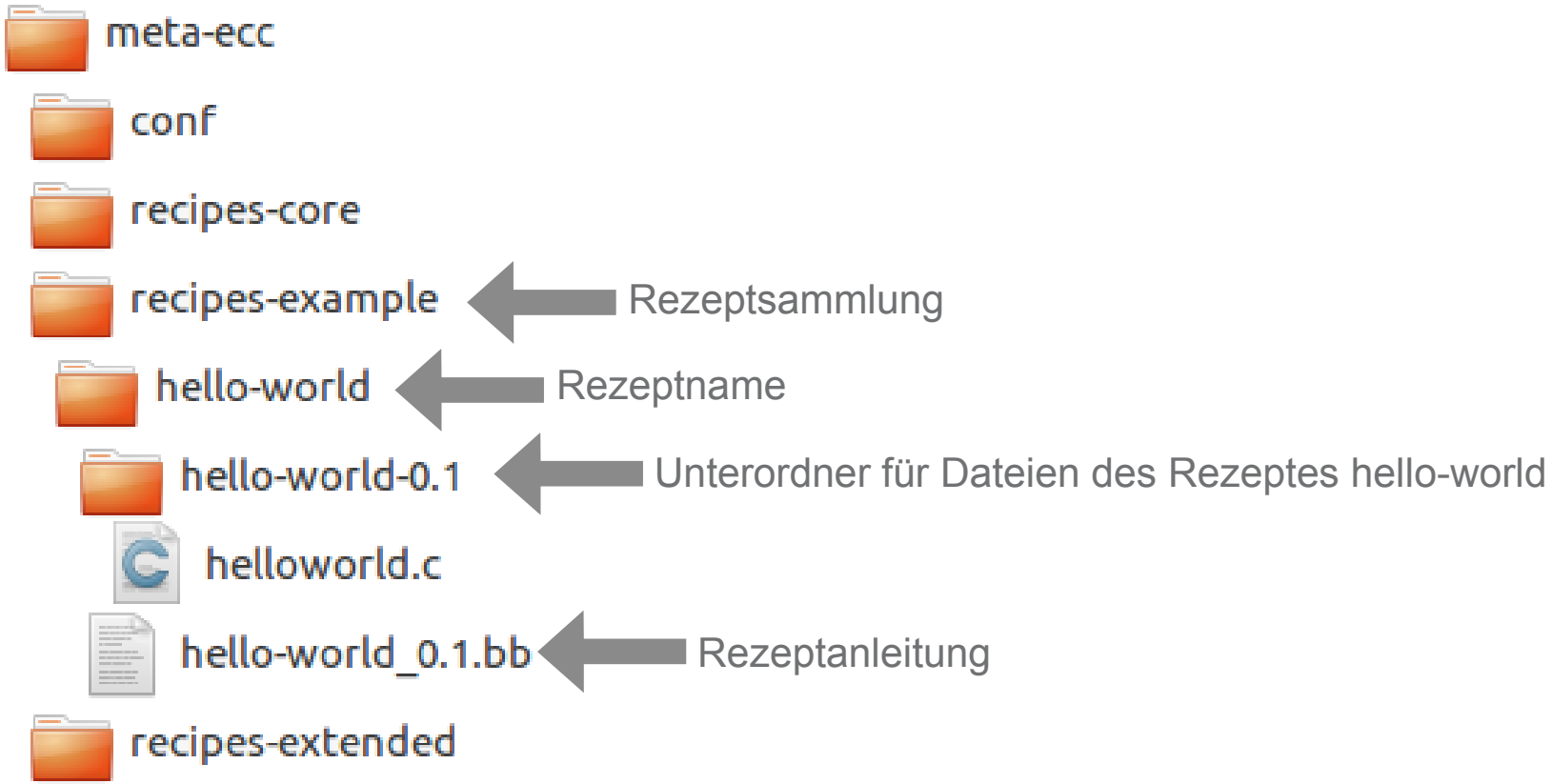
```
# We have a conf and classes directory, add to BBPATH
BBPATH .= ":${LAYERDIR}"

# We have recipes-* directories, add to BBFILES
BBFILES += "${LAYERDIR}/recipes-*/*/*.bb \
        ${LAYERDIR}/recipes-*/*/*.bbappend"

BBFILE_COLLECTIONS += "ecc"
BBFILE_PATTERN_ecc = "^${LAYERDIR}/"
BBFILE_PRIORITY_ecc = "6"
```

Layer Priorität

- ***bitbake-layers show-layers***

```
layer                 path                                          priority
==========================================================================
meta-raspberrypi      /media/eglis/unencrypted/yocto/poky/meta-raspberrypi  9
meta-ecc              /media/eglis/unencrypted/yocto/poky/meta-ecc          6
meta                  /media/eglis/unencrypted/yocto/poky/meta              5
meta-poky             /media/eglis/unencrypted/yocto/poky/meta-poky         5
meta-yocto-bsp        /media/eglis/unencrypted/yocto/poky/meta-yocto-bsp    5
```

Meta-ecc – hello-world



Rezepte

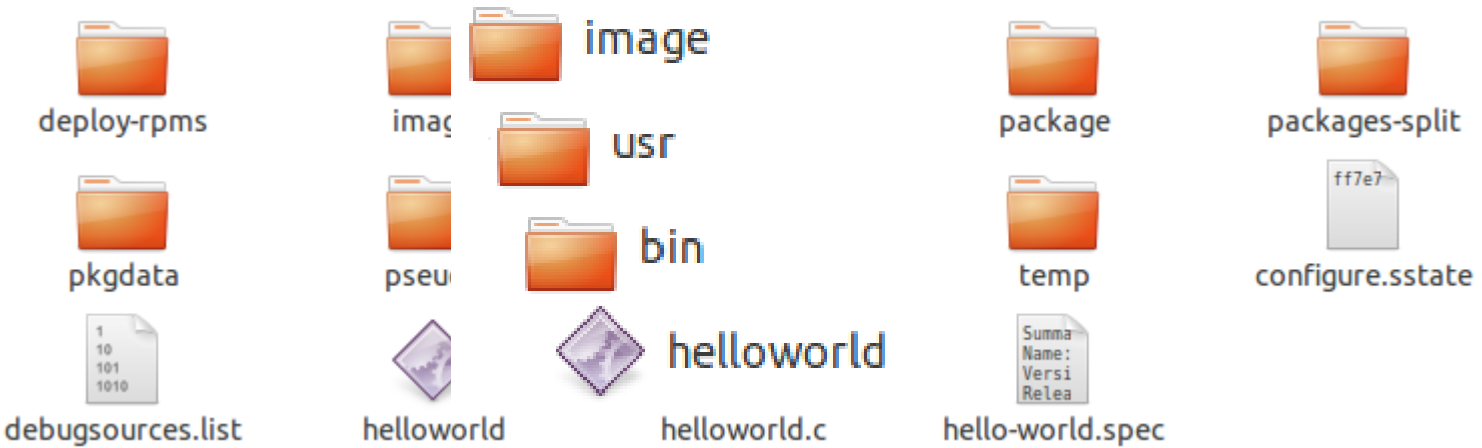
- Logische Einheit zu einem Softwarepaket
- Enthalten Metadaten über die zu bildende Programmeinheit
  - Was, wie, wo, wohin, ...
- Versionsnummern
- Abhängigkeiten
- Lizenz
- Anweisungen (Tasks)

Ein eigenes Rezept

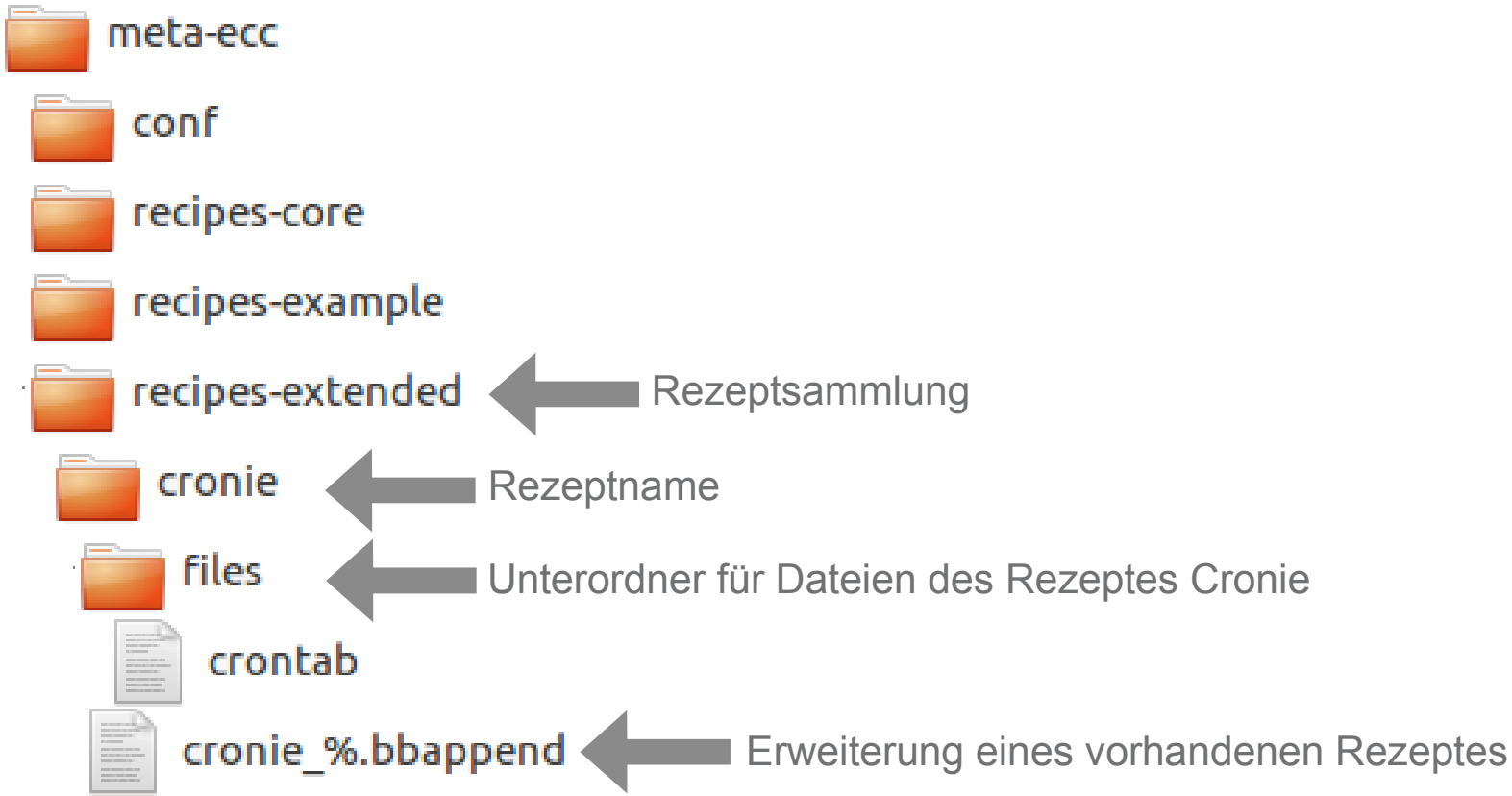
```
SUMMARY = "Simple helloworld application"
SECTION = "examples"
LICENSE = "MIT"
LIC_FILES_CHKSUM = "file://${COMMON_LICENSE_DIR}/MIT;md5=0835ade698e0bcf8506ecda2f7b4f302"

SRC_URI = "file://helloworld.c"

S = "${WORKDIR}"

do_compile() {
             ${CC} helloworld.c -o helloworld
}

do_install() {
             install -d ${D}${bindir}
             install -m 0755 helloworld ${D}${bindir}
}
```

Ein eigenes Rezept

- ***bitbake hello-world***
- Ergebniss unter …/build/tmp/work/cortexa7hf-vfp-vfpv4-neon-poky-linux-gnueabi/hello-world/0.1-r0/



Meta-ecc - cronie



Ein fremdes Rezept patchen

```
SUMMARY = "Cron daemon for executing programs at set times"
DESCRIPTION = "Cronie contains the standard UNIX daemon crond that runs \
specified programs at scheduled times and related tools. It is based on the \
original cron and has security and configuration enhancements like the \
ability to use pam and SELinux."
HOMEPAGE = "https://fedorahosted.org/cronie/"
BUGTRACKER = "https://bugzilla.redhat.com"

# Internet Systems Consortium License
LICENSE = "ISC & BSD-3-Clause & BSD-2-Clause & GPLv2+"
LIC_FILES_CHKSUM = "file://COPYING;md5=dd2a592178760e1386c769e1043b3722 \
                    file://src/cron.c;endline=20;md5=b425c3342650261771283 53a142633b4 \
                    file://src/popen.c;beginline=3;endline=31;md5=edd50742d8def712e9472dba353668a9"

SECTION = "utils"

SRC_URI = "https://fedorahosted.org/releases/c/r/cronie/cronie-${PV}.tar.gz \
           file://crond.init \
           file://crontab \
           file://crond.service \
           ${@bb.utils.contains('DISTRO_FEATURES', 'pam', '${PAM_SRC_URI}', '', d)}"

PAM_SRC_URI = "file://crond_pam_config.patch"
PAM_DEPS = "libpam libpam-runtime pam-plugin-access pam-plugin-loginuid"

SRC_URI[md5sum] = "9db75e1884d83a45e002d145c6c54d45"
SRC_URI[sha256sum] = "9cf8e3f4f5842a9c09413d02c8e0c855e12401f70b112465f0f81f2c84ebfb3f"

inherit autotools update-rc.d useradd systemd

PACKAGECONFIG ?= "${@bb.utils.contains('DISTRO_FEATURES', 'pam', 'pam', '', d)}"

PACKAGECONFIG[audit] = "--with-audit,--without-audit,audit,"
PACKAGECONFIG[pam] = "--with-pam,--without-pam,libpam,${PAM_DEPS}"

INITSCRIPT_NAME = "crond"
INITSCRIPT_PARAMS = "start 90 2 3 4 5 . stop 60 0 1 6 ."

USERADD_PACKAGES = "${PN}"
GROUPADD_PARAM_${PN} = "--system crontab"

SYSTEMD_SERVICE_${PN} = "crond.service"

do_install_append () {
        install -d ${D}${sysconfdir}/sysconfig/
        install -d ${D}${sysconfdir}/init.d/
        install -m 0644 ${S}/crond.sysconfig ${D}${sysconfdir}/sysconfig/crond
        install -m 0755 ${WORKDIR}/crond.init ${D}${sysconfdir}/init.d/crond

        # install systemd unit files
        install -d ${D}${systemd_unitdir}/system
        install -m 0644 ${WORKDIR}/crond.service ${D}${systemd_unitdir}/system
        sed -i -e 's,@BASE_BINDIR@,${base_bindir},g' \
               -e 's,@SBINDIR@,${sbindir},g' \
               ${D}${systemd_unitdir}/system/crond.service

        # below are necessary for a complete cron environment
        install -d ${D}${localstatedir}/spool/cron
        install -m 0755 ${WORKDIR}/crontab ${D}${sysconfdir}/
        mkdir -p ${D}${sysconfdir}/cron.d
        mkdir -p ${D}${sysconfdir}/cron.hourly
        mkdir -p ${D}${sysconfdir}/cron.daily
        mkdir -p ${D}${sysconfdir}/cron.weekly
        mkdir -p ${D}${sysconfdir}/cron.monthly
        touch ${D}${sysconfdir}/cron.deny

        # below setting is necessary to allow normal user using crontab

        # setgid for crontab binary
        chown root:crontab ${D}${bindir}/crontab
        chmod 2755 ${D}${bindir}/crontab

        # allow 'crontab' group write to /var/spool/cron
        chown root:crontab ${D}${localstatedir}/spool/cron
        chmod 770 ${D}${localstatedir}/spool/cron

        chmod 600 ${D}${sysconfdir}/crontab
}

FILES_${PN} += "${sysconfdir}/cron*"
CONFFILES_${PN} += "${sysconfdir}/crontab"
```

Ein fremdes Rezept patchen

- Inhalt des cronie_%.bbappend Files sehr reduziert

```
FILESEXTRAPATHS_prepend := "${THISDIR}/files:"

SRC_URI += "file://crontab"
```

Layer eintragen

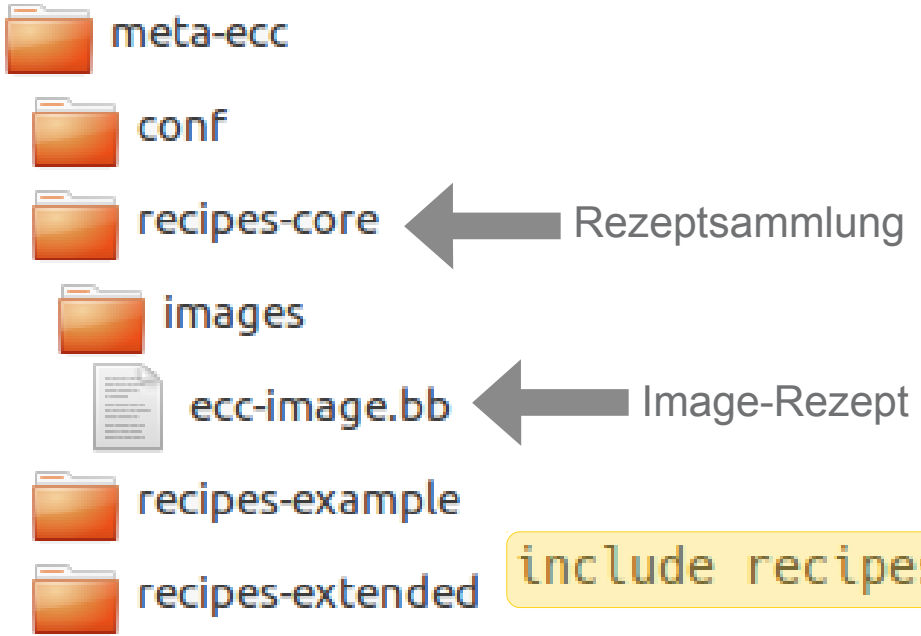
- Bekannt machen der verwendeten Layer
- build/conf/bblayers.conf

```
BBLAYERS ?= " \
  /media/eglis/unencrypted/yocto/poky/meta-raspberrypi \
  /media/eglis/unencrypted/yocto/poky/meta-ecc \
  /media/eglis/unencrypted/yocto/poky/meta \
  /media/eglis/unencrypted/yocto/poky/meta-poky \
  /media/eglis/unencrypted/yocto/poky/meta-yocto-bsp \
  "
```

Machine auswählen

- Platformauswahl
- Machine oft vom Hersteller bereitgestellt
- Im Build-Verzeichnis unter build/conf/local.conf
  - MACHINE = "raspberrypi2"

Meta-ecc - Image




```
include recipes-core/images/core-image-base.bb
```

```
IMAGE_INSTALL += "hello-world cronie"
```

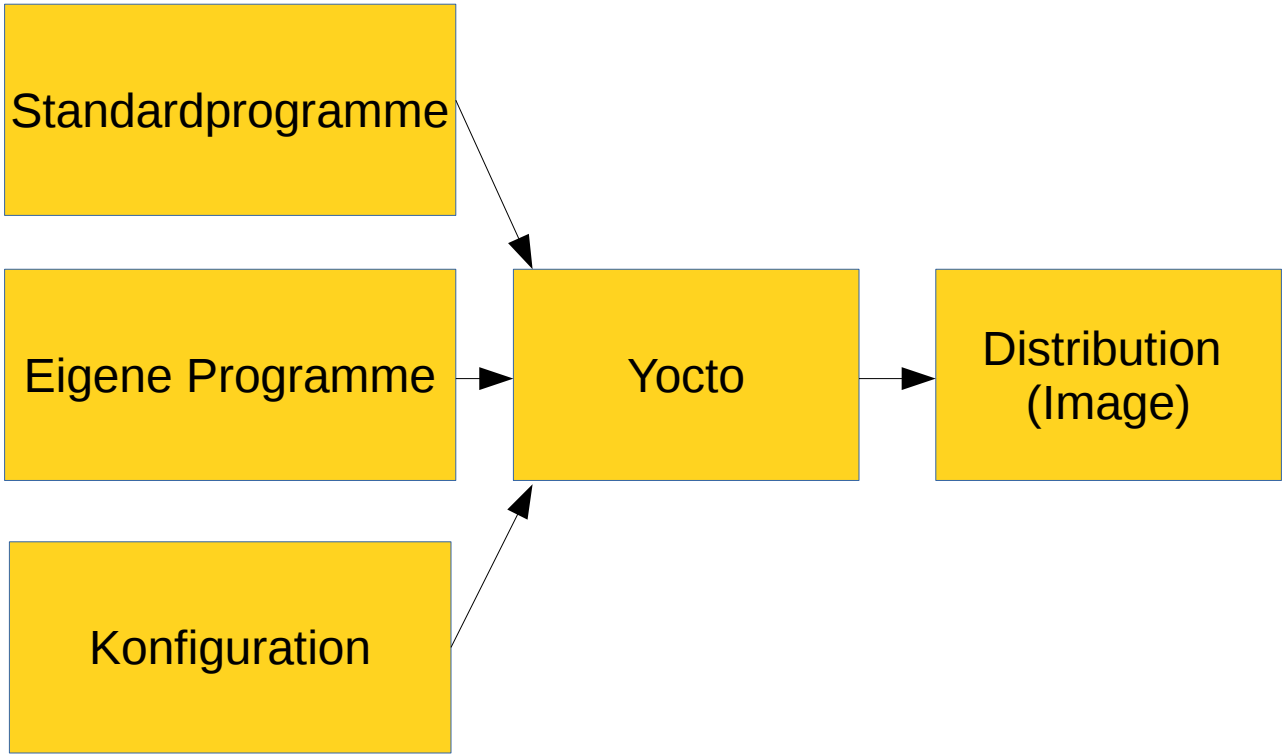
Image Deployment

- IMAGE_FSTYPES bestimmt Art des Output
  - tar, ext3, sdimg, wic, elf, vmdk, ...
- Beispielsweise kopieren des Images auf SD-Karte
- Viele weitere Möglichkeiten des Verteilens

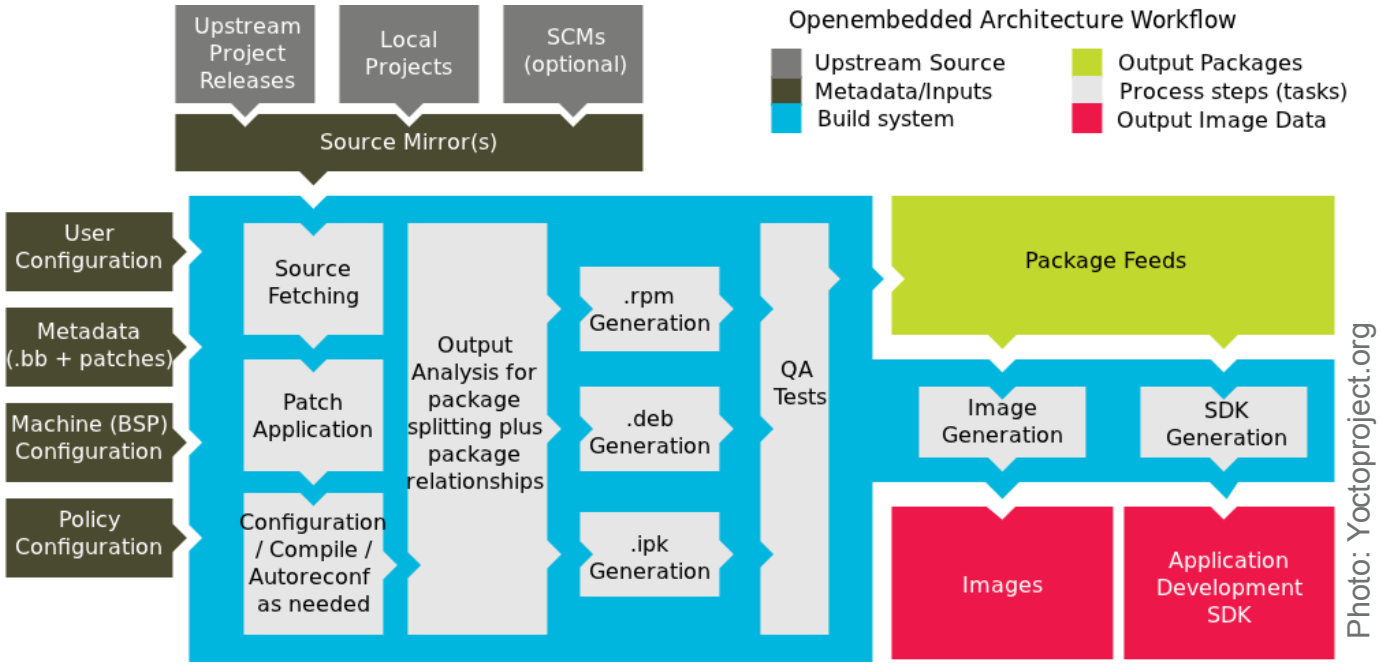
Image Generierung

- ***source poky/oe-init-build-env***
  - Initialisiert das Terminal
- ***bitbake ecc-image***
  - Buildet alle benötigten Rezepte
  - Buildet die selbst definierten Rezepte
  - Erstellt Partitionen / Images / Archive
- Dauert längere Zeit (ca. 2h für dieses Beispiel)
- Benötigt grosse Mengen an Speicherplatz (ca. 25GB in diesem Beispiel)

Yocto Übersicht einfach



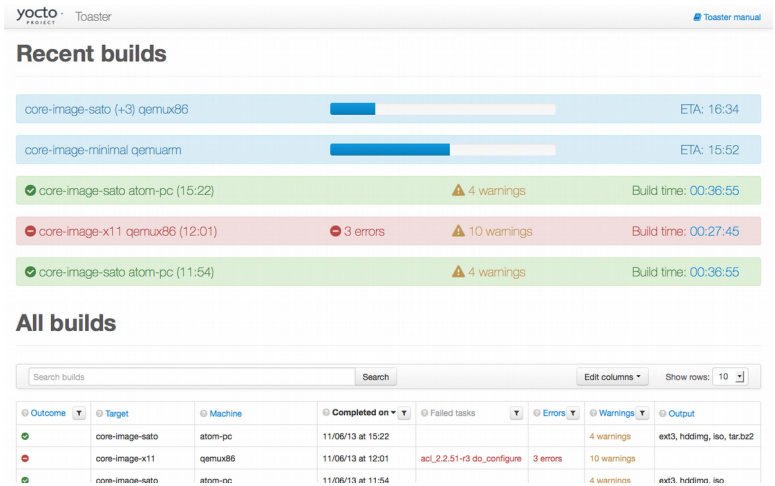
Yocto Übersicht erweitert

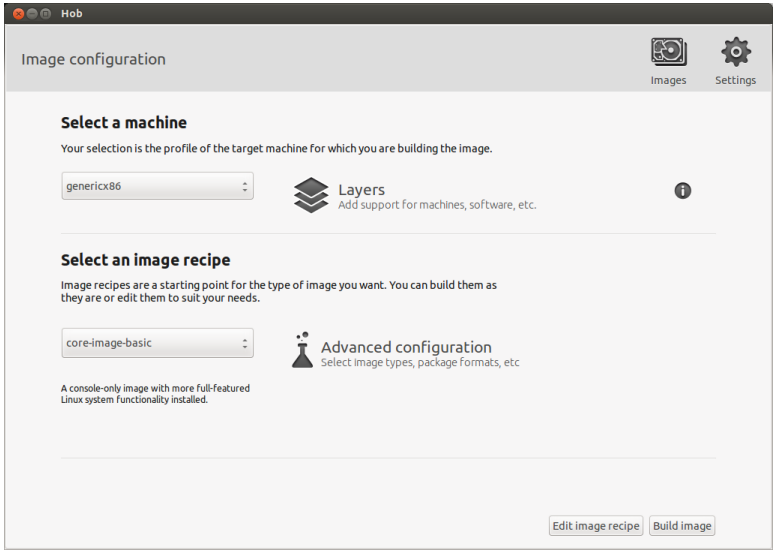


Und was jetzt?

Und was jetzt? Toaster / Hub

- (Web-)GUIs für den Build Prozess





Photos: Yoctoproject.org

Und was jetzt? Qemu Virtualisierung

- Test, verifizieren, Look and feel…
- Continous integration, continous development

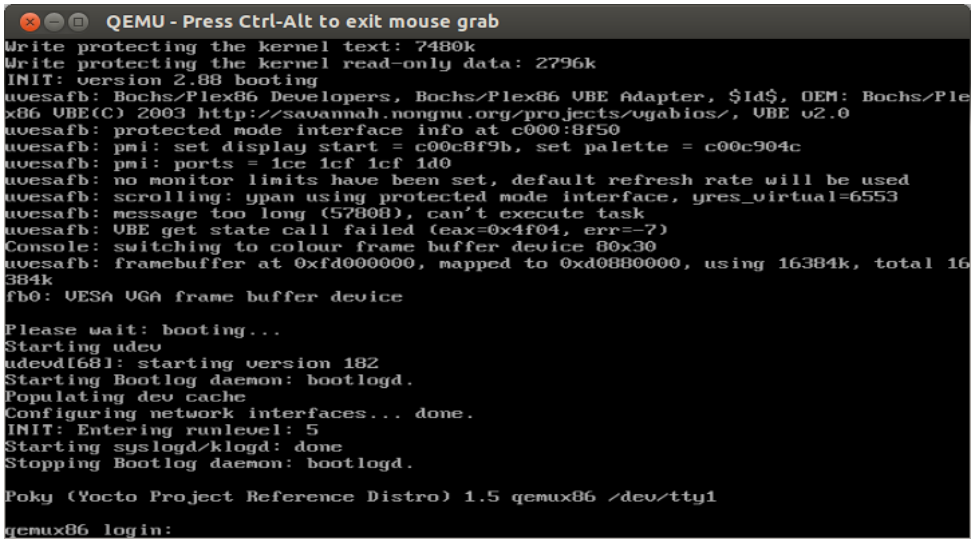


Photo: Yoctoproject.org

Und was jetzt? Mehrere Maschinen

- Eine Zeile ändern und für eine andere Machine builden
  - Mehrere Ausbaustandards unterstützen
  - Ressourcen sparen für schwächere Maschinen
  - Plugins nicht mitbauen
  - Etc...

Weiterführende Informationen

- http://www.yoctoproject.org/docs/current/dev-manual/dev-manual.html
- http://www.yoctoproject.org/docs/latest/bitbake-user-manual/bitbake-user-manual.html
- https://community.freescale.com/docs/DOC-94953

Links zur Präsentation

- git://git.yoctoproject.org/poky.git
- git://git.yoctoproject.org/meta-raspberrypi
- https://github.com/Deadolus/meta-ecc

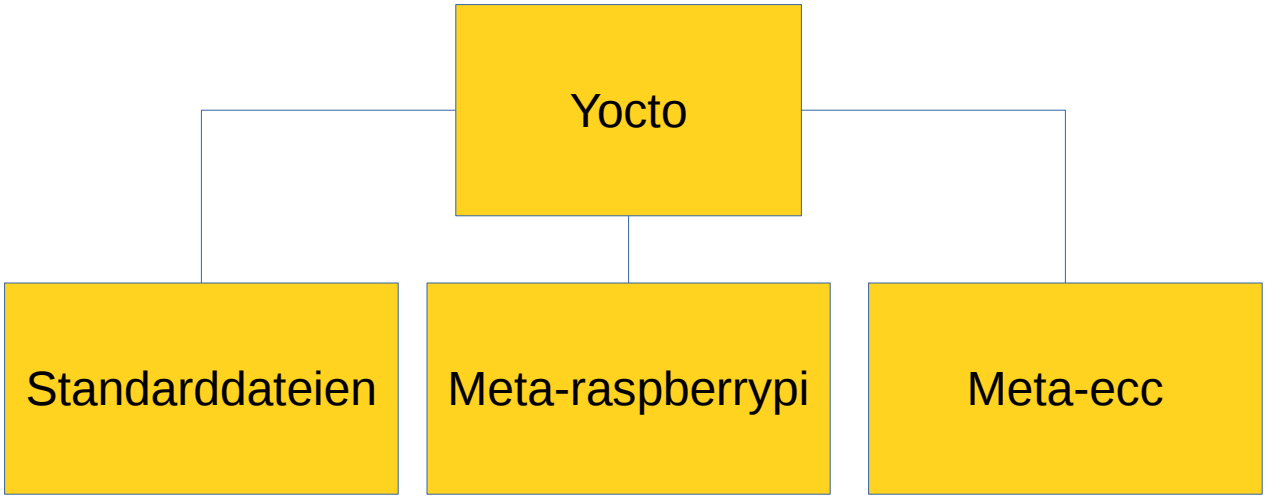
MAKING VISIONS WORK.

Simon Egli

bbv Software Services AG
Binzmühlestrasse 56
8050 Zürich
Schweiz

simon.egli@bbv.ch
Telefon +41 44 315 63 63
www.bbv.ch

Generierung einer Raspberry Pi Distribution



Einige BSPs

- meta-amd
- meta-axxia
- meta-baryon
- meta-cloud-services
- meta-darwin
- meta-darwin-contrib
- meta-dlna
- meta-eca
- meta-fsl-arm
- meta-fsl-ppc
- meta-intel
- meta-intel-edison
- meta-intel-galileo
- meta-intel-iot-devkit
- meta-intel-iot-middleware
- meta-intel-quark
- meta-ivi
- meta-jarvis
- meta-java
- meta-lsi
- meta-luv
- meta-maker
- meta-mentor
- meta-ti
- meta-xilinx
- ...

Rezeptvariablen

- SUMMARY / DESCRIPTION
  - Zusammenfassung des Rezeptes / für den Paketmanager
- PN = Package Name
- PV = Packet Version
- LICENSE = Lizenz des Packets
- LIC_FILES_CHKSUM = Checksumme
  - Verschiedene Arten und Optionen zum Prüfen vorhanden
  - Über 150 Standard Lizenzen bereitgestellt

Rezeptvariablen (cont.)

- SRC_URI = Woher stammt der Quellcode, wie wird er heruntergeladen
  - Verschiedenste Protokolle unterstützt (file, git, ftp, ..)
- S = Ort des entpackten Quellcode
  - ${WORKDIR}/${PN}-${PV}
  - Wird oft überschrieben

Einige Bitbake Zuweisungen

- VAR = "foo" simple assignment
- VAR ?= "foo" assign if no other value is already assigned
- VAR ??=foo weak default assignment
- VAR = "stuff ${OTHER_VAR} more“
- VAR := "stuff ${OTHER_VAR} more"
- VAR += "foo" append with space
- VAR =+ "foo" prepend with space

Source: http://elinux.org/Bitbake_Cheat_Sheet

Einige Bitbake Zuweisungen (cont.)

- VAR .= "foo" append without space
- VAR =. "foo" prepend without space
- VAR_append = "foo" append without space

Source: http://elinux.org/Bitbake_Cheat_Sheet

Meta-ecc


meta-ecc
conf
layer.conf
recipes-core
images
ecc-image.bb
recipes-example
hello-world
hello-world-0.1
helloworld.c
hello-world_0.1.bb
recipes-extended
cronie
files
crontab
cronie_%.bbappend
Layerkonfiguration
Rezeptsammlung
Image-Rezept
Rezeptname
Unterordner für Dateien des Rezeptes hello-world
Rezeptanleitung
Rezeptname
Unterordner für Dateien des Rezeptes Cronie
Erweiterung eines vorhandenen Rezeptes

Ein fremdes Rezept patchen

- Linux Cron-Scheduler
  - Führt vordefinierte Befehle zu vorbestimmten Zeitpunkten aus
- Konfigurationsdatei unter ***/etc/crontab***
  - Formatierte Textdatei die überschrieben werden soll

Und was jetzt? Repo

- Bündelt verschiedene Git-Repositories
- Hilft beim Auschecken
- Versionierung der Repositories