ilbers
Technology for better life

Isar

Build Debian-Based Products with BitBake

Baurzhan Ismagulov

Embedded Linux Conference Europe
Oct 11 - 13, 2016
Berlin, Germany

Contents

- About Us
- Motivation
- Existing Software
- What is Isar
- How It Works
- Using Isar
- Some Other Approaches
- Next Steps
- Summary
- Questions

About Us

- Based in Munich, Germany and Saint Petersburg, Russia
- Provide software development services since 2010
- Areas of expertise:
  - Linux and open-source software
  - Application and driver development
  - Real-time and safety-critical software development
  - Networking and wireless
  - Virtualization
- Contribute to Linux kernel and other FOSS projects

Motivation

Product build system

- One-command, on-demand building
- Produce complete, ready-to-use firmware images
- Low effort: No massive changes to upstream packages
- Efficiency: Pre-built binary packages

Features

- Adjust upstream packages
- Build several products
- Share components
- Multiple vendors

Customer requirements

- Native compilation for ARM
- Security updates
- Maintenance: 10+ years
- Legal clearing

Prior Art: Debian

- Provides many packages (armhf: 17575 src, 35555 bin)
- Provides cross-compilers
- Pre-built binary packages, shorter image creation times
- Very rich tool ecosystem (dpkg, apt, debootstrap, buildd...)
- Conservative version selection: Mature, pre-tested results
- Elaborate license process: Simpler product license clearing
- Long-term maintenance
- Security updates
- Usage scales between individual products and product lines
- One-command, on-demand building of the whole project: Not OOTB
- Build host: Debian (any with debootstrap + chroot / VM)
- ARM: Pre-built → Optimized for chosen CPU variants, e.g.:
  - armel: ARMv4+, no FPU, Thumb (“lowest common denominator”)
  - armhf: ARMv7, VFP v3 w/16 regs, Thumb-2 (“Cortex”)

Prior Art: Yocto

- Provides core packages (1298 src)
- Provides cross-compilers
- One-command, on-demand building of the whole project
- Modular, fully customizable build process
- Collaboration process (core / vendors / company / product layers)
- Build optimized for the particular hardware platform
- Builds cross-compilers from scratch
- Builds the whole project from scratch
- Build host: “Any” (in practice, issues beyond tested platforms)

Isar: Debian + BitBake

Integration System for Automated Root filesystem generation

- Base system: Debian binary packages (not a part of Isar)
- Build system: BitBake, the rule-based build system behind Yocto
- Structure, layering, workflow: Yocto

Isar at a Glance

- Isar:
  - Installs Debian binary packages as a base system
  - Builds and installs product's software packages
  - Creates ready-to-use images
- Isar is:
  - A set of scripts (BitBake *recipes*) to do the above
  - Product template for your own products (a *layer*)
- Provides infrastructure for:
  - Customizations
  - Product variability
  - Efficient component sharing

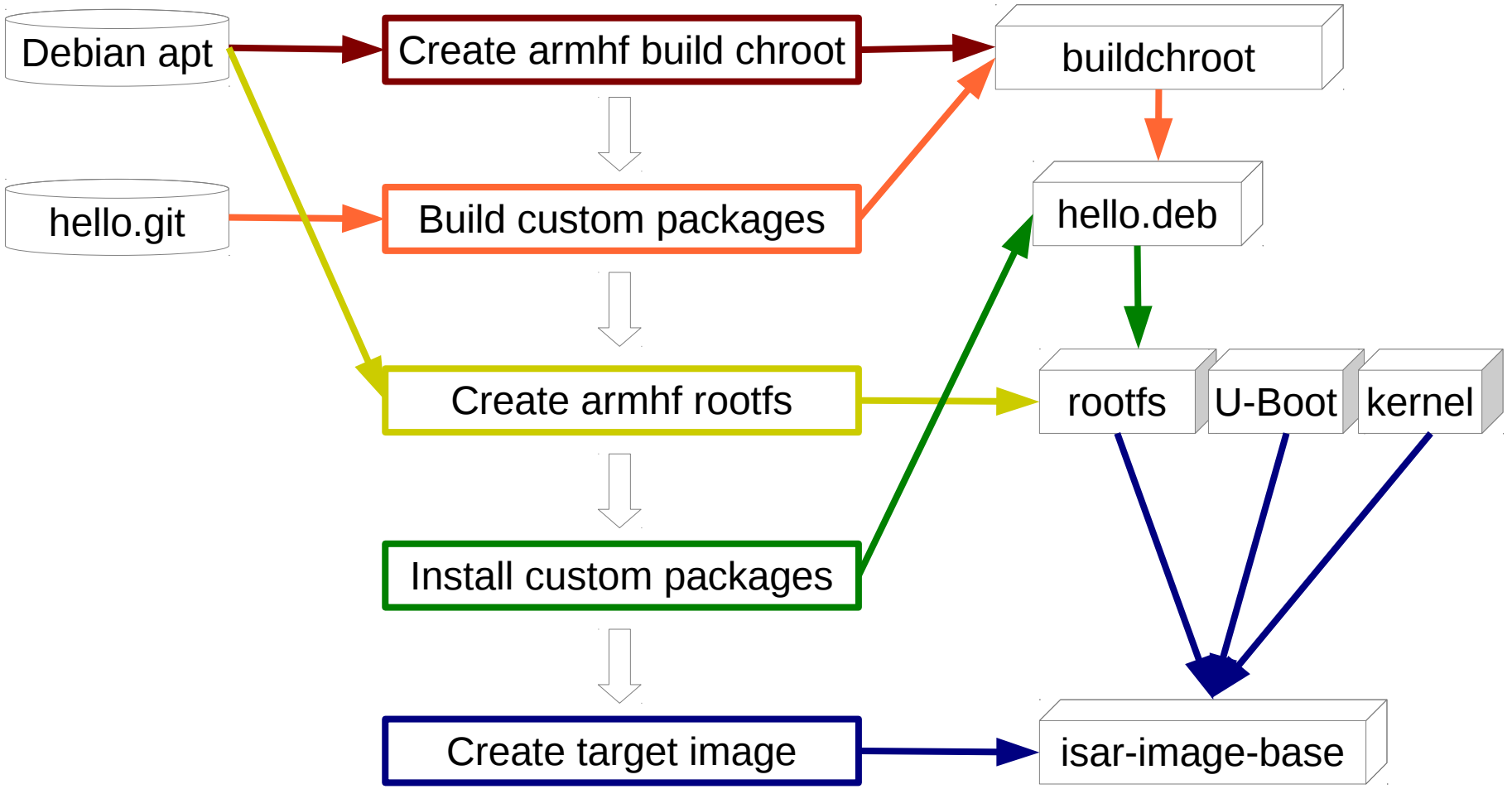
Areas of Application

- Possible uses:
  - Any Linux-based embedded devices
  - Component sharing across industries
- Benefits:
  - Multiple products, easy code reuse
  - Build automation
  - Build performance

Isar Development History

- 2004: SLIND (Siemens Linux Distribution) + build.sh
- 2011: SLIND + BitBake
- 2015: Debian + BitBake
- 2016: Started open-sourcing features

How Isar Works



- Native compilation with `dpkg-buildpackage` under QEMU armhf

BitBake Basics

- Isar: Everything is done in BitBake *recipes*
- Recipes:
  - Procedural rulesets for accomplishing specific work
  - Written in a shell-like BitBake language
  - Consist mostly of variable and *task* definitions
- Tasks:
  - Function-like code snippets
  - Implemented in shell or Python
  - May depend on other tasks
- Layers:
  - Directories grouping recipes according to e.g. their origin / ownership / function
  - Usually named `meta-*` (← "metadata")
  - Must be listed to be searched
  - Must have a layer config file

Isar Metadata Hierarchy

- `isar/`: Repo root
  - `bitbake/`: Recipe interpreter
  - `meta/`: Core layer
  - `meta-isar/`: Product template layer
  - `isar-init-build-env`: Build environment initialization script. Must be sourced in the current shell, not executed in a sub-shell.

Isar Core Recipes

- `meta/`: Core layer
  - `recipes-devtools/`: Development tool group (arbitrary)
    - `buildchroot/`: A recipe directory
      - `buildchroot.bb`: Recipe for creating an armhf build chroot on the host. Doesn't produce a binary package for the target.

```
BUILDCHROOT_PREINSTALL ?= "gcc make dpkg apt"
do_build() {
    sudo multistrap -a "${DISTRO_ARCH}" \
        -d "${BUILDCHROOT_DIR}" \
        -f "${WORKDIR}/multistrap.conf"
}
```

      - `files/`: Files belonging to the recipe

Isar Core Layer

- `meta/`: Core layer
  - `classes/`: Generic rules inherited by recipes to accomplish repeating tasks. Implemented in BitBake language.
    - `dpkg.bbclass`: Build binary `.deb` from `pkg.git`
    - `ext4-img.bbclass`: Create an ext4 image
    - `image.bbclass`: Create a filesystem image (uses pluggable `*-img.bbclass`)
  - `conf/`: Global configuration
    - `bitbake.conf.sample`: Global BitBake config (paths, etc.). Copied to the build directory by `isar-init-build-env`. `Include`s local configs to form a single global environment.
    - `layer.conf`: Layer config. Mandatory for every layer. Among other things, specifies where to look for recipes (`recipes-*/*/*.bb`).

Product Layer

- `meta-isar/`: Product template layer
  - `classes/`: Product-specific classes
    - `rpi-sdimg.bbclass`: Packs U-Boot, kernel, rootfs in an SD card image. Uses `ext4-img.bbclass`.
  - `conf/`: Layer configuration
    - `bblayers.conf.sample`: Global layer config. Copied to the build directory. Defines e.g. layers to use.

      ```
      BBLAYERS ?= "meta meta-isar"
      ```

    - `local.conf.sample`: Local build config. Copied to the build directory. Defines e.g. the default machine and number of tasks to start in parallel.

      ```
      MACHINE ??= "qemuarm"
      DISTRO ??= "debian-wheezy"
      IMAGE_INSTALL = "hello"
      BB_NUMBER_THREADS = "4"
      ```

Product Variants

- `meta-isar/`: Product template layer
  - `conf/`: Layer configuration
    - `distro/`: Distro configs (suite, arch, apt source, etc.)
      - `debian-wheezy.conf`
      - `raspbian-stable.conf`
    - `machine/`: Board configs (U-Boot, kernel, etc.)
      - `qemuarm.conf`
      - `rpi.conf`
    - `multiconfig`: Enables BitBake to create images for several different boards (*machines*) in one call

Product Recipes

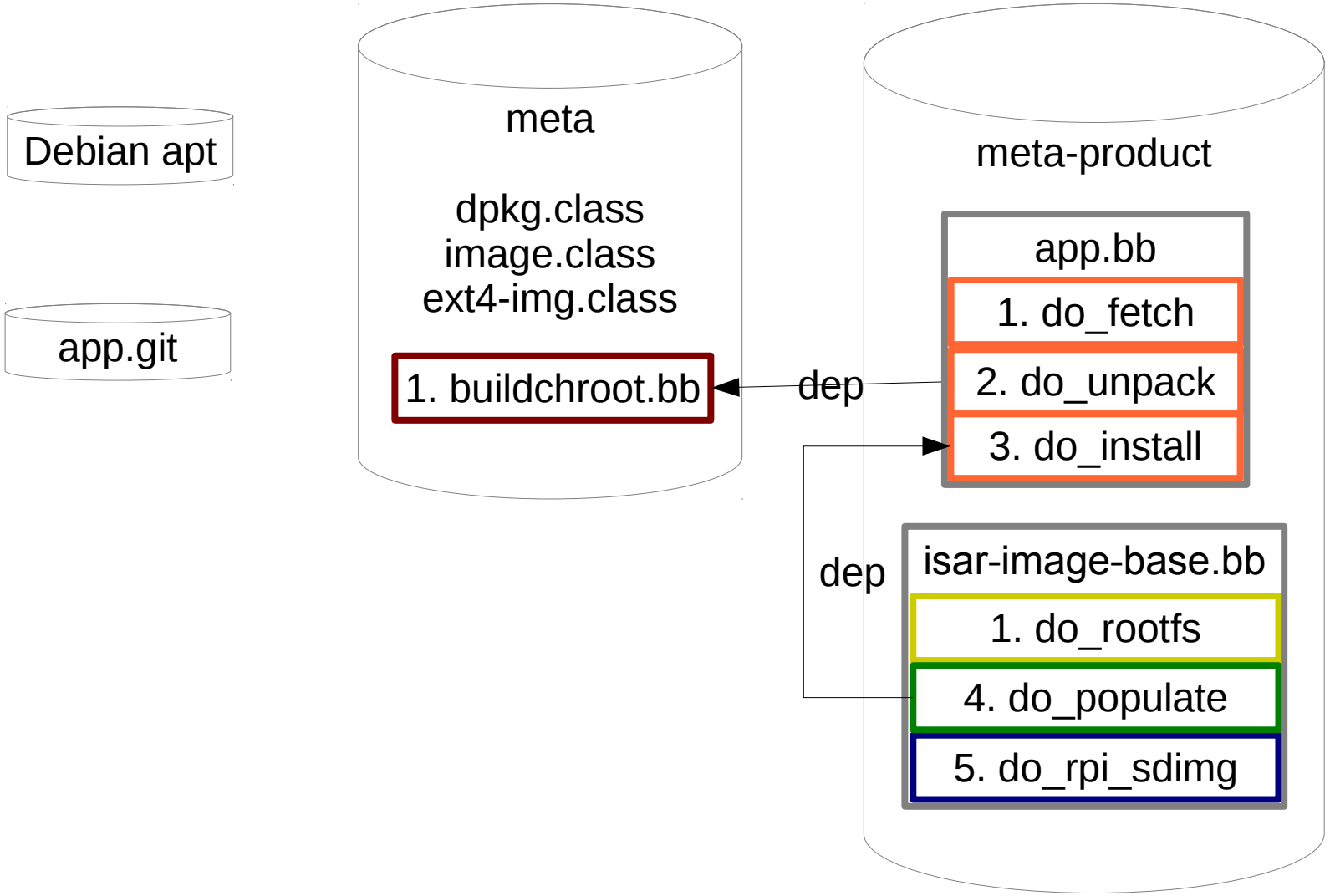
- `meta-isar/`: Product template layer
  - `recipes-app/hello/hello.bb`: Recipe building a target application binary Debian package

    ```
    SRC_URI = "git://github.com/ilbers/hello.git"
    SRVREV = "ad7065e"
    inherit dpkg
    ```
  - `recipes-core/images/`: Recipes producing target images on the host
    - `isar-image-base.bb`

      ```
      IMAGE_PREINSTALL += "apt dbus"
      do_rootfs () { ... }
      ```
    - `isar-image-debug.bb`

      ```
      IMAGE_PREINSTALL += "gdb strace"
      include isar-image-base.bb
      ```

Configuration Management



- Parallel execution through task dependencies

Isar: Current State

- Isar:
  - Core framework
  - Product template with `-base` and `-debug` images
- Example for building two products that share components:
  - https://github.com/ilbers/meta-two-products/
  - Targets: QEMU ARM, Raspberry Pi 1 Model B
  - Different buildchroots (Debian and Raspbian)
  - Image types: ext4, SD card with partitions
  - Product images:
    - Product A for QEMU
    - Product A for Raspberry Pi
    - Product B for QEMU

Using Isar

Common Development Tasks

- Build default images
- Add a new package
- Create a new product
- Override an upstream package
- Example product development workflow
- Build an older release

Build Default Images

```
$ git clone https://github.com/ilbers/isar
$ cd isar
$ . isar-init-build-env build
$ bitbake isar-image-base
```

Build dir

BitBake *target(s)*
Image name(s), e.g.:
isar-image-debug
multiconfig:rpi:isar-image-base

Add a New Package

- Create the package repo `hello.git`
  - Unpack the sources
  - Create `debian/*` if necessary (e.g., with `dh_make`)
- Create the recipe `meta-product/hello/hello.bb`:

```
SRC_URI = "git://server/hello.git"
SRCREV = "ad7065e"
inherit dpkg
```

- List package name in `IMAGE_INSTALL`

Create a New Product

- Copy `meta-isar` to your `meta-product` repo
- Add / modify packages
- Add / modify boards (*machines*)
- Add / modify images

Override an Upstream Package

- Quick and dirty: Image recipe (`inittab`, `fstab`, user creation, ...)
- Current way: Fork the respective package
- Vision: `sysvinit.bb`:

```
PV = "2.88dsf-59+myprj2"
SRC_URI = "http://server/sysvinit.dsc \
           file://99-inittab.patch"
SRC_URI[md5sum]="8f3ac1a308b594734ad3f47c809655f8"
inherit dpkg
```

Add to `IMAGE_INSTALL`

Product Development Workflow

- Release 1.0
  - Create repos for all components: Debian, apps, isar, meta-product
  - Develop your own code in app.git/master
  - Changes upstream code in pkg.git/yourbranch-1.0
  - Tag all input components, use the tags in meta-product recipes
  - Tag meta-product 1.0
  - Branch 1.0, maintain, tag 1.0.1...
- Release 2.0
  - Fast-forward upstream components: Debian, isar, modified pkgs
  - Develop your own code in app.git/master
  - Rebase modified upstream pkg.git/yourbranch-1.0 onto pkg.git/current master, put the result into pkg.git/yourbranch-2.0
  - Tag all input components, use the tags in meta-product recipes
  - Tag meta-product 2.0
  - Branch 2.0, maintain, tag 2.0.1...

Build an Older Release

- Making a release:
  - Tag the package repo
  - Recipes must use the tag (not a branch) as `SRCREV`
  - Tag meta-product
- Check out meta-product/tag 1.0
- Build the images

Reuse and Variability

Levels of development:

- meta: Isar core
  - meta-VENDOR1-bsp
  - meta-VENDOR2-libs
  - meta-COMPANY: Company-wide common stuff
  - meta-DEPT
  - meta-PRODUCT1
  - meta-PRODUCT2

Other Approaches: ELBE

Embedded Linux Build Environment: http://elbe-rfs.org/

- Same goals, similar project, different philosophy
- Central tool written in Python
  - Builds packages
  - Generates images
  - Creates a source CD with licenses
  - Many features OOTB
- Metadata in a single XML file
- Multiple products → Multiple XML files

Other Approaches: meta-debian

meta-debian:
http://elinux.org/images/7/74/LinuxCon2015_meta-debian_r7.pdf

- Different goals, different type of project, different focus
- Debian-based source distribution built with BitBake
- Builds packages from original sources + Debian patches
- Builds with a modified Yocto cross-compiler
- Recipes created from Debian rules manually

Other Approaches

More Debian image builders:
http://people.linaro.org/~riku.voipio/debian-images/

- “Each tool is tailored for the developer's use case and personal taste”
- Product development is more than creating a rootfs

The Isar Way

- Small tools for well-defined tasks
- Tools provide mechanism, policy is in metadata (recipes, conf files)
- Re-use as much as possible (tools, code, binaries)
- Familiar tools, structures, and workflows
- Self-contained, extensible build system
- Local adjustments to upstream: Reasonable effort
- Massive changes to upstream: Either avoid, or work with community
- You [will] want performance

Isar: Next Steps

- Isar:
  - Release creating Debian .dsc
  - Release building from Debian .dsc
  - Building from / to apt
  - Build caching: apt-aware build task (skip building if already in apt)
  - https://github.com/ilbers/isar/blob/master/TODO
- BitBake
  - Understand Debian build-deps (.dsc backend?)
- You! Yes, you!
  - Use it: https://github.com/ilbers/isar/
  - Ask for help: https://lists.debian.org/debian-embedded/
  - Suggestions?
  - Patches!
- Collaboration with other projects

Summary: Benefits of Isar

- **Quick project startup**
  - Familiar, mature tools
  - Product template with default images
- **Lower development and maintenance costs**
  - Modularity, flexibility, scalability through using BitBake
  - Focus on your core business
- **Fast builds**
  - Re-use pre-built Debian binary packages
  - Parallel building with BitBake and dpkg
- **Effective collaboration with vendors and community**
  - Proven-in-use structure and workflows of the Yocto project

References

- Code: https://github.com/ilbers/isar/
- User manual: https://github.com/ilbers/isar/wiki/User-Manual
- Mailing list: https://lists.debian.org/debian-embedded/

Questions?