Isar
Build Debian-Based Products with BitBake

Baurzhan Ismagulov

FOSDEM
Feb 4, 2017
Brussels, Belgium
Contents

• Motivation
• Introduction to Isar
• Under the hood
• What's new
• Summary
Motivation

Need a product integration system

- Produce complete, ready-to-use firmware images
- Use pre-built binary packages for efficiency
- Add own packages built from source
- One-command, on-demand image creation
- Low effort: Avoid touching every upstream package
- Reproducible builds
Further Requirements

- Customization of upstream packages
- Build several products from one repo
- Share components between products
- Workflow for maintaining code from several vendors
- Maintenance for 10+ years
Prior Art

Debian
- Pre-built binary packages
- Many standard tools
- Tested as a whole
- Clear licensing
- Long-term maintenance
- Security updates
- Fits big & small products
- No single-command integration OOTB
- N/A
- Collaboration process for distributed repos
- Builds available for chosen arches

Yocto
- Built from source
- Custom recipes
- Continuous development
- Clear licensing
- Continuous development
- Security process underway
- Issues with bigger products
- Single-command integration
- Modular, flexible integration tool
- Collaboration policy for centralized repos
- Build optimized for a given CPU
Isar: Finding the Right Mix

Integration System for Automated Root filesystem generation

Debian base system

BitBake integration tool

Isar

Yocto structure, workflow
The Isar Mix

- Debian: Stable, tested distro
- Security updates
- Distributed source repos
- Pre-built binary packages
- Reuse standard tools

- BitBake: Flexible and efficient
- Centralized metadata repos
- Ready-to-use product setup
- Change as little as possible
- Build as little as possible
- Reuse tools, structure, workflows
Isar: Reuse vs. Rebuild

- Debian packages: Use pre-built ones
- Modified Debian packages: Build once, then use pre-built ones
- Product-specific packages:
  - U-Boot: Build once, then use pre-built one
- Rebuild every time:
  - U-Boot
  - Kernel
  - Drivers
  - Libs
  - Apps
How Isar Works

- Native compilation with `dpkg-buildpackage` under QEMU armhf
- `buildchroot` dir: "SDK" and "devshell"
Using Isar: 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
Isar: Default Product

- Recipes: Files containing tasks executed in a defined order
  - meta/recipes-devtools/buildchroot/buildchroot.bb
  - meta-isar/recipes-app/hello/hello.bb
  - meta-isar/recipes-core/images/isar-image-base.bb
  - meta-isar/recipes-core/images/isar-image-debug.bb

- Tasks: Functions implemented in shell or Python
  - Ordering through tasks dependencies
  - Performance through parallel execution of independent tasks

- Layers: Organize code according to upstreams
  - meta: Isar core layer metadata (build scripts as opposed to source)
  - meta-isar: Product template layer metadata
Tasks and Layers

1. buildchroot.bb
2. do_unpack
3. do_install
4. do_populate
5. do_rpi_sdimg

Debian apt

hello.git
Starting Your Own Project

- Clone \texttt{isar} and push as your own repo
- Define your product images
- Add your packages
- Add your boards
Adding Images

• To add an image, copy `isar-image-base.bb` as `meta-PRODUCT/recipes-core/images/PRODUCT-image-base.bb` and modify to taste. E.g., to install more packages in the target rootfs:

```bash
IMAGE_PREINSTALL += "apt dbus openssh-server"
```

• To provide an image with more functionality (e.g., developer tools), create `meta-PRODUCT/recipes-core/images/PRODUCT-image-base.bb`:

```bash
include recipes-core/images/PRODUCT-image-base.bb
IMAGE_PREINSTALL += "gdb lsof strace"
```

• All vars from `.bb` and `.conf` files are added to a single global namespace

• Why not use a single config file? Not possible to reflect complexity without repeating yourself
Adding Packages

- Create the package repo `app1.git`
  - Commit unpacked sources
  - Create `debian/*` if necessary (e.g., with `dh_make`)
- Create recipe `meta-PRODUCT/recipes-app/app1/app1.bb`:
  ```
  SRC_URI = "git://server/app1.git;protocol=http"
  SRCREV="1234567890123456789012345678901234567890"
  inherit dpkg
  ```
- Edit `meta-PRODUCT/recipes-core/images/PRODUCT-image-base.bb` and list the recipe name (without .bb) in:
  ```
  IMAGE_INSTALL += "app1"
  ```
- Available tasks: `do_fetch`, `do_build`
Adding Boards

- Board *(machine)* definitions are files with Debian settings for the board

- `meta-isar/conf/multiconfig/qemuarm.conf`:
  - MACHINE = "qemuarm"
  - DISTRO = "debian-jessie"
  - DISTRO_ARCH = "armhf"

- `meta-isar/conf/machine/qemuarm.conf`:
  - IMAGE_PREINSTALL = "linux-image-3.16.0-4-armmp"
  - KERNEL_IMAGE = "vmlinuz-3.16.0-4-armmp"
  - INITRD_IMAGE = "initrd.img-3.16.0-4-armmp"
  - MACHINE_SERIAL = "ttyAMA0"
  - ROOTFS_DEV = "mmcblk0"
  - BAUDRATE_TTY = "9600"
  - IMAGE_TYPE = "ext4-img"
Distro Definition

- `meta-isar/conf/distro/debian-jessie.conf`:
  
  DISTRO_SUITE = "jessie"
  DISTRO_COMPONENTS = "main contrib non-free"
  DISTRO_APT_SOURCE = "http://httpredir.d.o/debian"
  DISTRO_CONFIG_SCRIPT = "debian-configscript.sh"
Organize Code in Layers

• Your layers might look like:
  • `meta: Isar core`
  • `meta-VENDOR1-bsp` (U-Boot, kernel, ...)
  • `meta-VENDOR2-libs` (codecs, ...)
  • `meta-COMPANY`: Company-wide common stuff
  • `meta-DIV`: Division-wide common stuff
  • `meta-PRODUCT1` (app1, ...)
  • `meta-PRODUCT2` (app2, ...)

• You may use `meta-isar` as a template for `meta-PRODUCT`.

• Layers may be in one or in different repos.
• Tools like `repo` can be used to clone all repos.
Include Layers in Build

- Layer clone dir must be listed in `build/conf/bblayers.conf`:
  ```
  BBLAYERS ?= " \
  /home/user/PRODUCT/meta \ 
  /home/user/PRODUCT/meta-PRODUCT \ 
  "
  ```

- `isar-init-build-env` creates the `build` dir and generates `build/conf/bblayers.conf` (existing files are not overwritten).

- Edit `meta/conf/bblayer.conf.sample` so that your desired `bblayers.conf` is generated:
  ```
  BBLAYERS ?= " \
  ##ISARROOT##/meta \ 
  ##ISARROOT##/meta-PRODUCT \ 
  "
  ```
Override an Upstream Package

- Quick and dirty: Image recipe (\texttt{inittab}, \texttt{fstab}, user creation, ...)
- Current way: Fork the respective package
- Vision: \texttt{sysvinit.bb}:
  \begin{verbatim}
  PV = "2.88dsf-59+myprj2"
  SRC_URI = "http://server/sysvinit.dsc \
             file://99-inittab.patch"
  SRC_URI[md5sum]="8f3ac1a308b594734ad3f47c809655f8"
  inherit dpkg
  \end{verbatim}
Add to \texttt{IMAGE\_INSTALL}.
Under the Hood

- **isar/**: Repo root
- **bitbake/**: Recipe interpreter (copy, pulled from time to time)
- **meta/**: Core layer
- **meta-isar/**: Product template layer
- **isar-init-build-env**: Build environment initialization script. Must be sourced in the current shell (`isar-init-build-env` or `source isar-init-build-env`), not executed in a sub-shell (`isar-init-build-env`).
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. Includes 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 variability through multiple images and `MACHINE / DISTRO / DEPENDS`**
- **Component sharing through `DEPENDS`, common packages, `MACHINE`, `DISTRO`**
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
In Development

- i386 support
- Jessie fixes
- Image creation with \texttt{wic}
- Documentation improvements
Isar: Combining the Best

- **Quality**: Debian is a stable, tested distro with security updates
- **Quick start**: Product template, familiar tools
- **Customization**: BitBake is a flexible, efficient integration tool
- **Performance**: Pre-built packages, parallel task execution
References

- Code: https://github.com/ilbers/isar/
- Mailing list: https://lists.debian.org/debian-embedded/
Questions?