Building Products with Debian and Isar

Jan Kiszka
Baurzhan Ismagulov

Open Source Summit Japan 2019

About us

Jan Kiszka <jan.kiszka@siemens.com>

- Member of embedded Linux team at Siemens Corporate Technology
- (In-house) consultant, architect, developer for OSS
- Focus on kernel, real-time, virtualization, embedded build systems
- Upstream contributor / maintainer

Baurzhan Ismagulov <ibr@ilbers.de>

- Work for ilbers GmbH
- Provide software development services
- Focus on embedded, real-time, networking, virtualization
- Contribute to Linux and other FOSS projects

Agenda

Motivation

- Build Linux-based products
  - Linux base system
  - Customizations
  - Product-specific applications
- We want a product build system
- Tools are distribution-specific

Which Distribution?

Source distributions, e.g.:

Yocto, OE, buildroot

- Toolchain bootstrap, cross-build => long build times
- Highly customizable, smaller footprint
- High maintenance costs

Binary distributions, e.g.:

Debian, Ubuntu, SUSE, Fedora

- Prebuilt binary packages => keep upstream QA
- Larger images, slower boot by default
- Maintained releases

Debian

- Mature, high-quality, mainstream Linux distribution
- Support for many new and old hardware architectures
- Suitable for small and big installations
- Security updates
- License information

Embedded Debian: Wishlist

- Build all sources in the right order with one command
- Bootable image generator
- Easy package customization
- Reproducible images
- Efficient variant management

Isar

- Integration System for Automated Root filesystem generation
  https://github.com/ilbers/isar
- Goals
  - Build systems in a Debian way
  - Developer-centric workflow: One-command building
  - Make customizations easy and repeatable
  - Efficient building
- The best of both worlds
  - Debian: Tested binary packages, tools, security updates
  - Yocto: bitbake, recipes, layers
- Reuse Yocto knowledge of developers

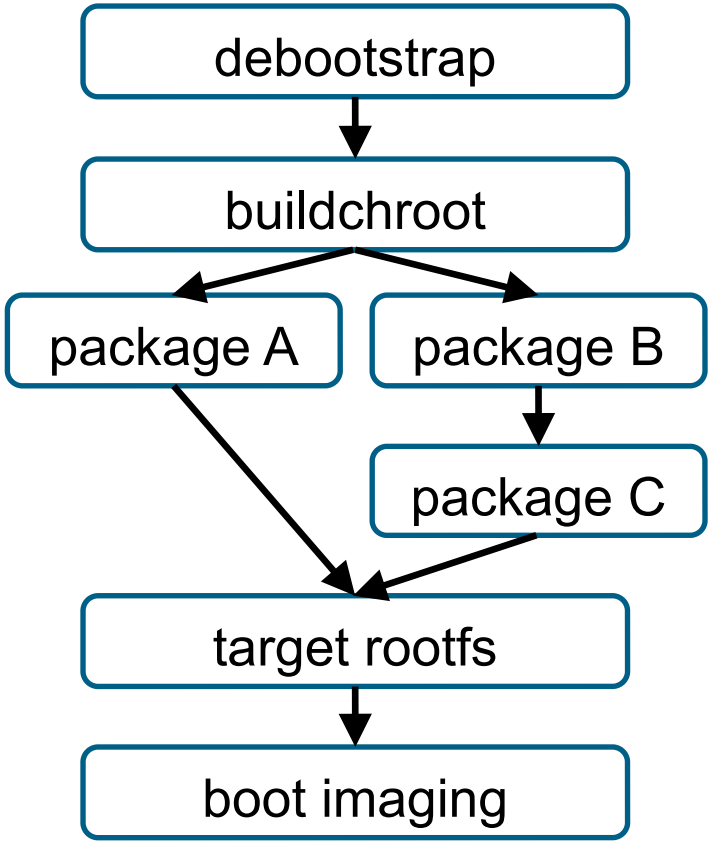
Hello, Image

```
$ git clone https://github.com/ilbers/isar
# Install packages and configure sudo as described in user_manual.md
$ . isar-init-build-env ../build
$ bitbake multiconfig:rpi-stretch:isar-image-base

$ ls tmp/deploy/images/rpi
isar-image-base.rpi-sdimg
```

Image Generation Sequence of Isar

1. debootstrap Debian for target,
   also for host if cross-building
2. Create buildchroots (target and host)
3. Build custom Debian packages
   - pre-debianized packages
   - ad-hoc debianized packages
     (customizations/overlays, u-boot, …)
4. Assemble rootfs
   - debootstrap output
   - external packages
   - self-built packages
5. Run imager (typically wic)
   - Filesystem image generation
   - Partitioning
   - Bootloader installation and configuration



BitBake

- A tool for task description and execution
- Recipes:
  - Describe tasks (python, shell)
  - May depend on other recipes
- Tasks: e.g., fetch, patch, build
- Dependencies define:
  - Which tasks to execute
  - Execution order
- Classes: Repetitive stuff
- Efficient building
  - Parallel building
  - Multiple targets in one run (multiconfig)

dpkg.bbclass

```
do_apt_fetch() { ... }

do_install_builddeps() { ... }

do_runbuild() { ... }
```

hello.bb

```
DESCRIPTION = "Hello app"
DEPENDS += "libhello"
SRC_URI = "\
  git://gh.com/ilbers/hello.git \
  file://0001.patch"
SRCREV = "86cc719..."
inherit dpkg
```

Building a Product

Requirements

- Directly bootable images for product boards
- Product-specific base system customizations
- Product-specific software
- Reproducible by developers

Tasks

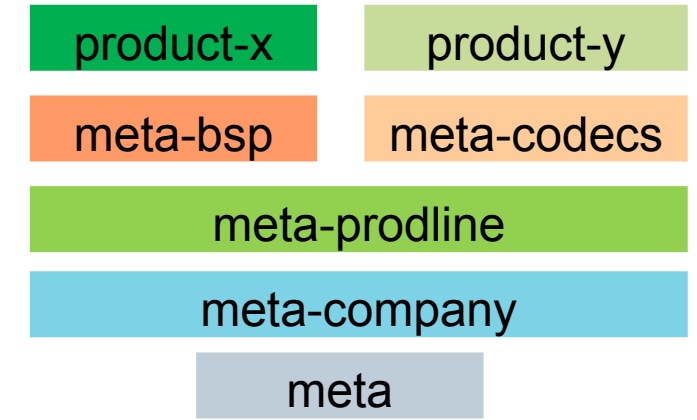
- Generate rootfs
- Perform customizations
- Build custom software: boot loaders, kernels, drivers, libs, apps
- Generate flash images for physical boards

https://en.wikipedia.org/wiki/Newbear_77-68

Layers

- Group by product / department / feature...
  - Config files, recipes, classes / include files, ...
- E.g.:
  - `meta`: Isar core
  - `meta-company`: Company common stuff
  - `meta-prodline`: Product line common stuff
  - `meta-bsp`: Product X COTS BSP
  - `product-x`: Product X-specific stuff
  - `meta-codecs`: Product Y COTS libs
  - `product-y`: Product Y-specific stuff

```
product-x
├── conf
│   └── layer.conf
└── recipes-core
    ├── customizations
    │   ├── customizations.bb
    │   └── files
    │       └── postinst
    └── images
        └── x-image.bb
```



Product Lifecycle

- Deployment: Copy image to flash
- Updates: Product-specific
  - Install packages via apt
  - Overwrite complete images
- Security updates: Packages and alerts from Debian
- Long-term support for chosen packages by Debian and Civil Infrastructure Project
- License compliance: Product-specific
  - Rely on Debian license information
  - FOSSology + manual checking

Is Isar Right for Me?

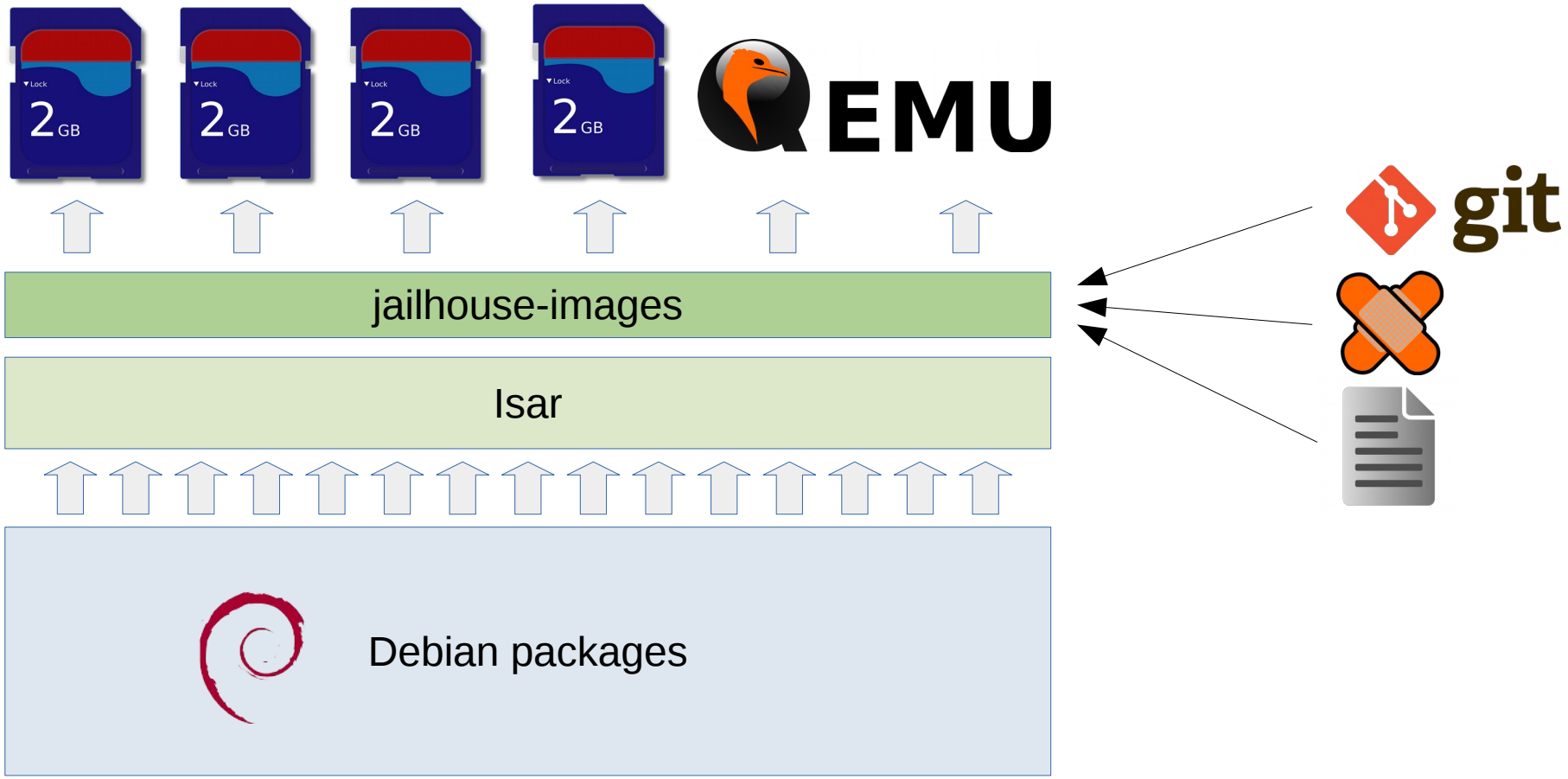
- Build with one command
- No infrastructure needed upfront
- Use mainstream build tools
- Reuse binary packages
- Reuse code in multiple products
- Collaborate with multiple vendors

Who Uses Isar?

- Jailhouse images
  https://github.com/siemens/jailhouse-images
- Xenomai images
  https://gitlab.denx.de/Xenomai/xenomai-images
- Civil Infrastructure Project core
  https://gitlab.com/cip-project/cip-core/isar-cip-core
- Used at Siemens for an increasing number of embedded products
- Core of Mentor Embedded Linux Omni OS
  https://www.mentor.com/embedded-software/linux/mel-omni-os

https://en.wikipedia.org/wiki/Isar

Example: jailhouse-images



jailhouse-images in a Nutshell

- Started early 2018 with x86 QEMU/KVM setup
- By now 8 targets:
  - 2 virtual
  - 6 physical
  - 3 architectures
- Prerequisite for building: docker (privileged)
- `git clone https://github.com/siemens/jailhouse-images`
- `./build-images.sh`
- `./start-qemu.sh <ARCH>`
- `dd if=target.img of=/dev/mmcblk0 …`

```
# ./build-images.sh
Available images demo images:
 1: QEMU/KVM Intel-x86 virtual target
 2: QEMU ARM64 virtual target
 3: Orange Pi Zero (256 MB edition)
 4: Intel NUC (NUC6CAY, 8 GB RAM)
 5: Marvell ESPRESSObin (1 GB edition)
 6: Marvell MACCHIATObin
 7: LeMaker HiKey (Kirin 620 SoC, 2 GB editi
 8: Avnet Ultra96
 0: all (may take hours...)

Select images to build (space-separated
index list): 1 2 4
```

Distro or Custom Kernel?

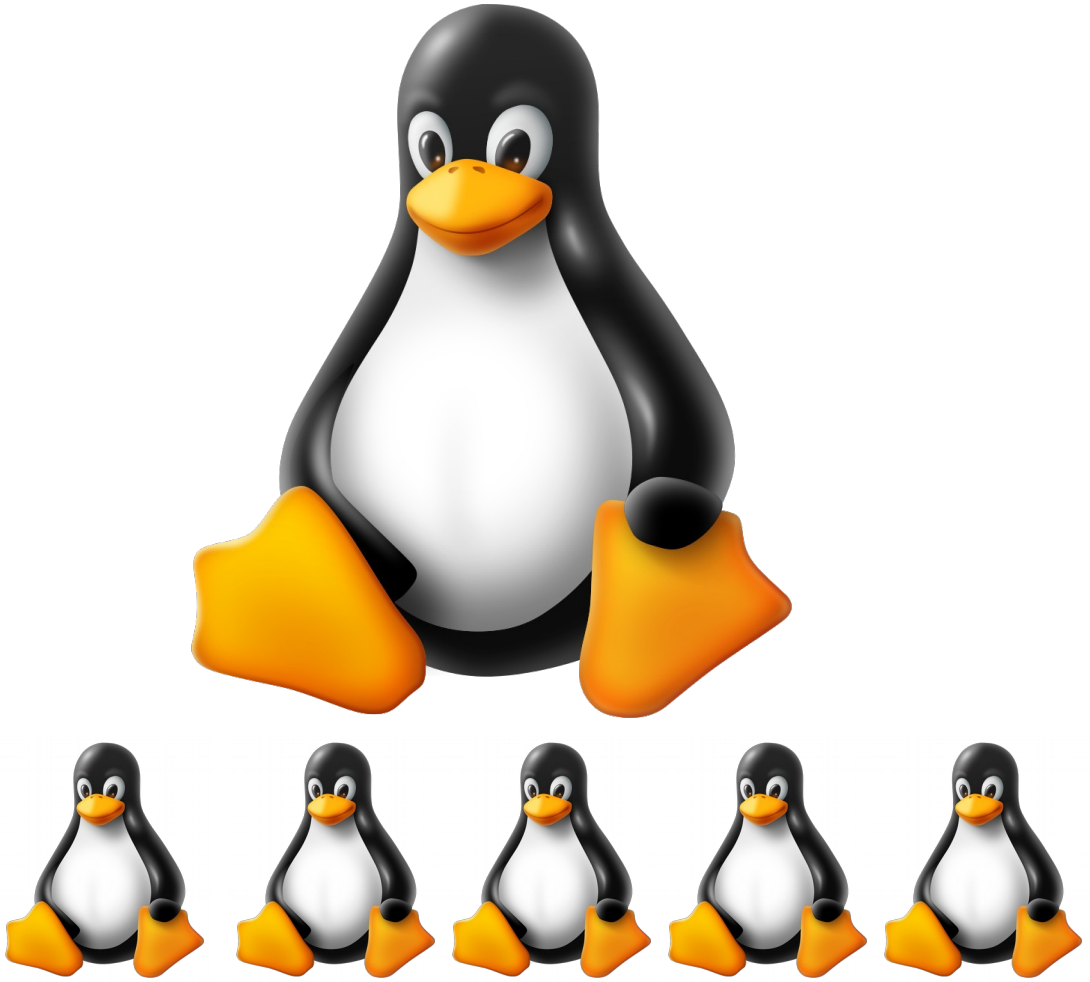
- Set `KERNEL_NAME` to select target `linux-${KERNEL_NAME}`
- Distro kernel is Isar default (and automatically provided)
- `linux-my-lovely-kernel.bb`
  - `require recipes-kernel/linux/linux-custom.inc`
  - Usage straightforward, just needs `SRC_URI` and `KERNEL_DEFCONFIG`
- `meta-isar/recipes-kernel/linux` contains two custom kernel examples (CIP and mainline)

Adding Out-of-Tree Kernel Modules

- `require recipes-kernel/linux-module/module.inc`
- Package name = recipe name + kernel name
  e.g. `example-module-${KERNEL_NAME}`
- Module Makefile according to kbuild rules, e.g.

```
obj-m := example-module.o
```

- `AUTOLOAD`: list of modules to be loaded during boot



U-Boot

- Distro U-Boot supported, though mostly available on maker boards (Raspberry Pi & Co.)
- `require recipes-bsp/u-boot/u-boot-custom.inc`
  - Trival case just needs setting `U_BOOT_CONFIG` and `U_BOOT_BIN`
  - Complex case may require overloading `debian/rules` file
  - Examples
    - meta-isar: mainline 2018.09
    - jailhouse-images: MACCHIATObin, Ultra96

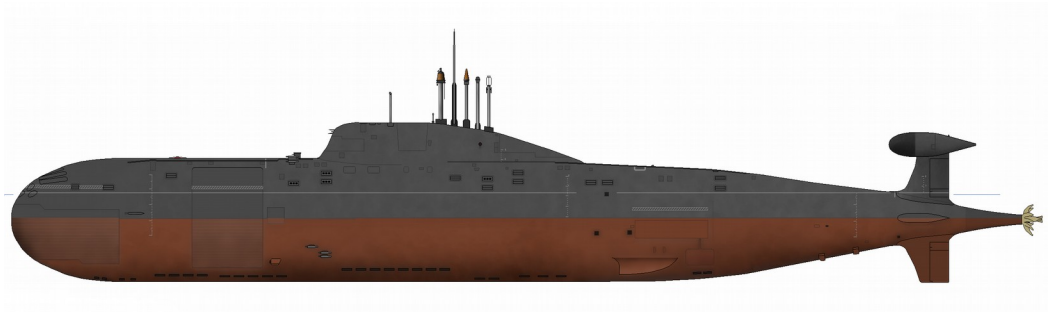
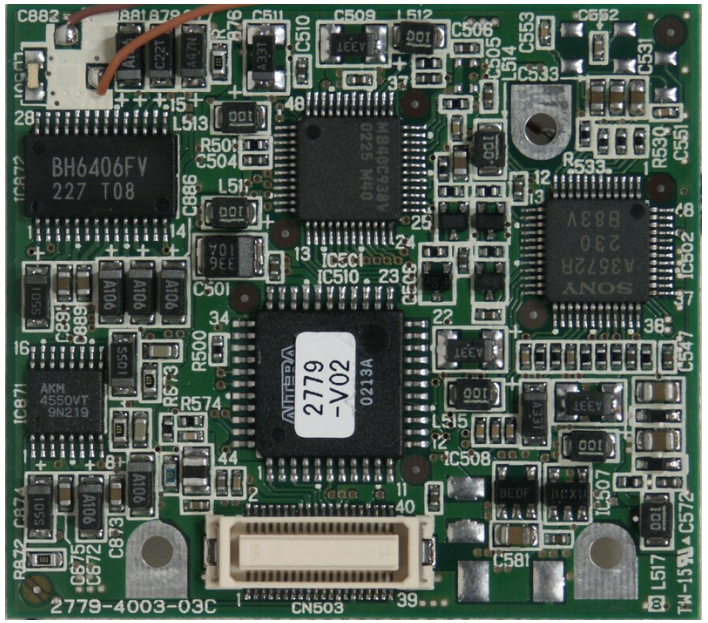


Image Partitioning and Generation

- Default case: wic tool
- .wks files control partition layout and content
- Same core as OE, but additional source plugins
  - bootimg-efi-isar (EFI-based boot partition)
  - bootimg-pcbios-isar (legacy boot partition)
  - rootfs-u-boot (used with u-boot-script)
- wic references
  - Yocto manual: https://www.yoctoproject.org/docs/latest/dev-manual/dev-manual.html#creating-partitioned-images-using-wic
  - Example wks files
- Additional image formats (more planned)
  - ext4.img
  - fit
  - ubifs
  - targz
  - ubi

Adding a New Board

- Key parameters
  - Name, selects `conf/machine/<machine>.conf`
  - `DISTRO_ARCH`, set in `<machine>.conf`
- Pick a kernel
  - `KERNEL_NAME`, set in `<machine>.conf` if deviating
- Define `IMAGE_TYPE`, typically `"wic-img"`
- If wic, define `WKS_FILE`
  - Typically set per machine
- Set bootloader, typically in `<machine>.conf`
  - Install via wic => tune WKS file
  - Add to `IMAGER_INSTALL`
    - Predefined: `GRUB_IMAGER_INSTALL`, `SYSLINUX_IMAGER_INSTALL`
  - Add to `IMAGER_BUILD_DEPS` if built from source
  - U-Boot: `U_BOOT_CONFIG`, `U_BOOT_BIN`
- **Study the examples, ask the community!**



Customizing Debian Packages

- Reasons to repackage
  - Bug or missing feature in upstream package
  - Size or configuration tuning
  - ...
- Steps with Isar
  - `inherit dpkg`
  - Fetch original source package
    `SRC_URI = "apt://${PN}"`
  - Apply changes, e.g.
    `SRC_URI += "file://bugfix.patch"`
  - Trigger changelog update

```
do_prepare_build() {
    deb_add_changelog
}
```

- Example: `meta-isar/recipes-app/hello`

Configuration Management for Isar

- Recommendation: https://github.com/siemens/kas
  - clones and checkouts layers
  - creates default bitbake settings (machine, architecture, …)
  - launches a minimal build environment
  - Initiates the bitbake build process
- Configured via yaml (or JSON) files
- All relevant configuration data is centralized and easily versionable
- https://kas.readthedocs.io
- Also provides docker build container

- Alternatives
  - Layers in git sub-modules
  - repo tool
  - Custom setup scripts

```
header:
  version: 8

distro: cip-core-buster
target: cip-core-image

machine: simatic-ipc227e

repos:
  cip-core:

  isar:
    url: https://github.com/ilbers/isar
    refspec: bdf8d29eacfd
    layers:
      meta:
```

Summary

- Embedded system building remains complex
  - Non-upstream changes
  - Adjustments to standard packages
  - Unpackaged components
  - Full, preconfigured images needed
- Isar bridges the gap between standard distro installation and customized embedded image building
  - Enables binary Debian package feed
  - Toolbox for typical customizations
  - Reusable & shareable
  - Extensible
- More users of Isar to come, OSS and products
- Feedback welcome!

Getting Started

- Code: https://github.com/ilbers/isar
- Mailing list: isar-users@googlegroups.com

- Examples
  - https://github.com/siemens/jailhouse-images
  - https://gitlab.denx.de/xenomai/xenomai-images

- BitBake manual:
  - https://www.yoctoproject.org/docs/latest/bitbake-user-manual/bitbake-user-manual.html

Thank You! Any Questions?

Jan Kiszka jan.kiszka@siemens.com
Baurzhan Ismagulov ibr@ilbers.de