Introduction to PetaLinux

Fernando Rincón
Julio Dondo

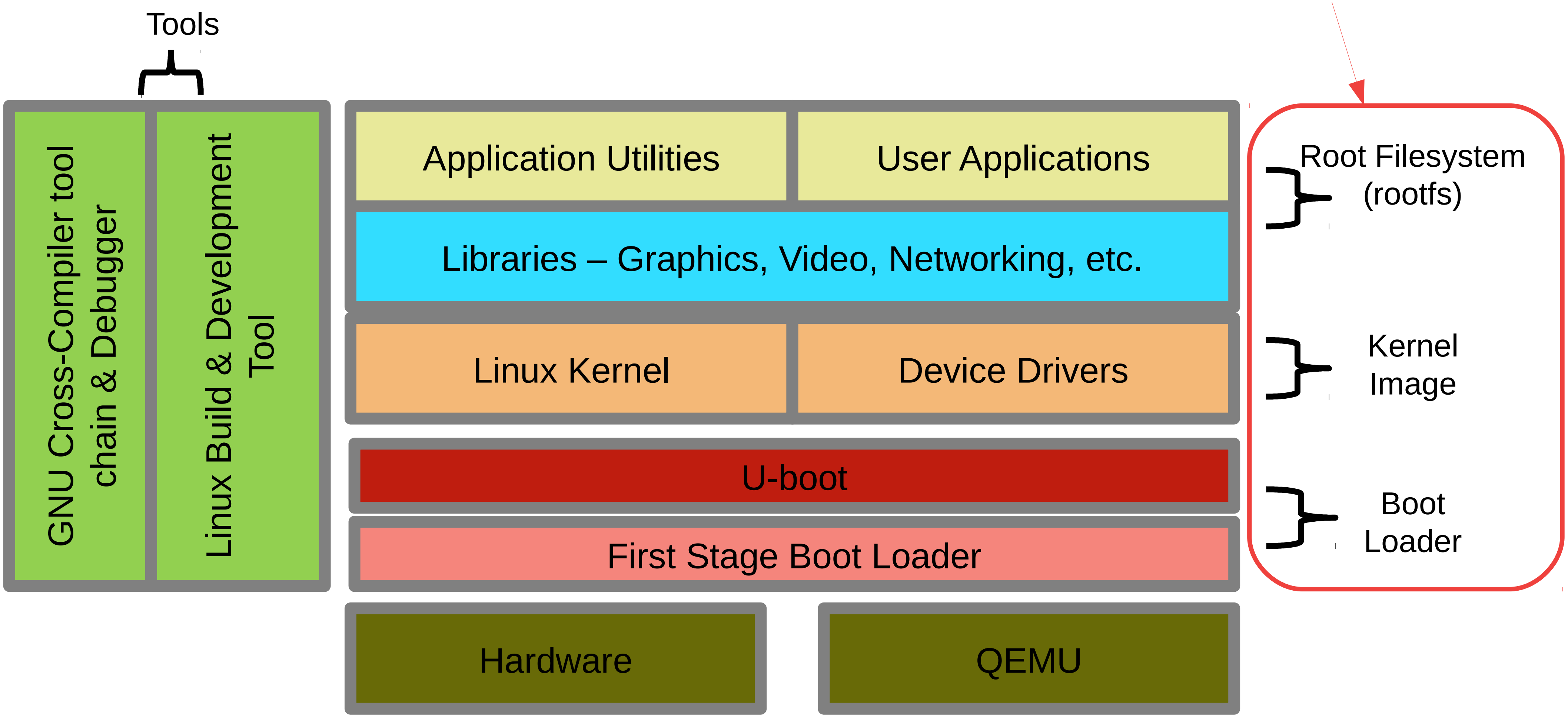
Contents

- Why PetaLinux?
- PetaLinux Tools & Flow
  - Project creation
  - Project configuration
  - Project building
  - Project booting
  - Project packaging

Why PetaLinux?

Linux Components

All these should be built!!



Why PetaLinux?

- Buiding a Linux system requires:
  - Building the bootloader from its source code
  - Building the Linux kernel:
    - Requires a *Toolchain* for cross-platform compilation (*baremetal* compiler)
    - Kernel source code
    - Drivers source code (for peripherals not in the standard kernel tree)
  - Building a root filesystem
    - To hold the libraries, graphical environment, user applications, ….
  - Buiding system and user applications
    - Such as system services: shell, network communication, …
    - And final user applications
    - But requiring a different compiler:
      - Also *cross-compiler* but using *linux* libraries instead of baremetal

Why PetaLinux?

- Lots of documentation and good books about the Linux building process from scratch
- And all code specific to Xilinx boards and drivers is publicly available:
  - https://github.com/xilinx
  - Because the standard kernel tree and bootloaders do not directly support all Xilinx Hw
- However, this is a really painful and long process to go for non-experts

Why PetaLinux?

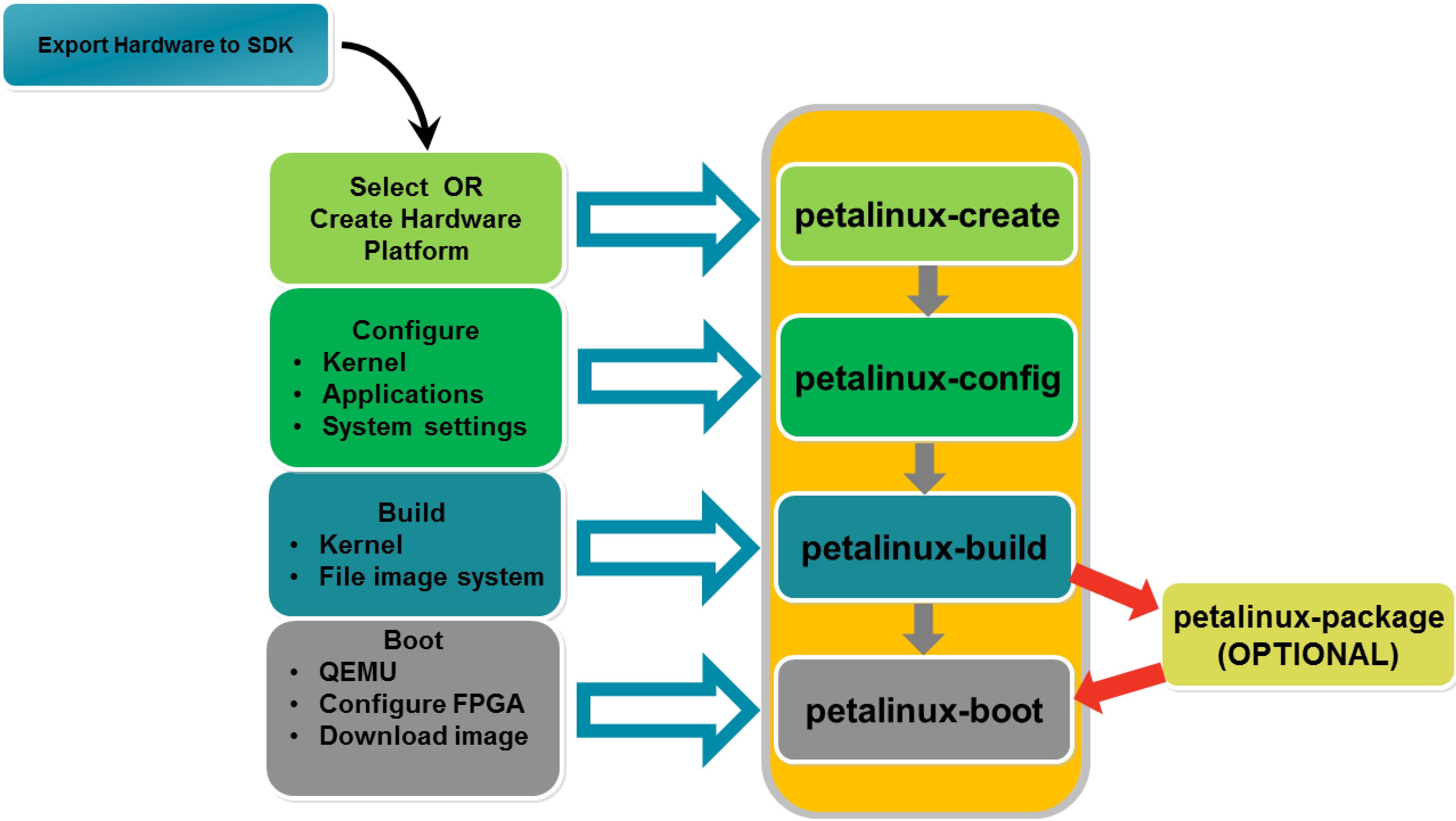
- PetaLinux is a all-in-one development environment
  - Kernel/library/user application sources
  - Compiler toolchains
  - Hardware reference designs
  - PetaLinux BSP generator
  - QEMU full-system simulator
  - Tools to bring it all together
  - Lots of documentation

PetaLinux requirements

- **Host machine**
  - Linux OSrequirements (supported)
    - Red Hat Enterprise Linux 6.5/6.6/7.0 (64-bit)
    - CentOS 7.0 (64-bit)
    - SUSE Enterprise 12.0 (64-bit)
    - Ubuntu 14.0.4 (64-bit)
  - Hardware requirements
    - 4 GB RAM
    - 2 GHz CPU
    - Minimum of 5 GB free HDD space
  - Xilinx Requirements
    - Vivado Design Suite 2016.4
    - Petalinux Tools 2016.4

- **Target machine**
  - ARM® CortexTM-A9 MPcore CPU
  - External memory controller
    - 32 MB recommended minimum
  - Interrupt controller
  - Triple timer count (TTC)
  - Other I/O as required
    - Serial, Ethernet
    - Flash memory (NOR/NAND/QSPI)

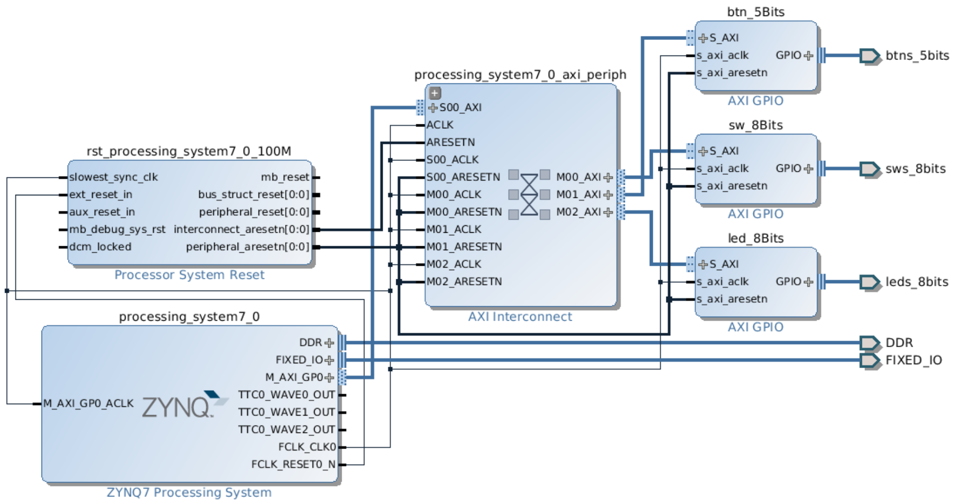
Petalinux Tools Flow



Petalinux Project Flow

- **Create a hardware design**
  - Launch the Vivado Design Suite
  - Use Vivado IP integrator (IPI) to create a block design
    - Add processor (ARM Cortex-A9 or MicroBlazeTM processor)
    - Add required peripherals such as AXI GPIO, AXI Interrupt Controller, Timer
- **Synthesis, implementation, and bitstream**
- **Export the hardware design to SDK**

Export Hardware to SDK



PetaLinux Project Flow

Select OR
Create Hardware
Platform

- Create the PetaLinux project
  - `petalinux-create` tool
  - Builds the basic project structure
  - Two options
    - **From a template:**
      - General case for an architecture or board
      - Preconfigured
      - Customized hardware

```
$ petalinux-create [options] --type project -s <path to template>
```

- **From a BSP:**
  - Previously packed from a working configuration
  - May include more hw & sw than required

PetaLinux Project Flow

- PetaLinux Project Structure
  - A Built linux system is composed of:
    - First Stage Boot Loader
    - U-Boot
    - Linux Kernel
    - Device Tree
    - Root Filesystem, which typically includes
      - Prebuilt packages
      - User applications (optional)
      - User modules (optional)

PetaLinux Project Flow

- PetaLinux Project Structure

```
.
├── components
│   └── plnx_workspace
│       ├── device-tree-generation
│       ├── fsbl
│       ├── fsbl_bsp
│       ├── fsbl_hwproj
│       └── SDK.log
├── config.project
├── hardware
│   └── avnet-digilent-zedboard-2016.4
│
├── pre-built
│   └── linux
│       ├── etc
│       ├── images
│       └── implementation
└── project-spec
    ├── attributes
    ├── configs
    ├── hw-description
    ├── meta-plnx-generated
    └── meta-user
        ├── README
        ├── recipes-apps
        └── ...
```

Other components apart from the kernel and user apps: fsbl, device-tree

Project configuration file

Vivado hardware project

Pre-built images and bitstream

Specification of the project

Kernel and rootfs configuration files

Hw description exported from Vivado

Recipes for petalinux generated components

Recipes for user generated components: applications and overrides of the default platform settings

PetaLinux Project Flow

- Configure the project:
  - Select the characteristics of the kernel, booting arguments, root filesystem location & contents, …
  - **petalinux-config** tool
  - To import the hardware platform generated in vivado:
    - cd to the location of the exported .hdf file

```
$ petalinux-config --get-hw-description -p <path to project> \
  --template zynq
```

  - To configure the PetaLinux in general

```
$ petalinux-config
```

  - To configure the kernel

```
$ petalinux-config -c kernel
```

  - To configure the root filesystem

```
$ petalinux-config -c rootfs
```

Configure
- **Kernel**
- **Applications**
- **System settings**

PetaLinux Project Flow

- Build the project:
  - **petalinux-build** tool
  - Can generate the whole project: bootloader, kernel, root filesystem and target image

kernel
rootfs

```
$ petalinux-build
```

- The bootable images will be found at: <project>/images/linux
  - Or just single components:

```
$ petalinux-build -component <component>
```

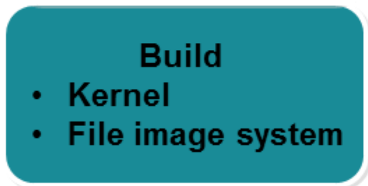
  - In order to clean the project:

```
$ petalinux-build -x clean
```

  - Or more drastically:

```
$ petalinux-build -x mrproper
```

  - Any component can be cleaned individually



PetaLinux Project Flow

Boot
- **QEMU**
- **Configure FPGA**
- **Download image**

- Boot the image
  - **petalinux-boot** tool
  - Can boot on a real processor (Microblaze / Zynq)
  - But also on an emulator (QEMU)

```
$ petalinux-boot--qemu|--jtag -c|--component <COMPONENT> [options]
```

  - Some examples:
    - Boot the prebuilt images

1 – FSBL
2 – Uboot
3 – Kernel

```
$ petalinux-boot --jtag -prebuilt 1|2|3
```

    - Download current bitstream

```
$ petalinux-boot --jtag --fpga --bitstream <BITSTREAM>
```

    - Download current kernel

```
$ petalinux-boot --jtag --kernel
```

PetaLinux Project Flow

- **QEMU: Quick EMUlator**
  - Open Source (GPL) multi-architecture emulator
  - Like a Virtual Machine
    - Emulates CPU architecture (e.g. emulating a ARM CPU on a x86 host)
    - Emulates Devices (e.g. SPI Flash, Ethernet, SDHCI + SD Card, USB HCI, etc.)
    - Not a simulator, has no timing accuracy (can however interact with simulators)
    - Can load a system machine model from a Device Tree (this is only for the Xilinx QEMU)
  - Great way to test your system without needing hardware
    - Quick boot times, no need to play around with JTAG/SD cards/etc to get a booting system

PetaLinux Project Flow

- **QEMU Boot Flows**
  - FSBL is not compatible or required
    - QEMU handles the Zynq Initialization
  - You can boot into U-Boot
    - And then follow a boot flow from a storage device
  - Or you can boot directly to the Kernel
    - QEMU can handle kernel, root file system and device tree loading
      - This is much quicker that loading U-Boot, and is the recommended flow

PetaLinux Project Flow

- Other useful tool: **`petalinux-package`**
  - packages various image format, firmware, prebuilt and BSPs

```
$ petalinux-package --boot| --bsp| --firmware| --image| --prebuilt [options]
```