ADIUVO
ENGINEERING AND TRAINING, LTD.

PetaLinux

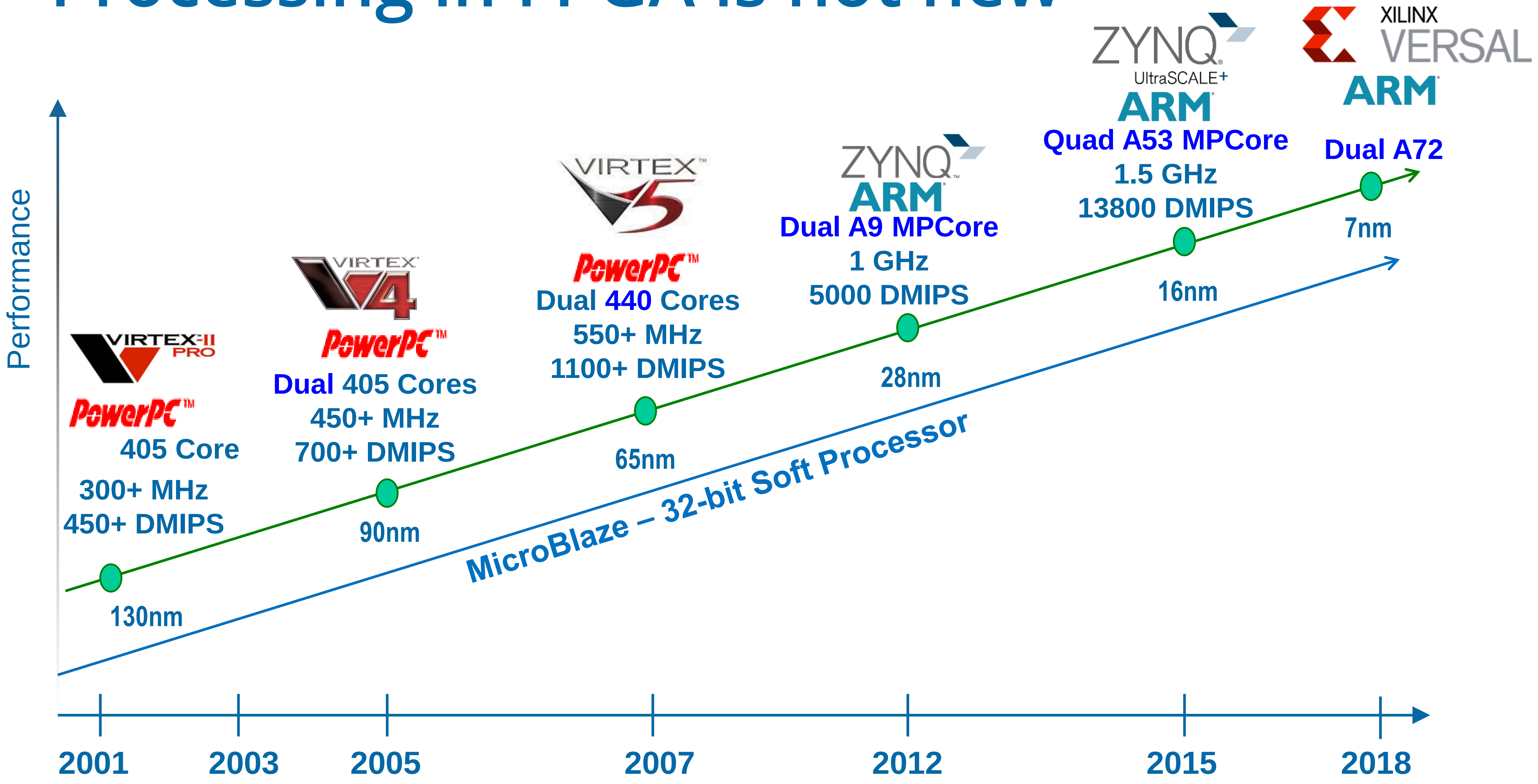
Agenda

- Introduction
- Processing Solutions in AMD Xilinx Devices
- Petalinux
  - What is Petalinux
  - Relationship to Yocto
  - Device Trees
  - BSP
  - Bit Bake
- Lab

Processing Solutions

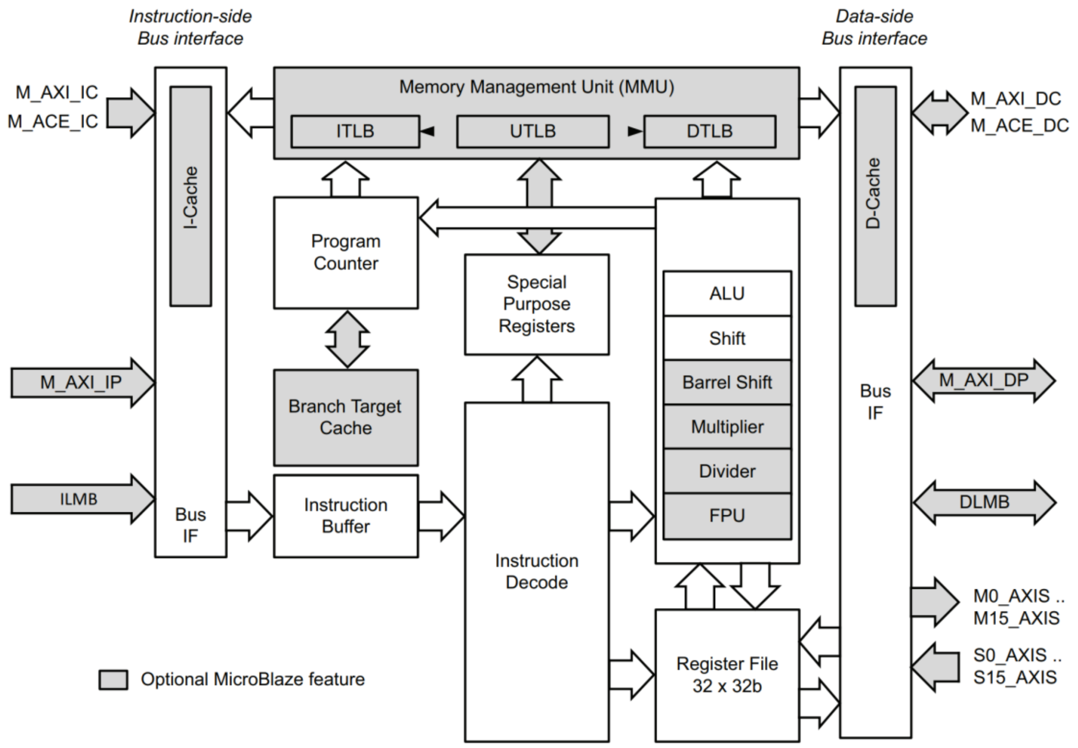
ADIUVO
ENGINEERING AND TRAINING, LTD.

Processing in FPGA is not new



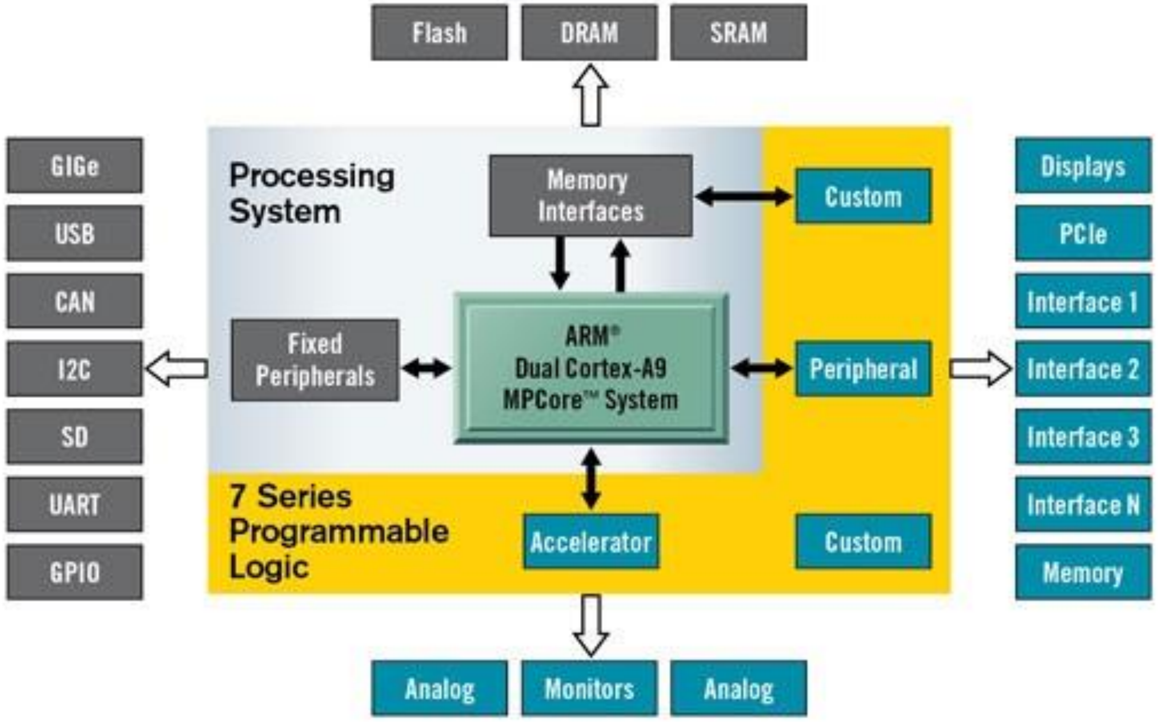
MicroBlaze

- (Mostly) softcore processor
- Harvard Architecture
- Scalable from small micro to running Linux
- Performance scales with configuration options
- Can be implemented in TMR solution (very useful for high rel)



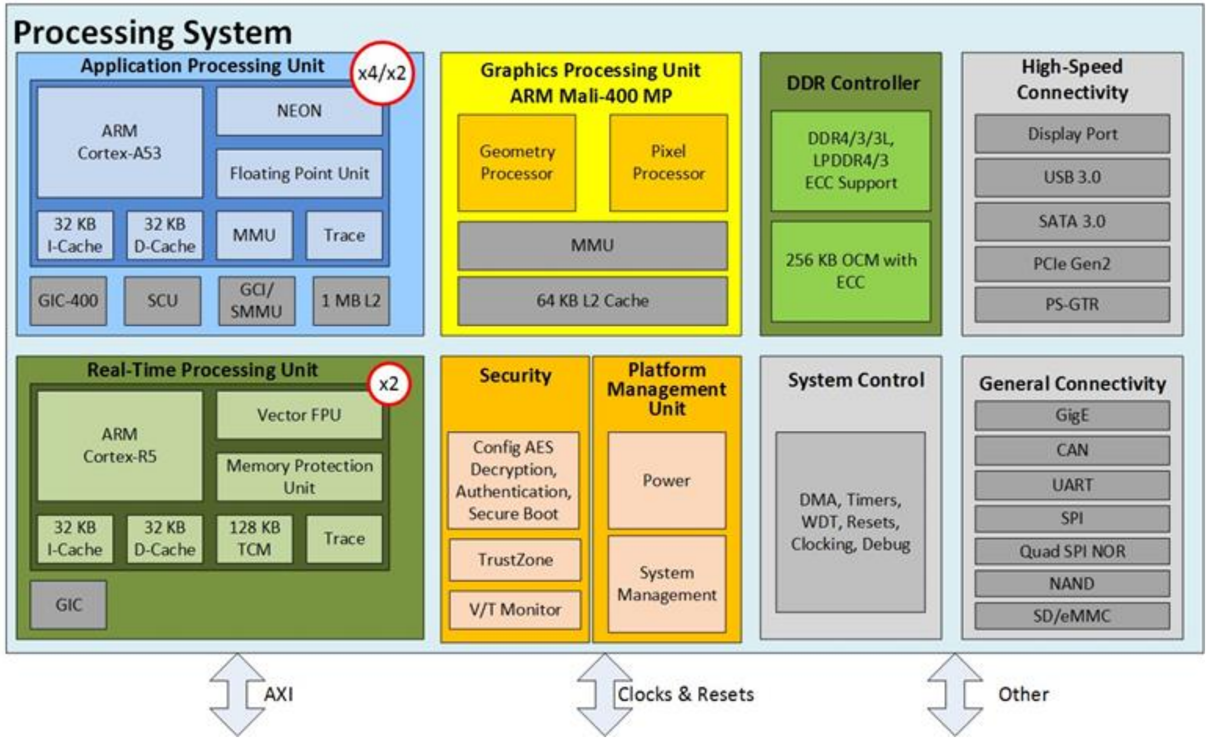
Zynq PS In Detail

- Dual-core Arm® Cortex®-A9 processors
- Memory Controller – DDRx and SMC
- Peripherals
  - Gigabit Ethernet, USB, UART, SPI, IIC, CAN, GPIO
- Secure Boot – AES, RSA and SHA



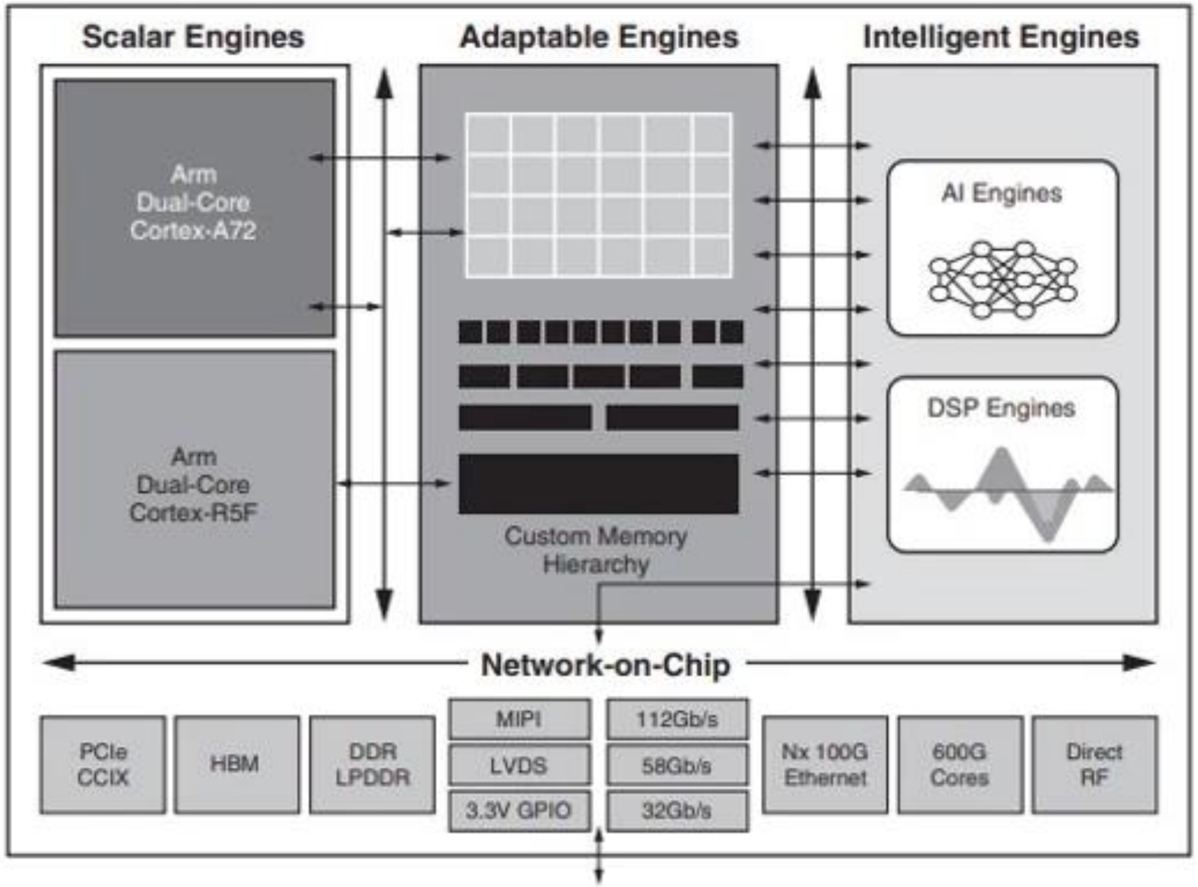
MPSoC/RFSoC PS in detail

- Quad/dual-core Arm Cortex-A53 processor cluster
- Dual-core Arm Cortex-R5 real-time processor cluster Arm Mali-400MP graphics processor
- Memory controllers (DDRx and SMC)
- Peripherals
  - High-speed peripherals: USB 3.0, SATA 3.0, PCIe Gen 2
  - technology, DisplayPort
  - Low-speed peripherals: CAN, UART, I2C
- Security, power management, safety, and reliability

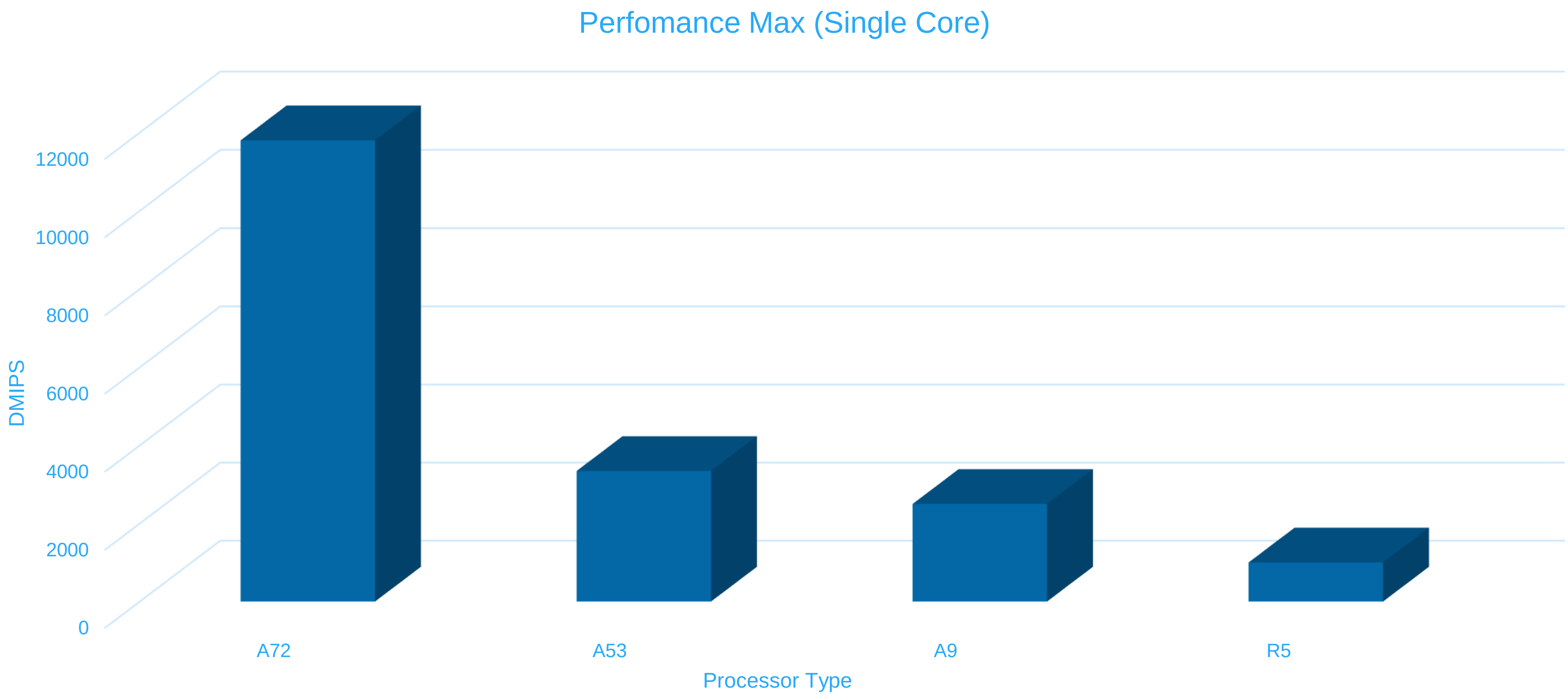


Versal

- Dual Core Arm Cortex-A72
- Dual Core Arm Cortex-R5F
- Network on Chip
    - DDR4 Controllers
- Integrated Peripherals
    - 100G Ethernet
    - GTM
    - Crypto Engine
- Integrated Hardware
    - AI Engines
    - Acceleration RAM



Hard Processors In AMD Xilinx



Which One Should I use?

Parameter	Hard Processor	Soft Processor	Comment
Performance	High	Medium to Low	
Impact on Logic Resources	Low	Medium to High	Depends on additional supporting components required
Customize Processor	Medium	High	Hard Processors have limited configurability
Security	High	Medium	Programmable Logic based soft implementations can still encrypt the bit stream.
Power Efficiency	High	Medium	
Portability	Low	High	If Open source is used can be very portable
Ease of Development	Medium	Low	Need to create the processor in the programmable logic first.

PetaLinux

History of PetaLinux

Linux was created as a free version of Unix for the x386 Intel CPU by Linus Torvalds in 1991

Linux was ported to Arm in 1994 on the Acorn processor which was not embedded

The first embedded project that Linux was ported to is unclear, it is believed to have been an x86 variant in 1997

In 2012, Xilinx acquires embedded Linux company: PetaLogix

PetaLinux first public release in 2013

Yocto build system utilized since version 2016.3

What is PetaLinux? (a set of tools)

A build system:

» A set of open-source cross-platform development command line and menu tools for use running under an x86 Linux Host OS

» Just six PetaLinux commands (with many options):

`petalinux-create`, **`petalinux-config`**, **`petalinux-build`**, **`petalinux-boot`**, **`petalinux-package`**, **`petalinux-util`**

» The build system tailors embedded Linux to run on many different AMD Xilinx platforms: Zynq MPSoC, Zynq and MicroBlaze. See: https://www.xilinx.com/support/answers/71653.html

» PetaLinux tools now use the open-source community Yocto system since v2016.3

- This is very useful for embedded Linux developers!

» Also includes an emulator (QEMU) that can emulate PetaLinux under x86 Linux

- Very useful for Kernel and some device driver development

What is PetaLinux? (what comes with it)

Source code, libraries, applications and Yocto recipes

» Mostly open-source based with a few exceptions for some hardware drivers
» PetaLinux includes and manages the Linux kernel sources and libraries
» Various hardware drivers and modules
» Traditional Linux applications and utilities

Through Yocto, PetaLinux can make use of additional embedded Linux applications beyond what AMD Xilinx supplies.

» More on that later!

A more complete list of what is included by AMD Xilinx can be found here:

https://www.xilinx.com/support/answers/71653.html

What is PetaLinux? (what goes in the firmware)

Includes Bootloaders, Trusted Platforms

» U-Boot, FSBL, ARM Trusted Firmware,

Vivado Hardware Designs

» PetaLinux device drivers work with Vivado IP blocks
» PetaLinux imports hardware designs into the PL (FPGA)

A Linux OS that runs on the AMD Xilinx Zynq MPSoC, Zynq (or MicroBlaze) CPUs

» 2018.3 uses Linux Kernel version 4.14
» For single or multiple CPUs

Linux applications, device drivers and lots of configuration files

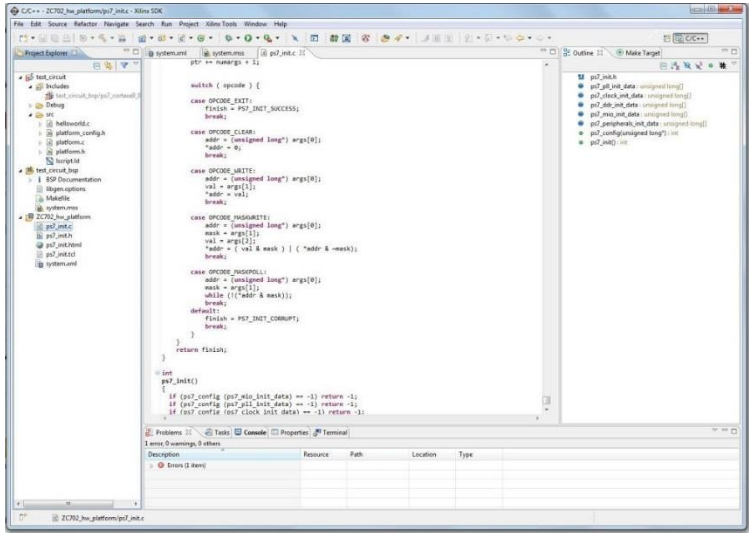
Detailed hardware and memory requirements can be found here: http://www.xilinx.com/cgi-bin/docs/rdoc?v=latest;d=ug1144-petalinux-tools-reference-guide.pdf

This is User Guide 1144: it is THE master reference book for PetaLinux!

What is PetaLinux? (part of an ecosystem)

AMD Xilinx SDK and PetaLinux work together

» SDK is a complete editing, compiling and debugging environment based on Eclipse

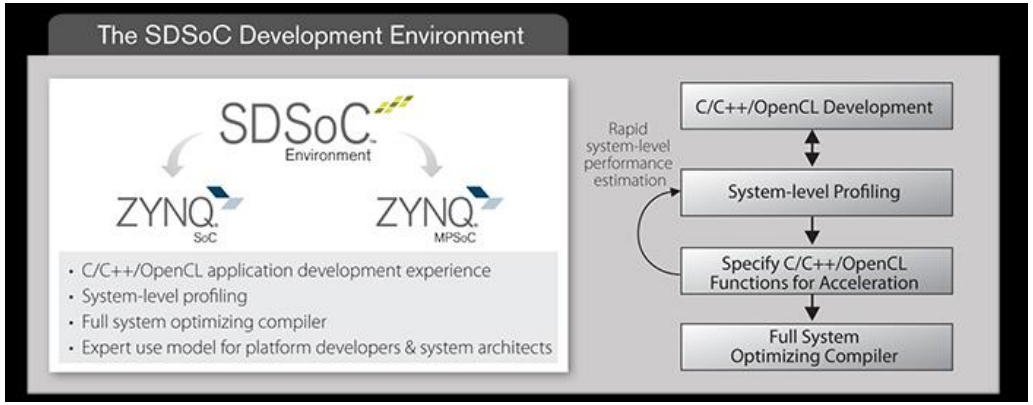


The PetaLinux tools, packaging, version management and source are all funded and supported directly by AMD Xilinx

- Vivado and PetaLinux work together

VIVADO™

- SDSoC and PetaLinux work together



PetaLinux Tools Testing, Release, and Documentation

- u-boot, Linux kernel, Xen hypervisor
  - Sources:
    - Kernel.org + AMD Xilinx patches
  - Testing:
    - AMD Xilinx silicon and AMD Xilinx reference board features
  - Documentation:
    - Source code
    - Open source communities/forums
    - Answer Records for select interim issues

- Linux file system
  - Sources:
    - Yocto community code base
  - Testing:
    - Build verification
    - Testing of AMD Xilinx-unique software and features
  - Release strategy:
    - Bug fixes with next Yocto release integration
  - Documentation:
    - Source code
    - Open source communities/forums

What is PetaLinux for Ultra96?

Linux

U-Boot

Arm Trusted Firmware

FSBL

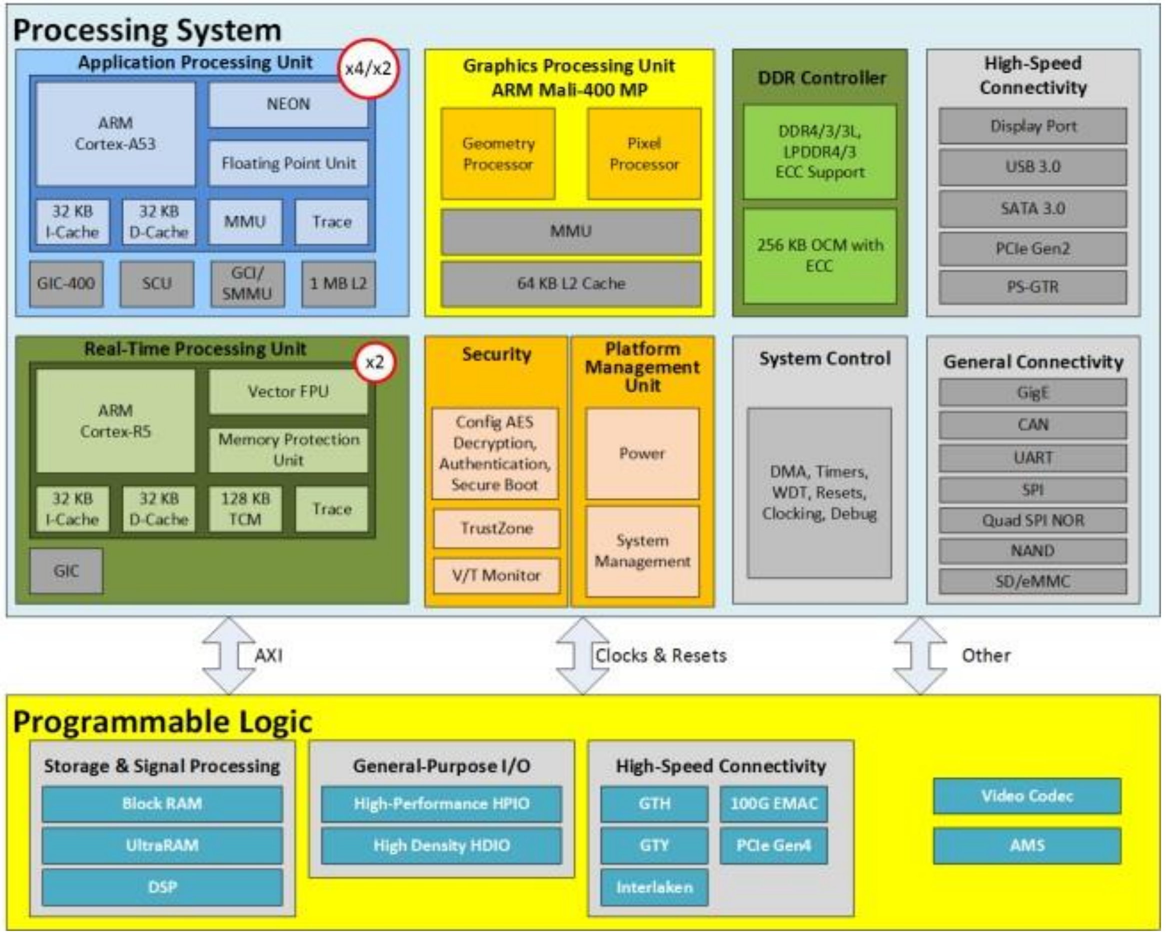
ARM Cortex-A53

Programmable Logic

I/O

GPU

RPU



Stage 0	BootROM	Detects boot mode and loads the First Stage Boot Loader
Stage 1	FSBL	Initializes processor resources so larger boot loader can be loaded
Stage 2	U-Boot	U-boot loads kernel and passes bootargs or a device tree to Linux
Operating System	Linux Kernel	Initialize system hardware and mount root file system
File System	Init Process → Login Prompt → Startup Scripts	`/sbin/init` – 1st process execution by kernel • Terminal console is usually one of first programs to run • Startup script launches application specific processes

PetaLinux Multitasking Capabilities

Embedded Linux (like Linux) is a multi-tasking multi-user operating system

» On the Ultra96 there are four APU cores

PetaLinux multitasking means:

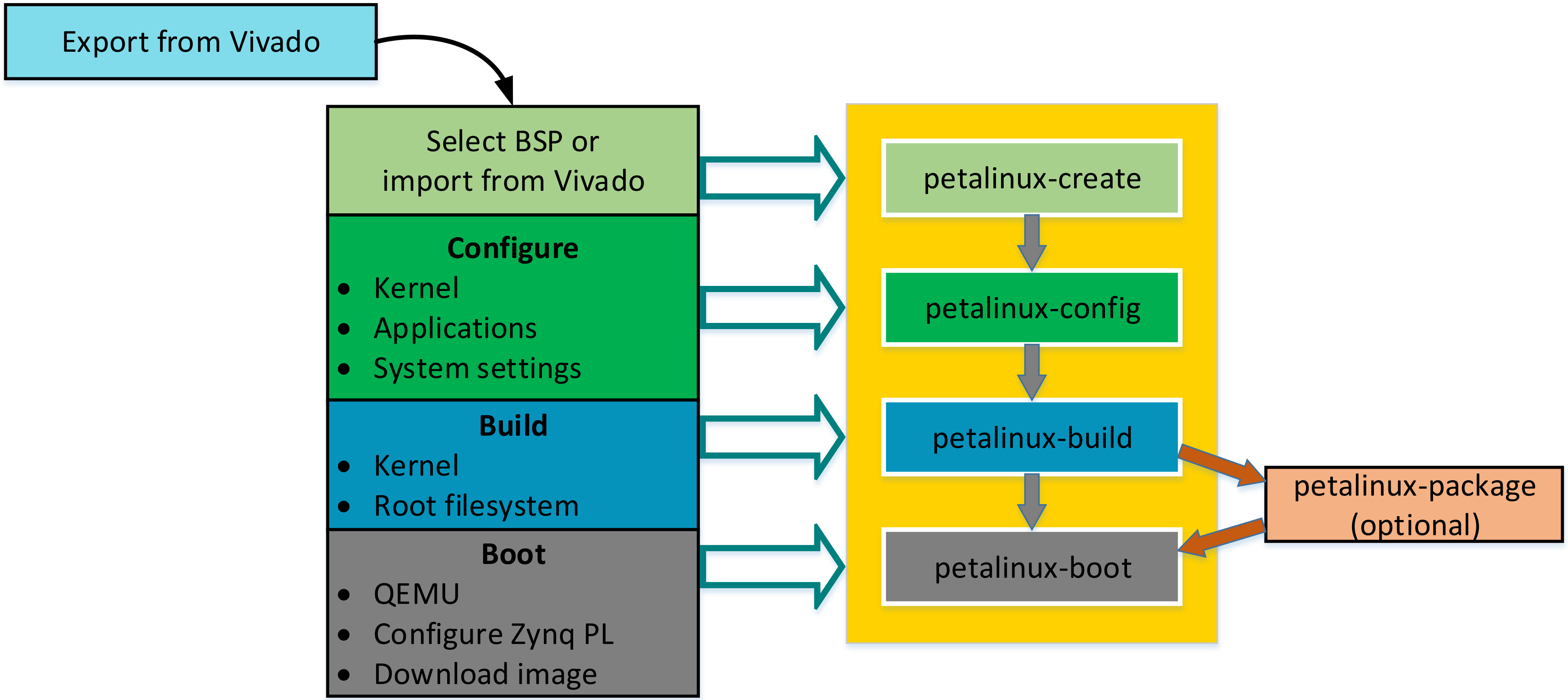
» Fast (Four 64-bit cores) but not a real-time OS (RTOS)

» If your application requires real-time, use the R5 cores. These core cannot run PetaLinux but can run FreeRTOS etc.

» Communication between the A53 (PetaLinux) and R5 (FreeRTOS etc.) can use OpenAMP

» Even though PetaLinux is not a pure RTOS it can still be used for many applications (with careful system architecture, design and expectations)

PetaLinux Tools Flow



Creating a PetaLinux Project

Select BSP or import from Vivado

petalinux-create

- » Creates a new PetaLinux project or component
- » Usage

```
petalinux-create [options] \
-t|--type <TYPE> \
-n|--name <COMPONENT_NAME>
```

Required/Optional	Available Type
`-t, --type <TYPE>`	`project` - Create a PetaLinux project `apps` - Create an application `libs` - Create a library `modules` - Create a module
`-n, --name <COMPONENT_NAME>`	Specify name for the component/project Not required if BSP is specified

Configuring a PetaLinux Project

Configure
- Kernel
- Applications
- System settings

`petalinux-config`

- » Configures the PetaLinux project or component
- » Usage

```
petalinux-config [options] {--component <COMPONENT> |\
--get-hw-description[=SRC] |\
--searchpath <--ACTION> [VALUE]}
```

Required/Optional	Available Type
`--get-hw-description[=SRC]`	Get hardware description Look in the location of Vivado SDK export
`-p, --project <PROJECT>`	Path to PetaLinux project

--get-hw-description this is the same HDF file we used in the Avnet TTC HW and SW courses.

Building a PetaLinux Project

Build
- Kernel
- Root filesystem

`petalinux-build`

» Builds the PetaLinux project or component

» Usage

`petalinux-build [options]`

Required/Optional	Available Type
`-p, --project <PROJECT>`	Path to PetaLinux project

`-c, --component <COMPONENT>`	Specify the component
	If specified, e.g.:
	`-c rootfs`
	`-c rootfs/myapp`

What is The Yocto Project?

The Yocto Project is an open source collaboration project that provides templates, tools and methods to help you create custom Linux-based systems for embedded products regardless of the hardware architecture. It was founded in 2010 as a collaboration among many hardware manufacturers, open-source operating systems vendors, and electronics companies to bring some order to the chaos of embedded Linux development.

It's a framework, not a distribution

» Allows users to create ***their own*** distributions
» Contains reference distributions

Commonly used by several organizations:

» Wind River Linux
» Mentor Embedded Linux
» Just about everyone else…

Uses a build tool called **bitbake**

PetaLinux 2018.3 abstracts most of the bitbake details, commands, etc.

Why Yocto?

The challenge…

- Linux is becoming increasingly popular for embedded systems
- Non-commercial and commercial embedded Linux has many distros

Leads to the result…

- Developers spend lots of time porting or making build systems
- Leaves less time/money to develop interesting software features
- The industry needs a common build system and core technology
- Industry leaders joined together in 2010 to form the Yocto Project

With these benefits…

- Less time spent on things which don't add value (build system, core Linux components)
- Linux grows more in embedded
- More time spent on product differentiation and interesting software

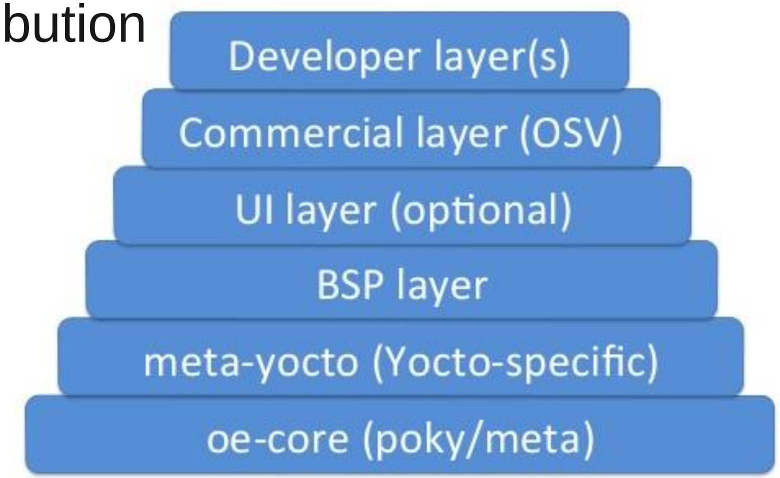
Yocto Layers, Recipes, and Bitbake

Yocto builds and assembles Linux images based on ***layers***

» Layers are named **meta-*something***, such as:
- **meta-*yocto***, **meta-*xilinx***, **meta-*xilinx-tools***, etc.

Layers consist of ***recipes***

» Recipes describe individual components of a target Linux distribution
» Can be written to do all sorts of things:
- Apply source code patches to the kernel
- Copy files to a target root filesystem
- Build / install user applications
- *Build and install kernel drivers for wireless modules*



PetaLinux 2018.3 specifics

» Bitbake is abstracted using familiar commands, e.g. petalinux-build, etc.
» Layer set aside for user customization: <project>/project-spec/meta-user

Source Fetching

Recipes call out location of all sources

» Can be local or remote (internet)

(look for SRC_URI in *.bb files, more about this in Lab 6)

Bitbake can get sources from git, svn, bzr, from tarballs, and many more

Versions of packages can be fixed or updated automatically

PetaLinux and Yocto

PetaLinux supports and is supported by the Yocto ecosystem

- » PetaLinux tools have the ability to integrate Yocto-generated RPM feeds into the native configured Linux filesystem
  - Allows designs to fully leverage in-house and Yocto-community software

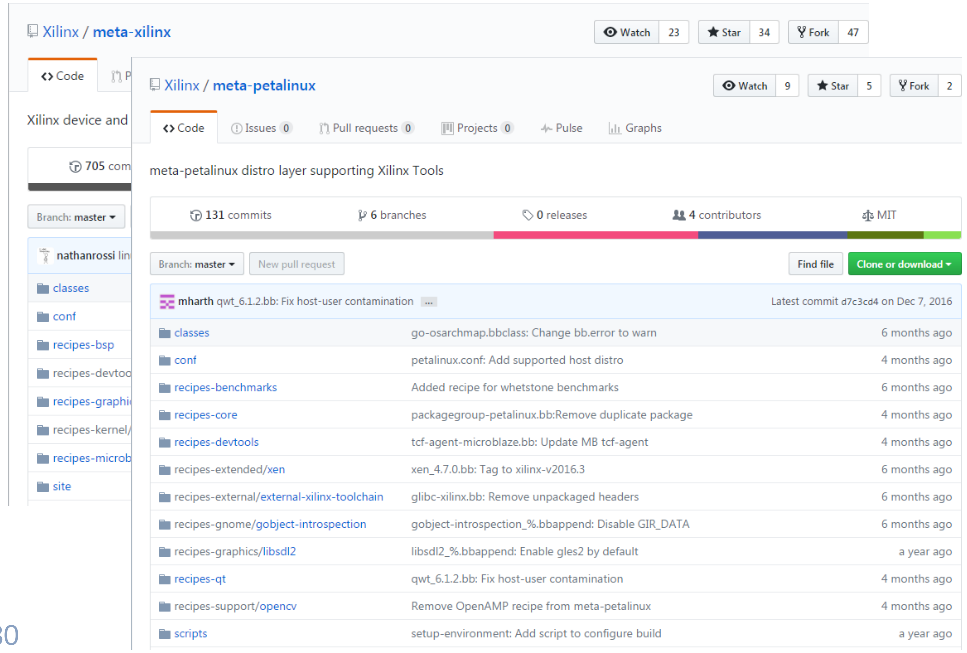
- » Yocto build system can generate the complete PetaLinux filesystem
  - Ensures that all Yocto-built software is library-compatible with AMD Xilinx-provided software
  - Allows comprehensive revision and source control

- » AMD Xilinx GIT provides Yocto recipes for all PetaLinux components
  - Arm Trusted Firmware
  - U-Boot
  - Kernel
  - Root filesystem

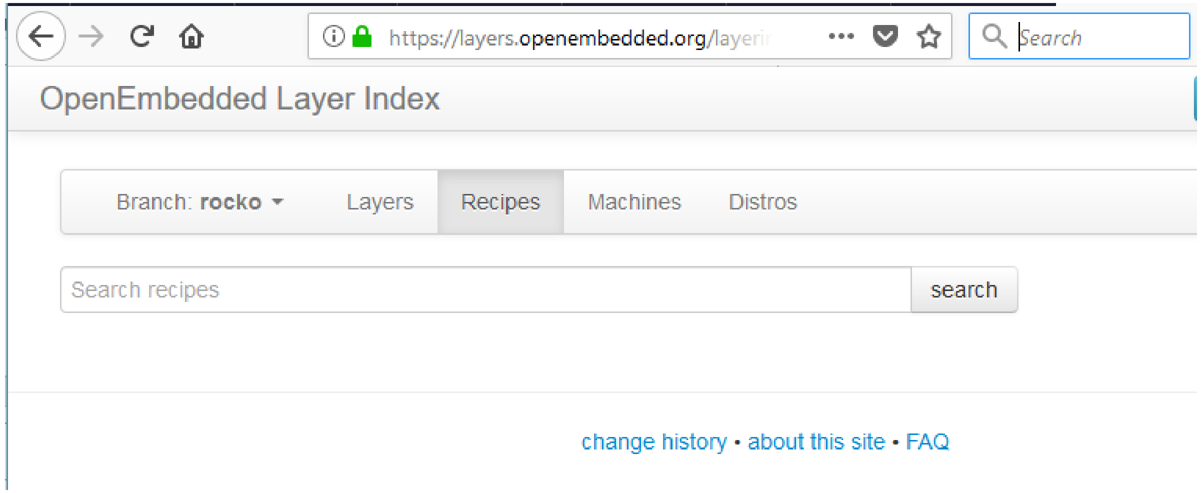
PetaLinux Yocto Recipes

AMD Xilinx Yocto recipes available now!

» https://github.com/xilinx
  - meta-xilinx
  - meta-petalinux

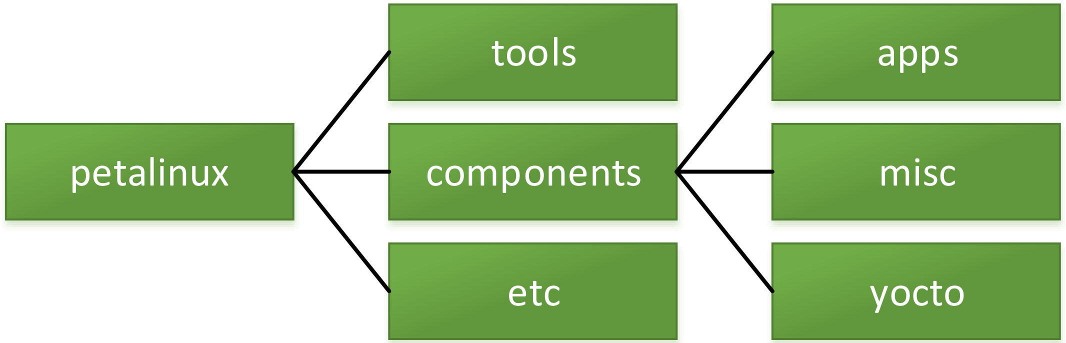


- 3rd party recipes (use at own risk)
  – PetaLinux 2018.3 aka rocko
  https://layers.openembedded.org/layerindex/branch/rocko/recipes/



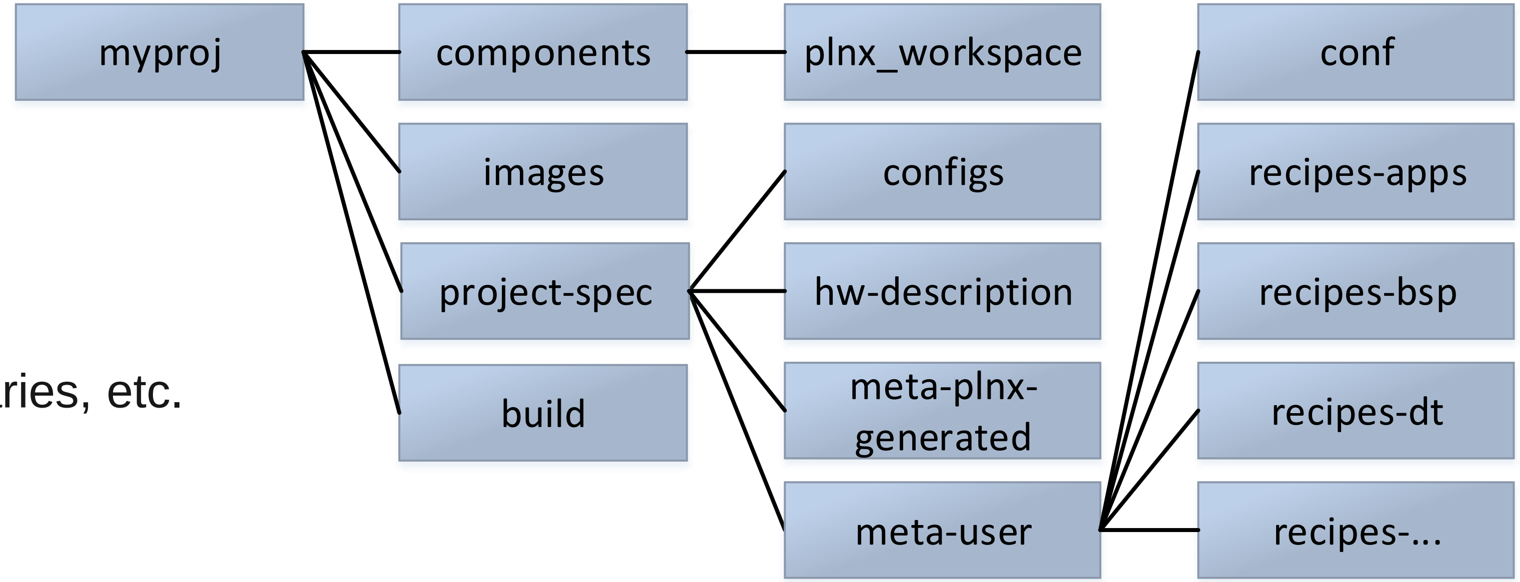
PetaLinux Install and Project Taxonomy

Tools and default source components in install area



Per-project directory

- » Configuration files
- » Custom recipes
  - Applications, scripts, libraries, etc.
- » Build products



PetaLinux Project Example

```
.
├── conf
├── recipes-apps
│   ├── gpio-demo
│   │   └── files
│   └── peekpoke
│       └── files
├── recipes-bsp
│   ├── custom
│   │   └── files
│   ├── device-tree
│   │   └── files
│   ├── minized-firmware
│   │   └── files
│   ├── minized-misc
│   │   └── files
│   └── u-boot
│       └── files
├── recipes-connectivity
│   ├── bluez5
│   │   └── bluez5
│   └── wpa-supplicant
│       └── files
├── recipes-core
│   └── images
├── recipes-kernel
│   └── linux
│       └── linux-xlnx
├── recipes-modules
│   └── lis2ds
│       └── files
└── recipes-multimedia
    ├── pulseaudio
    │   └── files
    └── pulseaudio-service
        └── files
```

meta-user Sub-folder Name	Purpose
recipes-apps	Custom user applications
recipes-bsp	Customizations to the u-boot bootloader, device tree, hardware platform and user file additions to the root file system
recipes-connectivity	Networking protocols and support files
recipes-core	Configuration files to tell PetaLinux what recipes to add to the system
recipes-kernel	Kernel patches to correct, customize or enhance baseline kernel functionality
recipes-modules	Custom device drivers
recipes-multimedia	Custom application and support files for multimedia functionality

Yocto References and Resources

Online trainings

» https://www.doulos.com/content/events/LinuxZynqYocto.php
» https://www.yoctoproject.org/tools-resources/videos/getting-started-yocto-project-new-developer-screencast-tutorial

References

» https://wiki.yoctoproject.org/wiki/FAQ
» https://www.yoctoproject.org/
» http://www.wiki.xilinx.com/Yocto

Documentation

» https://www.yoctoproject.org/documentation
» https://www.xilinx.com/support/documentation/sw_manuals/xilinx2018_3/ug1144-petalinux-tools-reference-guide.pdf

PetaLinux Platform Configuration

In a Zynq MPSoC device, every PS and PL configuration can be different

- Address map, memory size
- IRQ structures, CPU, and device configuration

How does the kernel know what the hardware looks like?

- Hard coded values or #defines? Definitely not!
- A special data structure called the Device Tree

Device Tree

- A Zynq MPSoC Linux system can be viewed as a tree structure
  - CPU is the root node
  - Buses are branches
  - Devices are nodes
  - Memory, peripherals
- Each node has properties
  - Address range
  - Device properties
  - Interrupt connection
  - Etc…

Device Tree

DTS file

- » Device Tree Source
- » Textual description of system device tree

DTB

- » Device Tree Blob
- » Compiled, binary representation of the DTS

DTC

- » Device Tree Compiler
- » Provided in PetaLinux installation
- » Converts the DTS to a DTB
- » Can de-compile a DTB back to a DTS

Device Tree Structure

A DTB is provided to the kernel at boot time

- » Kernel startup code parses the tree
- » Instantiates and initializes kernel appropriately
  - System memory location/size
  - Device types, address mappings, configuration options

Kernel infrastructure binds drivers to devices

- » Matched against the "compatible" property in the device node
- » Device driver declares known "compatible" devices

Greatly simplifies the problem of device ↔ driver binding

Device Tree Source Example (shown on right)

```
&sdhci0 {
    status = "okay";
    bus-width= <4>;
    xlnx,has-cd = <0x0>;
    xlnx,has-power = <0x0>;
    xlnx,has-wp = <0x0>;
    non-removeable;
    wifi-host; /* adding for compatibility w/ bcmdhd driver */
};

/* Define the external I2C devices and their names and addresses */
&axi_iic_0 {
    #address-cells = <0x1>;
    #size-cells = <0x0>;
    compatible = "xlnx,xps-iic-2.00.a";
    i2c5@1d {
        #address-cells = <0x1>;
        #size-cells = <0x0>;
        compatible = "stm,lis2ds12";
        reg = <0x1d>;
    };
};

/* QSPI addresses are defined with petalinux-config, but here they are overwritten so that one can
program the flash internally */
&qspi {
    #address-cells = <1>;
    #size-cells = <0>;
    flash0: flash@0 {
        compatible = "micron,m25p80";
        reg = <0x0>;
        #address-cells = <1>;
        #size-cells = <1>;
        spi-max-frequency = <50000000>;
        partition@0x00000000 {
            label = "boot";
            reg = <0x00000000 0x00ff0000>;
        };
    };
};
```

Device Tree Creation

PetaLinux includes an automated device tree generator

- » Generates device tree based on hardware design
- » All IP cores and properties exported in DTS

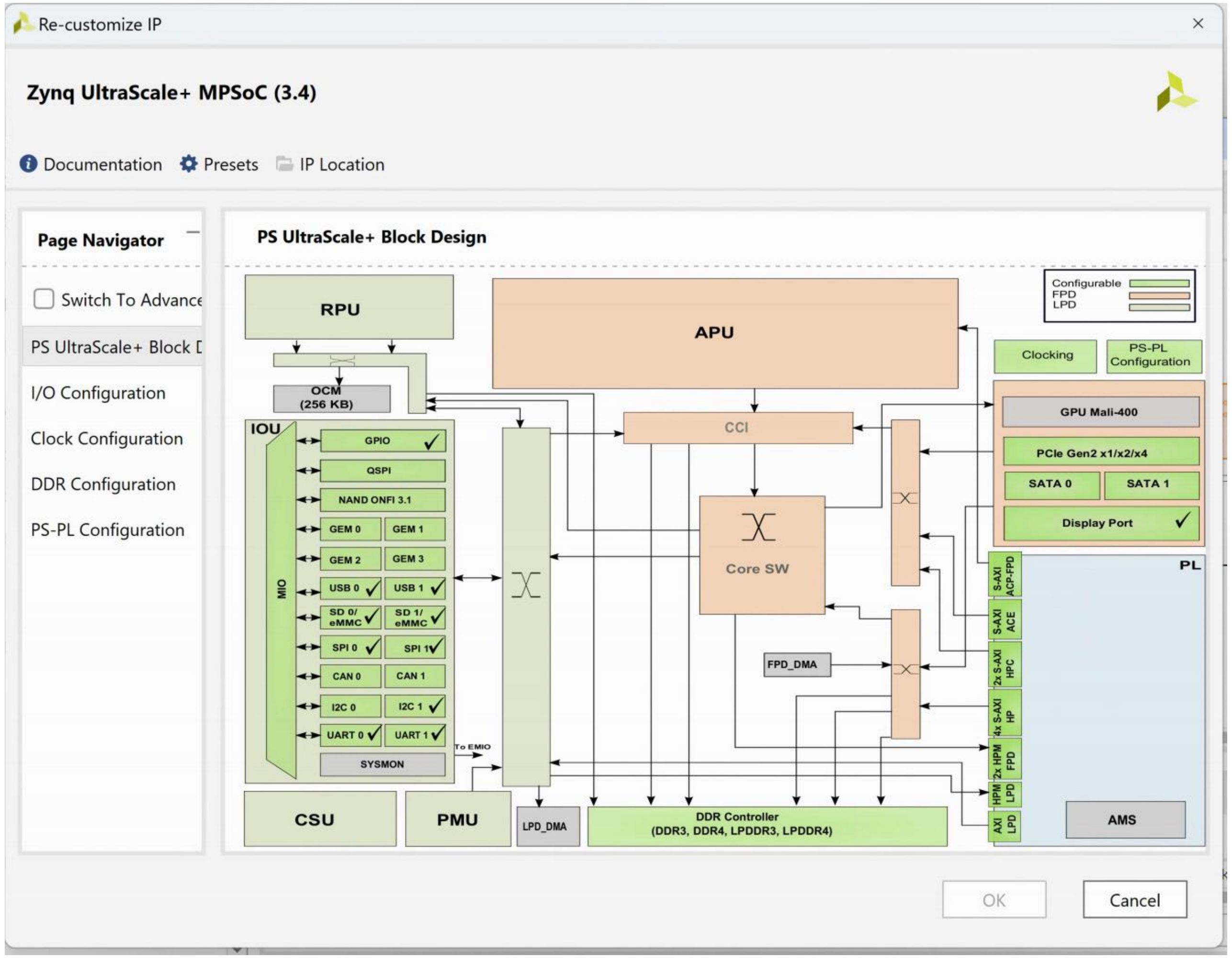
Kernel build process compiles DTS → DTB and links into kernel image

- » Auto-generated DTS files in <petalinux_project>/components/plnx_workspace/device-tree-generation/
  - Do not edit! Auto-generated on import of hardware platform from Vivado
- » User-editable system-top.dts DTS file in <petalinux_project>/project-spec/meta-user/recipes-dt/device-tree/files/
  - User overwrite of default parameters and addition of new parameters
  - Changes to the system-top.dts will not be overwritten by subsequent petalinux-config

What does the Device Tree Know?

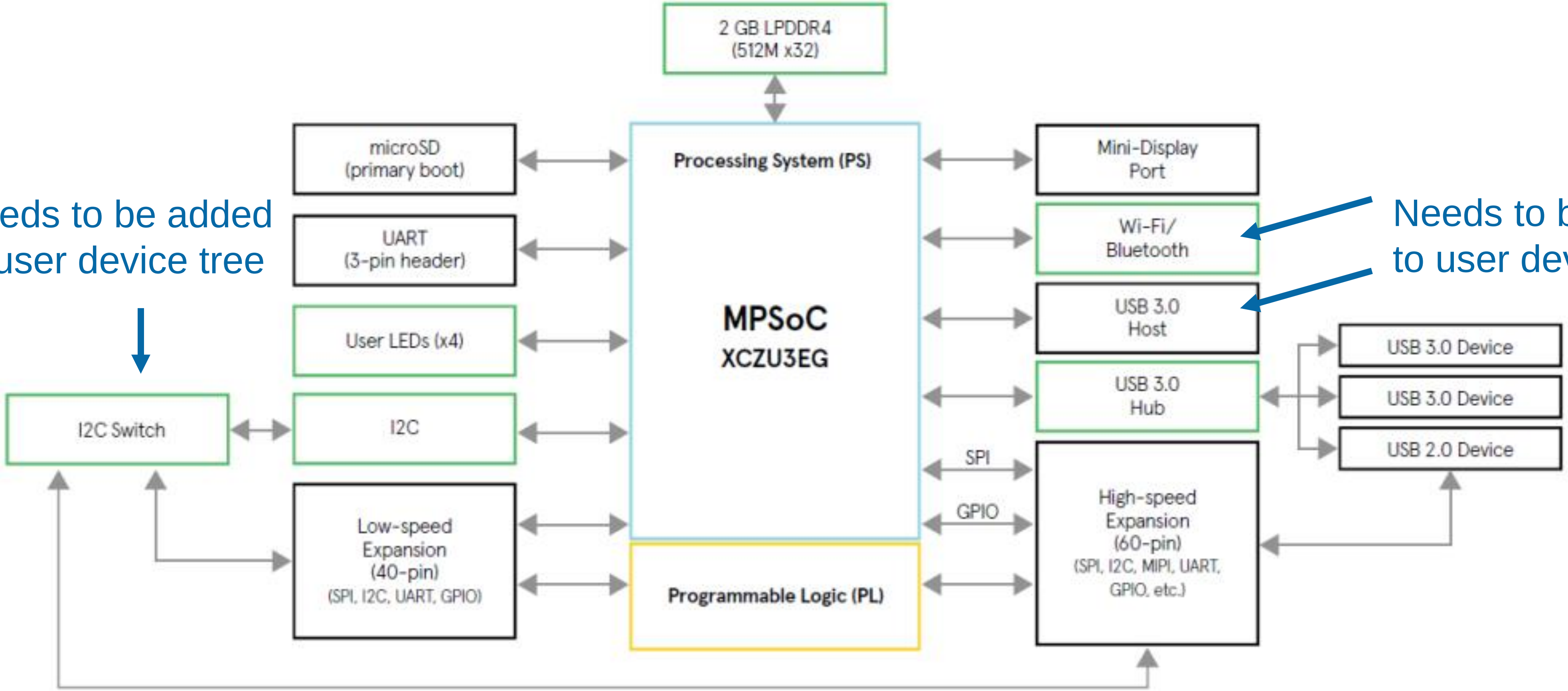
Device Tree created is limited to the interface definitions known by Vivado.

It cannot know about system level interfaces e.g., PHYs or I2C devices.



Device Tree Creation

Needs to be added to user device tree



Needs to be added to user device tree

Role of the BSP

- Setting up the device tree for a board is important, can take considerable time.
- It is unlikely there will only ever be one project created
- Board Support package provide portable way of configuring Petalinux projects for the board with all the interfaces and applications correctly set up
- Single petalinux command can be used to create the BSP

Example in Meta Avnet

Avnet / meta-avnet Public

Watch 13 | Fork 13 | Star 8

Code | Issues 3 | Pull requests 3 | Actions | Projects | Security | Insights

master | meta-avnet / recipes-bsp / device-tree / files / ultra96v2 / system-bsp.dtsi | Go to file | ...

tnizan-witekio meta-avnet: change meta-avnet priority ... | Latest commit 03f276d on Aug 16, 2020 | History

1 contributor

233 lines (208 sloc) | 5.27 KB | Raw | Blame

```
/include/ "system-conf.dtsi"
/include/ "openamp.dtsi"
/ {
        chosen {
                bootargs = "earlycon console=ttyPS0,115200 clk_ignore_unused root=/dev/mmcblk0p2 rw rootwait cma=512M";
        };

   sdio_pwrseq: sdio_pwrseq {  /* This needs to be able to manipulate the chip_en and the resetn properly */
      compatible = "mmc-pwrseq-simple";
      /* post-power-on-delay-ms = <10>; */
      reset-gpios = <&gpio 7 1>; // MIO[7] for WILC3000 RESETN, Active low
      /* reset-gpios = <&gpio 78 1>;  // Some prototype boards only! device pin A3 for RESETN, Active low */
      chip_en-gpios = <&gpio 8 1>;
   };
```

PetaLinux Device Tree Inspection

By inspecting /proc/device-tree you can look at what PetaLinux has detected in the FPGA:

```
root@plnx_arm:/proc/device-tree# ls -l
total 0
-r--r--r--  1 root root  4 Sep 18 11:13 #address-cells
-r--r--r--  1 root root  4 Sep 18 11:13 #size-cells
drwxr-xr-x  2 root root  0 Sep 18 11:13 aliases
drwxr-xr-x 33 root root  0 Sep 18 11:13 amba
drwxr-xr-x  9 root root  0 Sep 18 11:13 amba_pl
drwxr-xr-x  2 root root  0 Sep 18 11:13 chosen
-r--r--r--  1 root root 15 Sep 18 11:13 compatible
drwxr-xr-x  3 root root  0 Sep 18 11:13 cpus
drwxr-xr-x  2 root root  0 Sep 18 11:13 fixedregulator
drwxr-xr-x  2 root root  0 Sep 18 11:13 fpga-full
drwxr-xr-x  2 root root  0 Sep 18 11:13 memory
-r--r--r--  1 root root  1 Sep 18 11:13 name
drwxr-xr-x  2 root root  0 Sep 18 11:13 pmu@f8891000
drwxr-xr-x  2 root root  0 Sep 18 11:13 usb_phy@0
drwxr-xr-x  2 root root  0 Sep 18 11:13 wlreg-on
```

Inspect attributes with: hexdump –C ‘/proc/device-tree/<attribute name>’

```
root@plnx_arm:~# hexdump -C '/proc/device-tree/amba_pl/i2c@41610000/compatible'
00000000  78 6c 6e 78 2c 78 70 73  2d 69 69 63 2d 32 2e 30  |xlnx,xps-iic-2.0|
00000010  30 2e 61 00                                       |0.a.|
00000014
root@plnx_arm:~#
```

Key Concept: Memory Management Unit

User applications access program and data from virtualized (pseudo) addresses

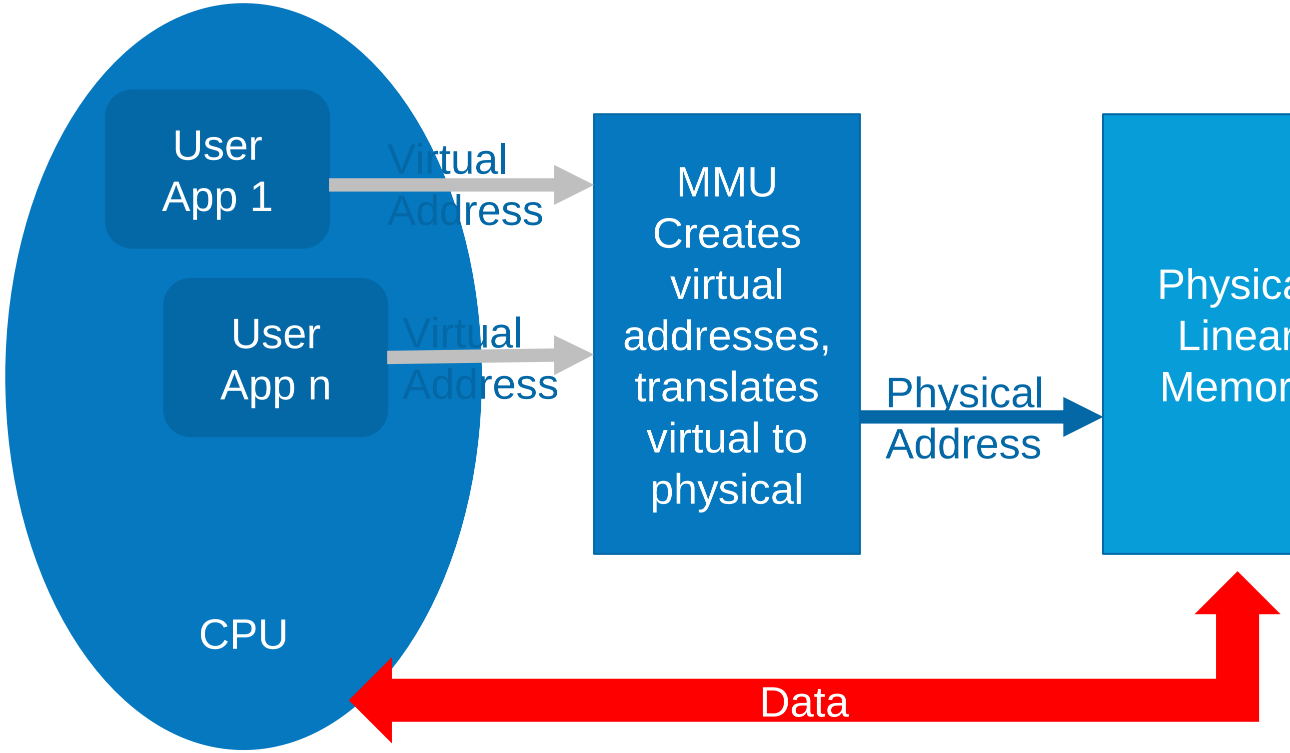
The MMU unit of the Zynq MPSoC is the translator from virtual to physical addresses

Each user program runs as if it is the only program on the CPU

Each user program has it's own unique memory mapped address space

Simplified

Operation:



Example /dev/i2c

- This is the general approach from C code for using a device on the I^2C bus
- Ultra96 has 1 I^2C bus which is connected to a I^2C switch which contains several sub buses. These include Two I^2C buses for each of the high and low speed expansion ports, Power Management bus and Current Monitor
- Example code below (for reference only)

```
int num_read;
Char read_bytes[2];
int file = open("/dev/i2c-2", O_RDWR); // Get a file handle to the I2C bus
ioctl(file, I2C_SLAVE, 0x40);  // Set the I2C address to 0x40
write(file, "\0x1\0x2",2);  // write 2-byte values 1,2
num_read = read(file,read_bytes,2); // Attempt to read back up to 2 bytes
close(file);
```

Customize PetaLinux to Add New Components

Components can be:

- » Software applications
- » Kernel modules (device drivers)
- » Shell scripts
- » Kernel patches

The PetaLinux commands like petalinux-create perform a lot of hidden tasks

```
$ petalinux-create -t apps --template c++ --name  lsm6dsl-sensor --enable
```

- » Creates a new application named '**`lsm6dsl-sensor`**'
- » Creates a new C++ source file that can be used as a starting point for development
- » Adds the new application to the root filesystem configuration menu and enables it
  - Subsequent runs of petalinux-build will automatically build and add the app to the rootfs
- » Creates a new bitbake recipe and Makefile

New component additions placed in special folder in the PetaLinux project

- » Can be found in project-spec/meta-user/recipes-<apps | kernel | modules>

PetaLinux and Yocto Bitbake

PetaLinux commands abstract the details of the bitbake commands

- Creation of new software applications, kernel modules (drivers), scripts, etc.
- Add new bitbake recipes
- Additions to configuration menus
- Build, clean, config
- Installation onto the root file system

Easy edits to makefile and recipe as application complexity grows

- Add source files, etc.

PetaLinux command to build the root file system becomes bitbake command…

```
$ petalinux-build -c rootfs

$ bitbake petalinux-user-image -c do_rootfs
```

What is in a Bitbake Recipe?

Descriptive information about the package

The version of the recipe

Existing Dependencies

Where the source code resides

Whether the source code requires any patches

How to compile the source code

Where on the target machine to install the package being compiled

Simple Bitbake Recipe

Open

```
#
# This file is the lsm6dsl-sensor recipe.
#

SUMMARY = "Simple lsm6dsl-sensor application"
SECTION = "PETALINUX/apps"
LICENSE = "MIT"
LIC_FILES_CHKSUM = "file://${COMMON_LICENSE_DIR}/MIT;md5=0835ade698e0bcf8506ecda2f7b4f302"

SRC_URI = "file://main.c \
           file://Makefile \
                  "

S = "${WORKDIR}"

FILES_${PN} += "/home/root/*"

do_compile() {
             oe_runmake
}

do_install() {
             install -d ${D}${bindir}
             install -m 0755 lsm6dsl-sensor ${D}/home/root/
}
```

Location and MD5 checksum of the license file for the application

Location of the source file(s) and Makefile

Recipe instructions

Specify where the application should be installed

ADIUVO

ENGINEERING AND TRAINING, LTD.

www.adiuvoengineering.com

adam@adiuvoengineering.com