Запускаем Embedded Linux на Hard и Soft CPU Xilinx Zynq

Панкратов Павел, Булина Яна

Павел Панкратов

Ведущий инженер-программист

- Более 10 лет опыта коммерческой Embedded разработки
- Участник ряда международных проектов в сферах Telecom и Automotive

Яна Булина

Инженер по разработке ПО искусственного интеллекта

- 3 года в AI разработке
- Знает какие IP-блоки пригодятся для инференса вашей нейросети

Постановка задачи

Что мы хотели сделать?

- Параллельный запуск двух ОС на FPGA с процессорной подсистемой
- Проект для повышения квалификации сотрудников отдела
- Освоение системного подхода к разработке

Почему это интересно?

- Охватывает весь цикл разработки начиная с уровня аппаратного обеспечения и заканчивая запуском пользовательского приложения
- Мало доступной информации о решении подобных задач на FPGA с процессорной подсистемой

Содержание

Часть 1: проект программируемой логики (Яна Булина)

- Определим отличия в подходах для FPGA со встроенным Hard CPU и без
- Рассмотрим основные компоненты аппаратного обеспечения

Часть 2: сборка Embedded Linux (Павел Панкратов)

- Поговорим о необходимых составляющих Embedded Linux
- Соберем ОС под две архитектуры
- Подготовим загрузочный носитель

Часть 3: запуск и верификация проекта (Павел Панкратов)

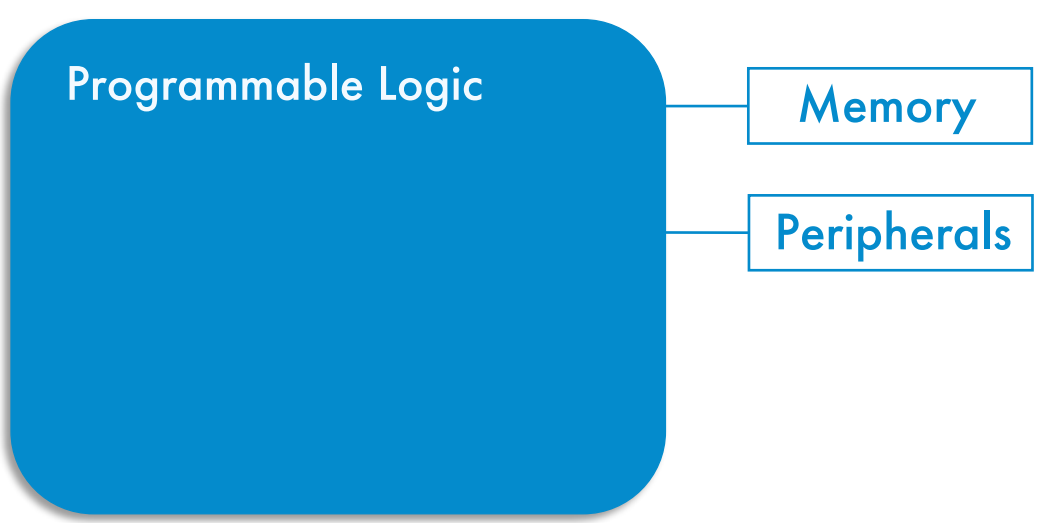
- Разгадаем загадку: “Сколько нужно загрузчиков для запуска двух ОС?”
- Убедимся в работоспособности проекта

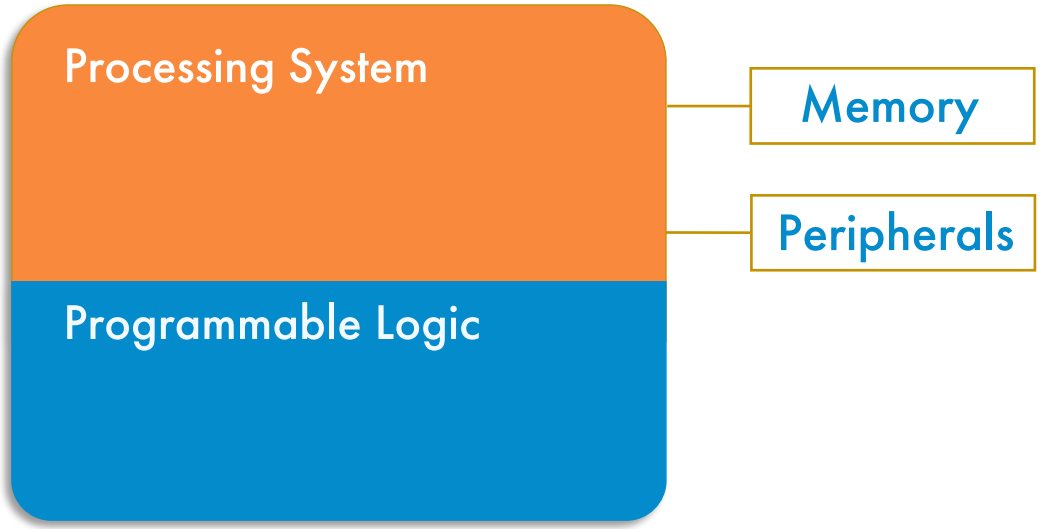
Часть 1: проект программируемой логики

Часть 2: сборка Embedded Linux

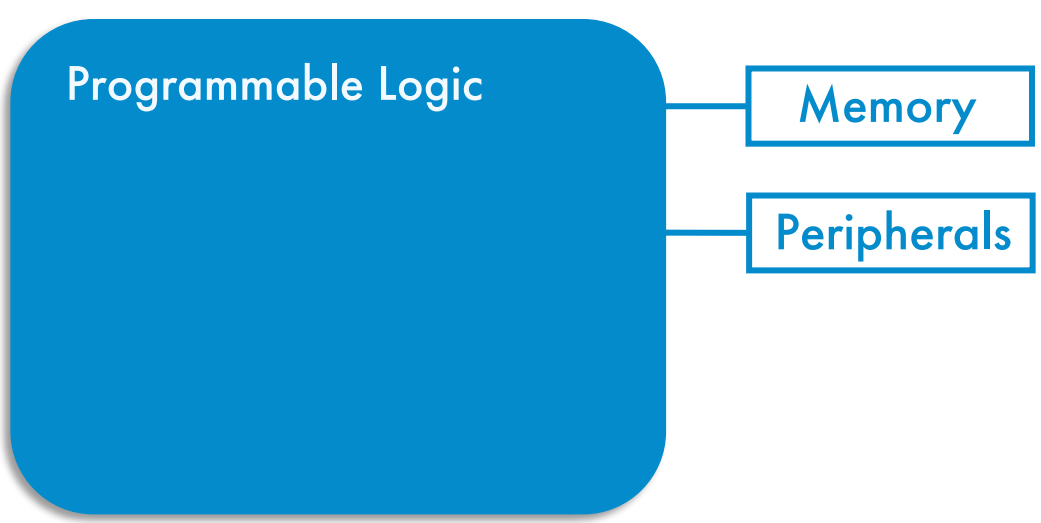
Часть 3: запуск и верификация проекта

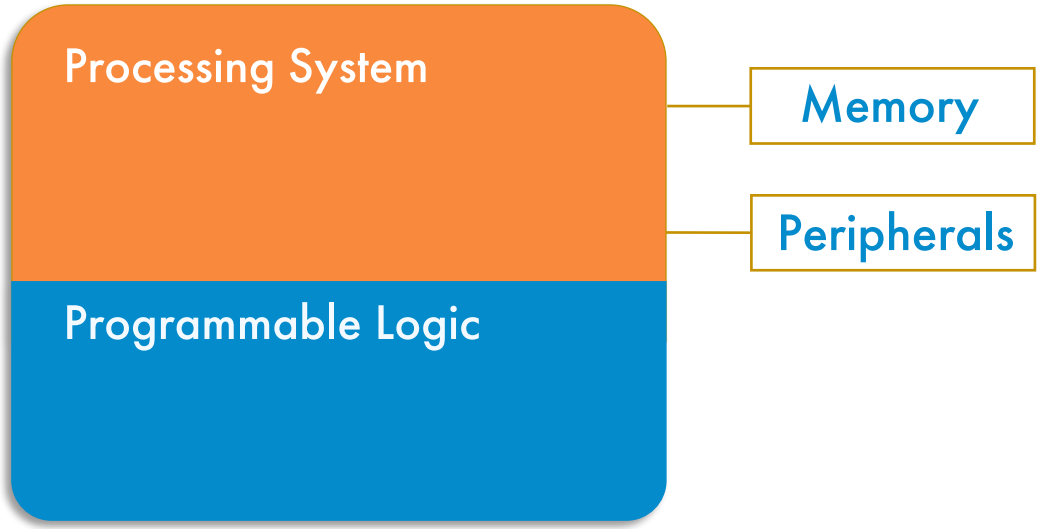
Различия в архитектуре FPGA и SoC



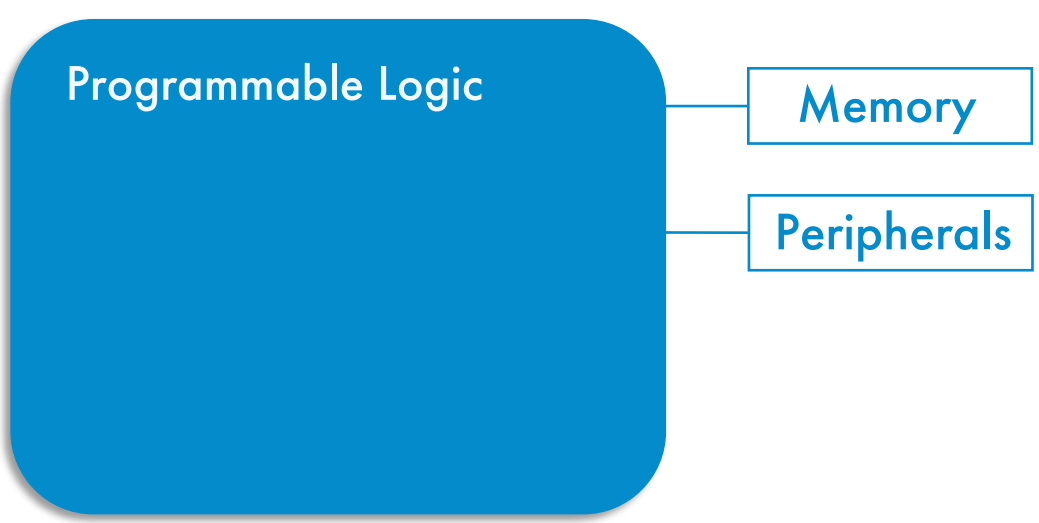


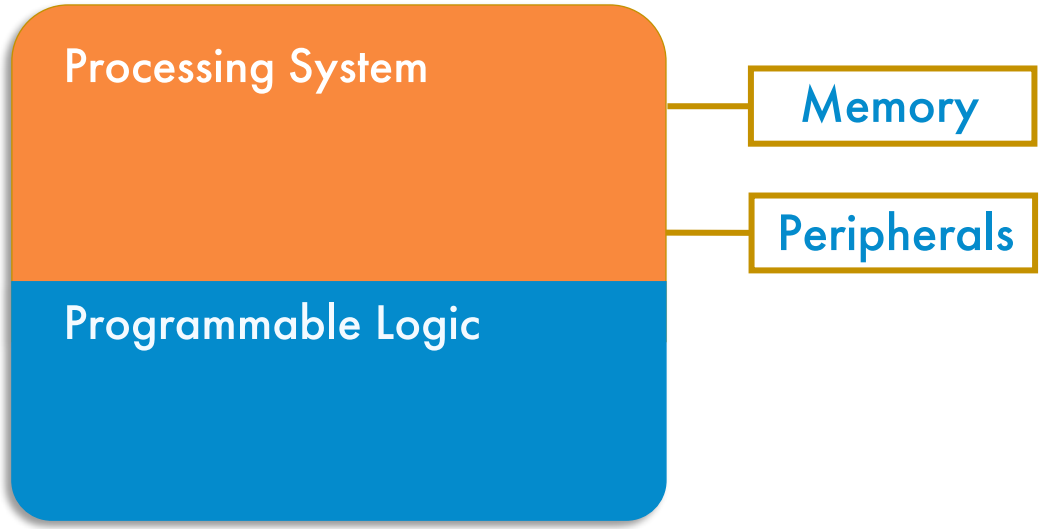
Различия в архитектуре FPGA и SoC



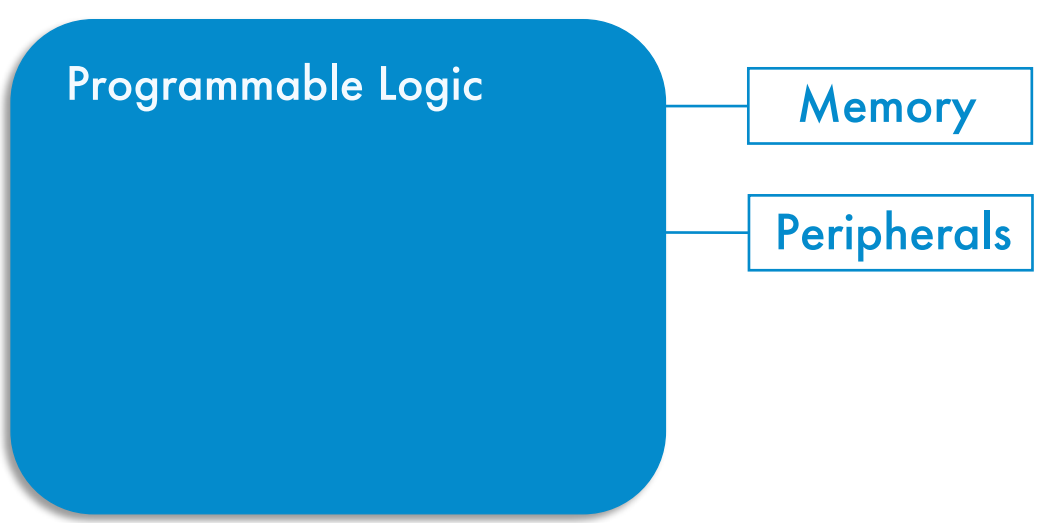


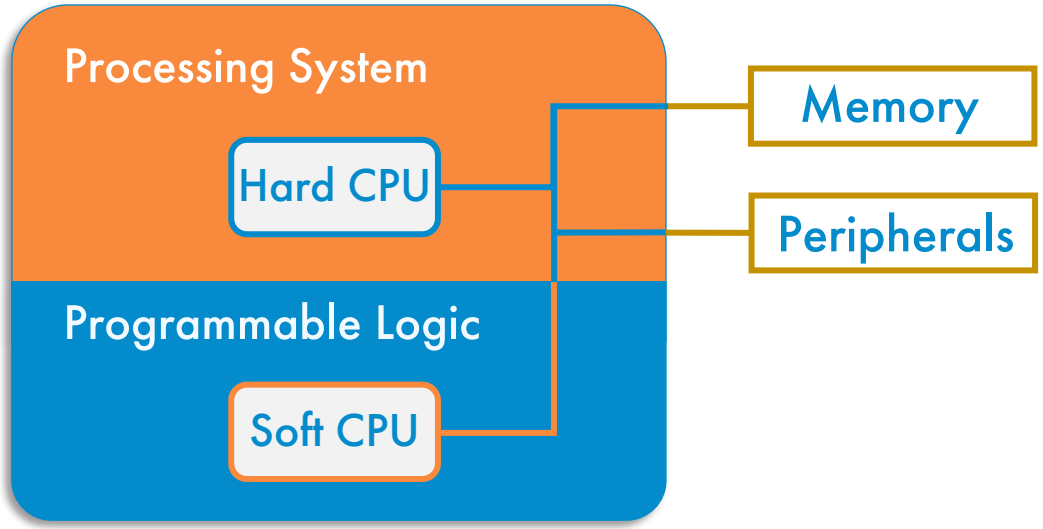
Различия в архитектуре FPGA и SoC



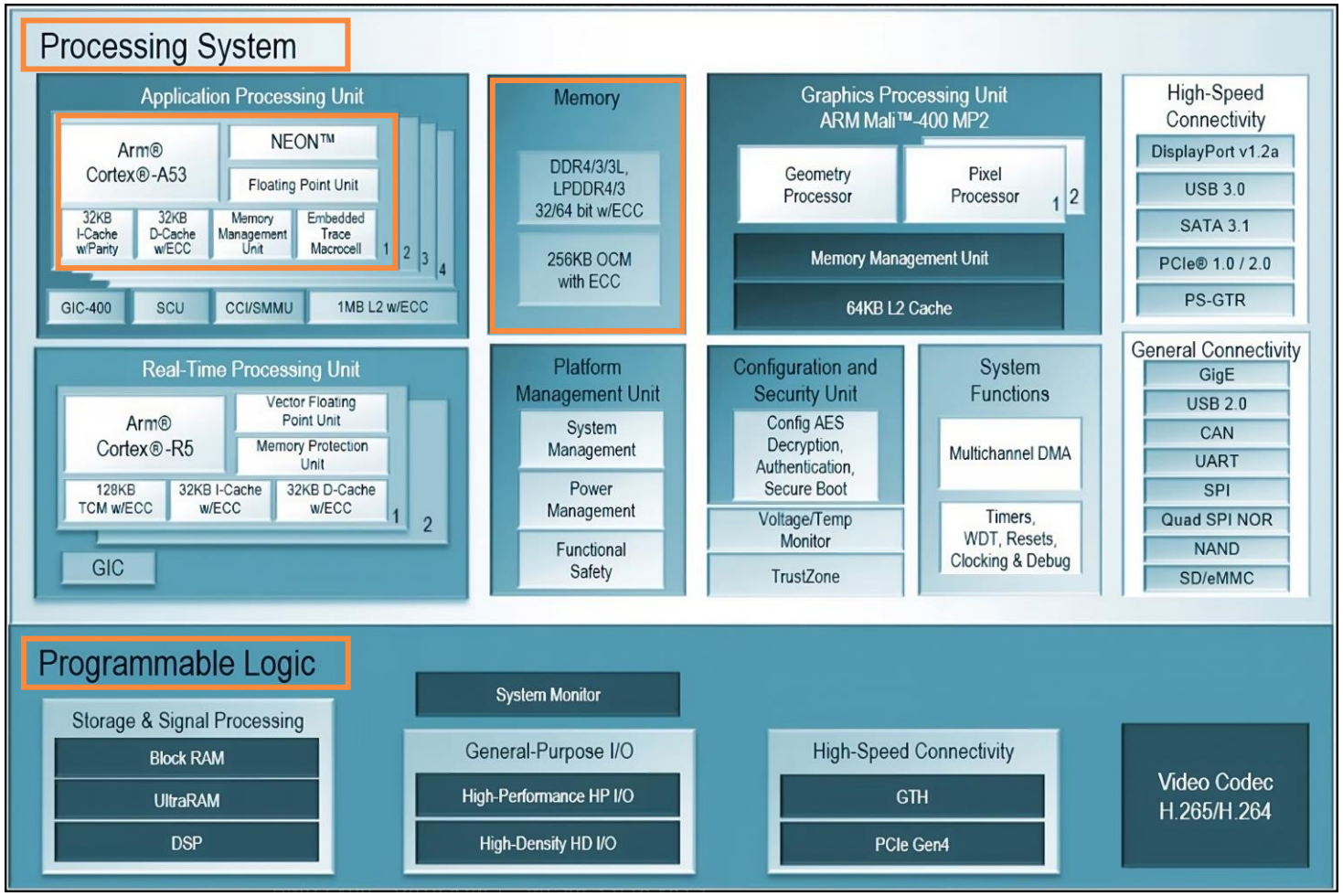


Различия в архитектуре FPGA и SoC

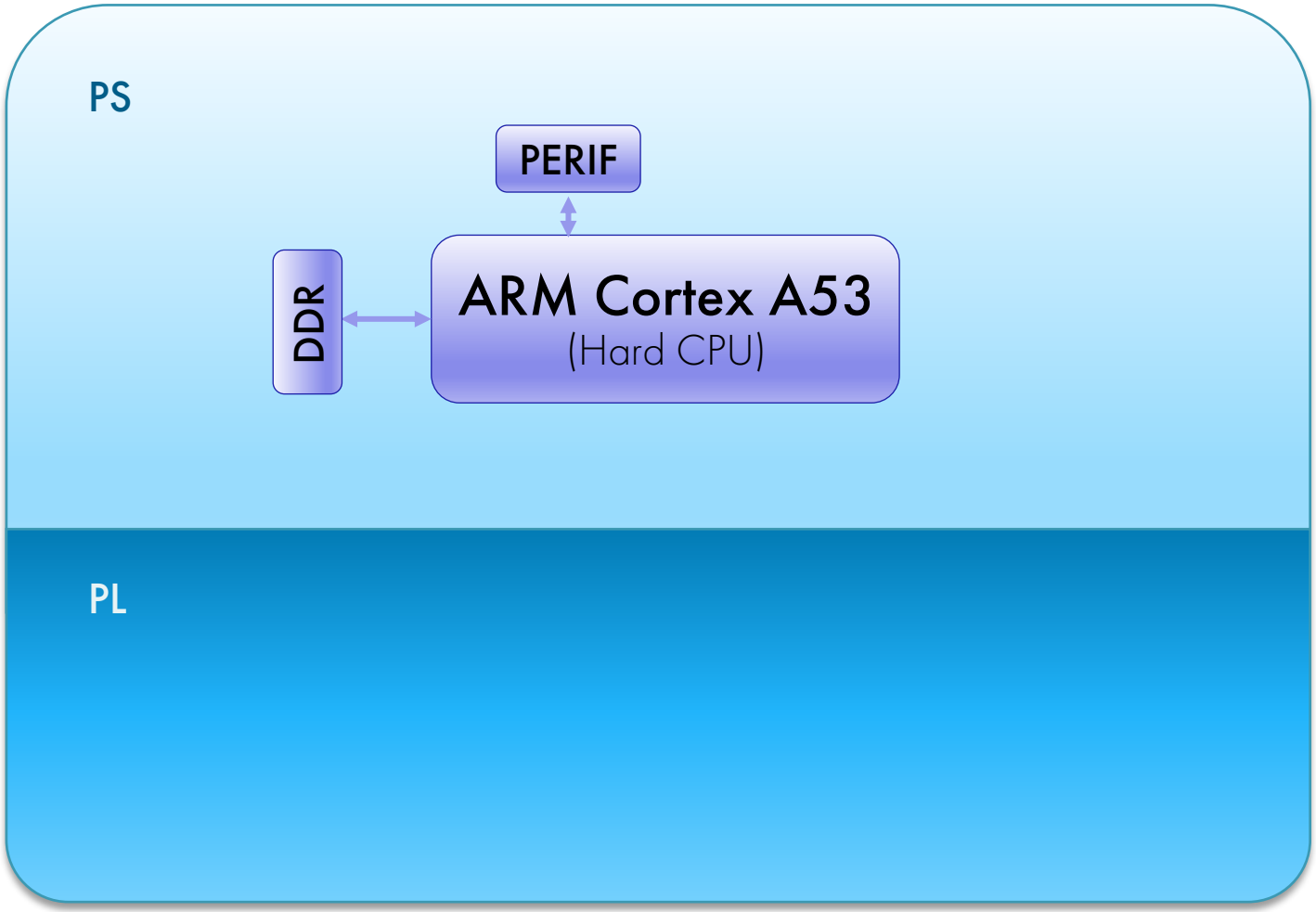




MYIR Tech FZ5BOX



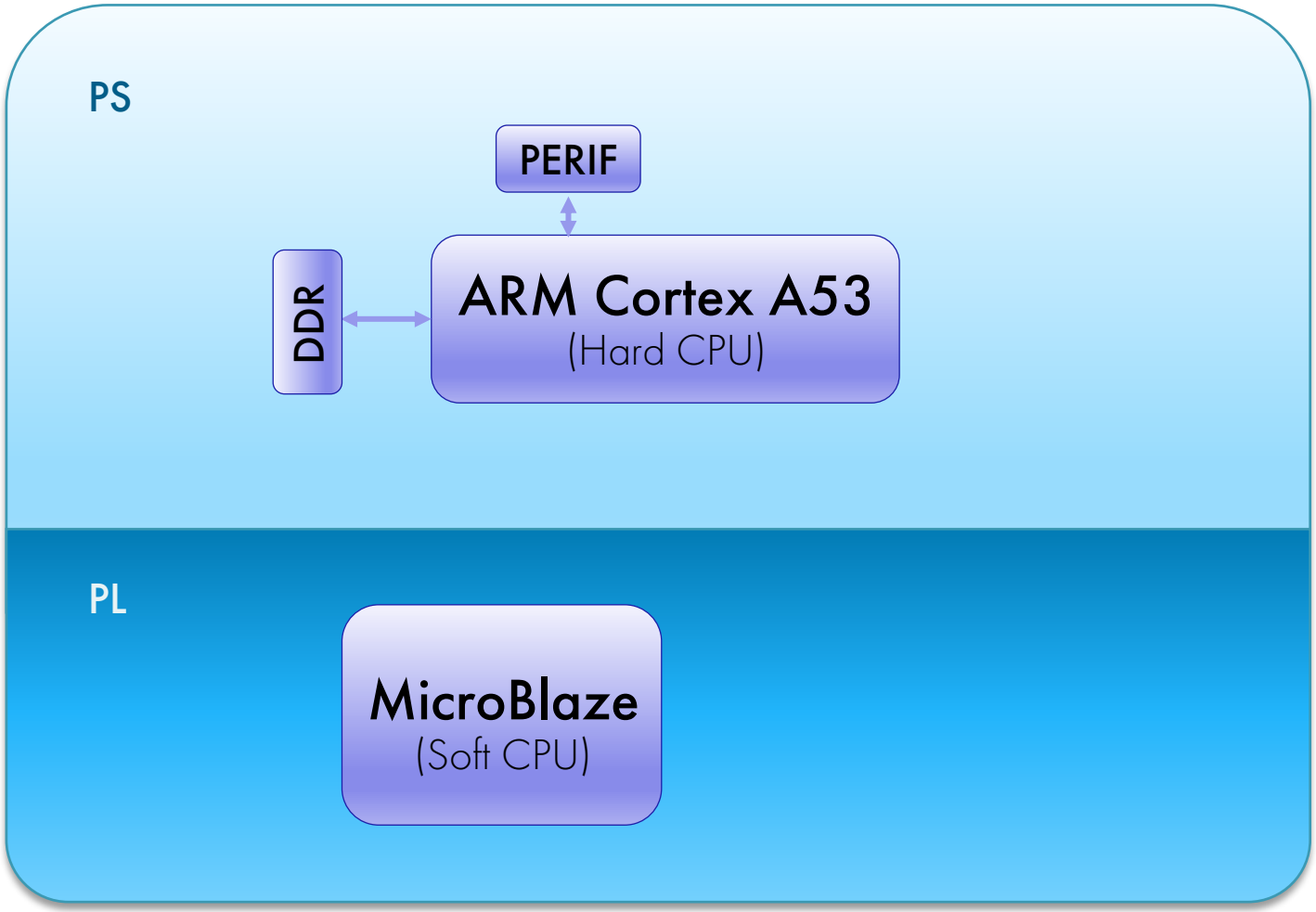
Архитектура проекта



Что нам нужно для запуска Linux:

- ✓ CPU с MMU
- ✓ Оперативная память
- ✓ Контроллер прерываний
- ✓ Системный таймер

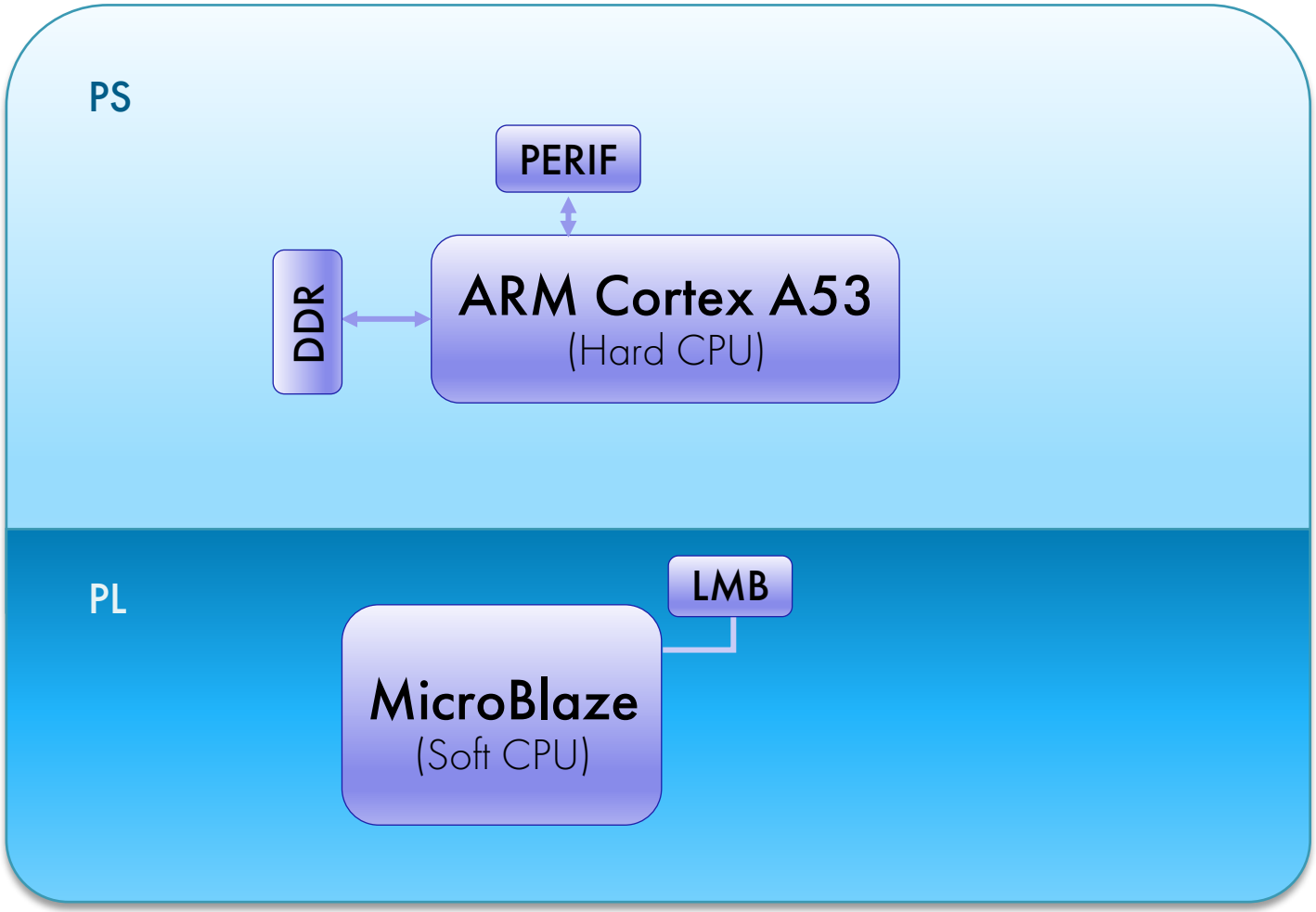
Архитектура проекта



Что нам нужно для запуска Linux:

- ✓ CPU с MMU
- ✓ Оперативная память
- ✓ Контроллер прерываний
- ✓ Системный таймер

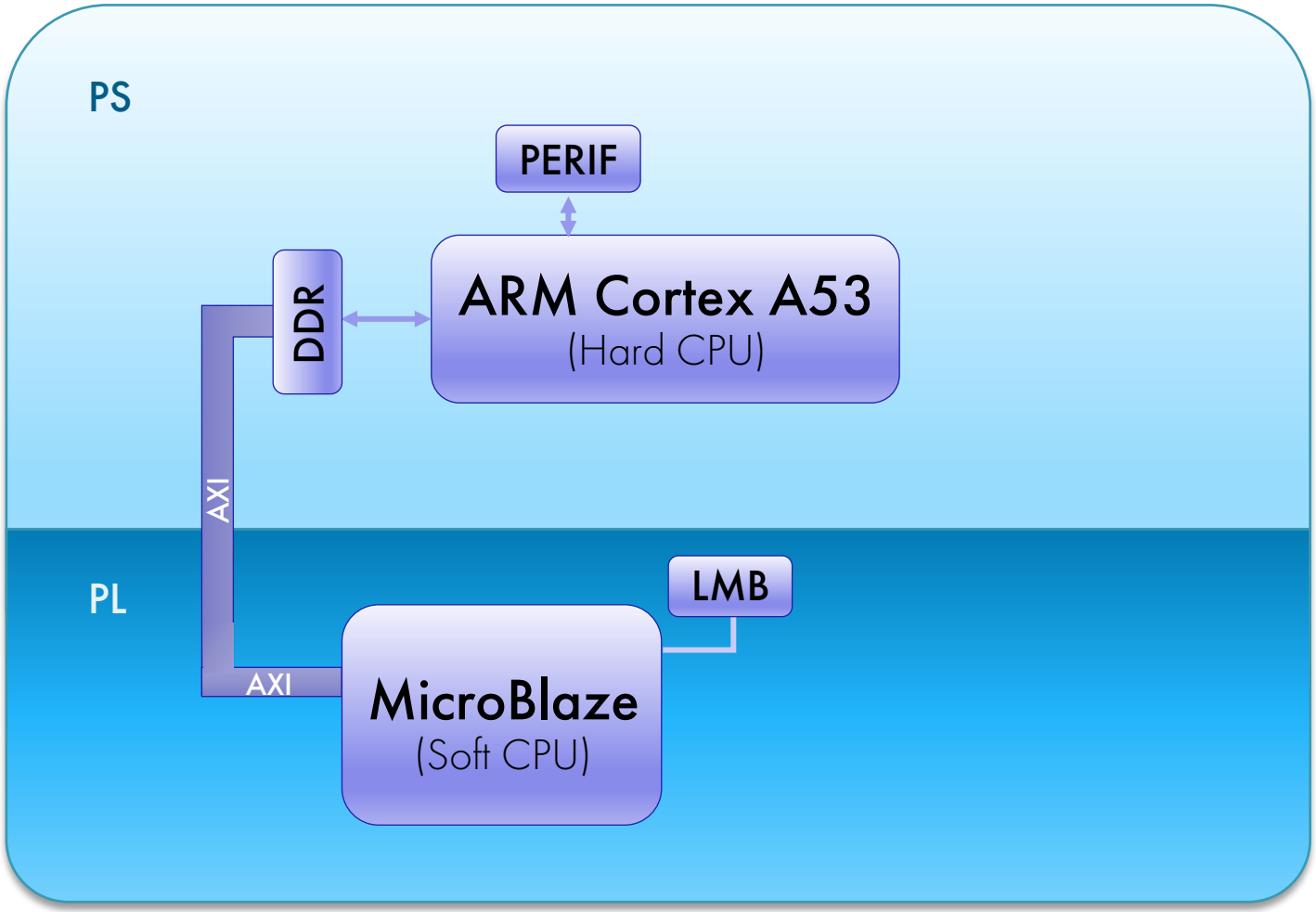
Архитектура проекта



Что нам нужно для запуска Linux:

- ✓ CPU с MMU
- ✓ **Оперативная память**
- ✓ Контроллер прерываний
- ✓ Системный таймер

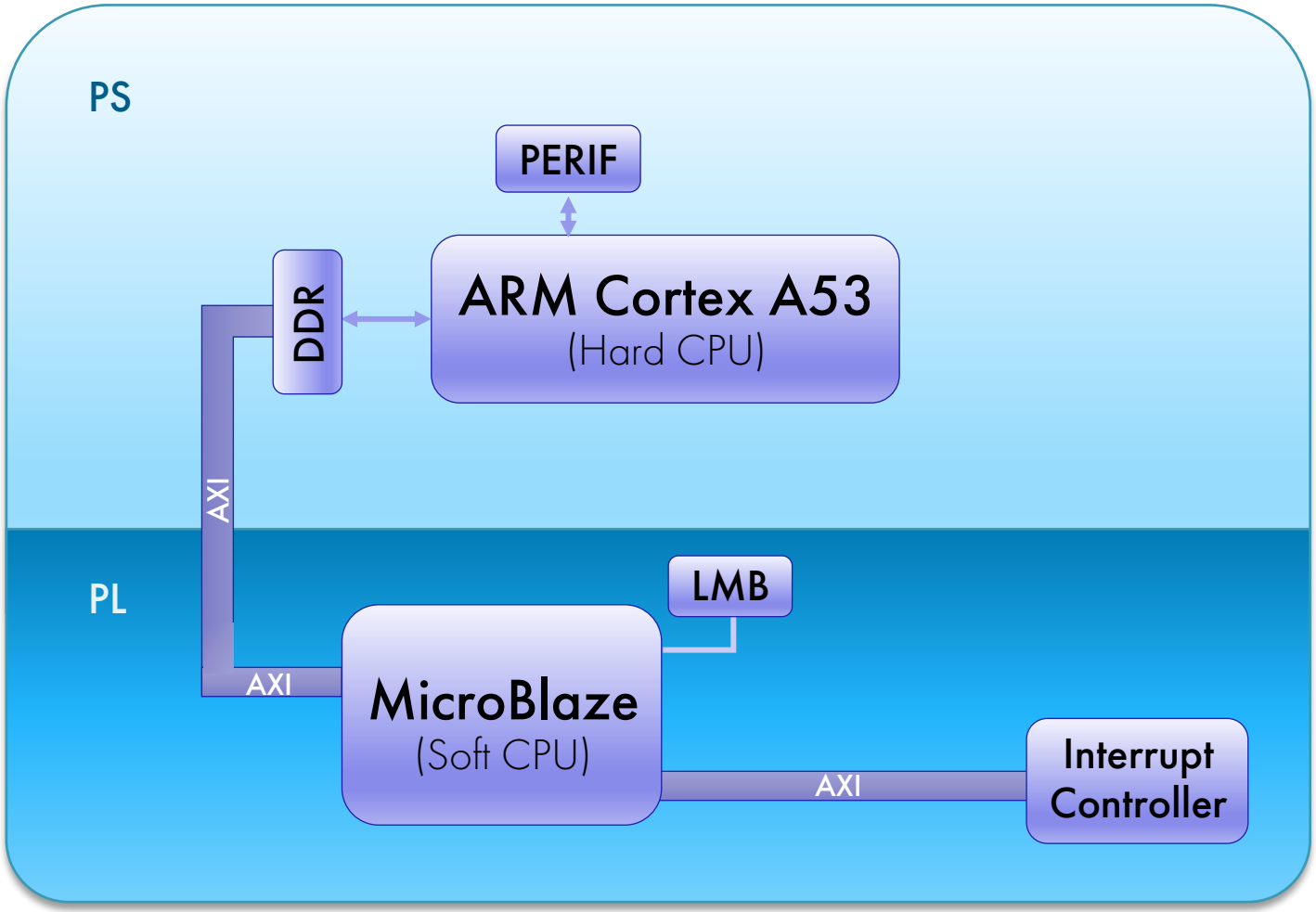
Архитектура проекта



Что нам нужно для запуска Linux:

- ✓ CPU с MMU
- ✓ **Оперативная память**
- ✓ Контроллер прерываний
- ✓ Системный таймер

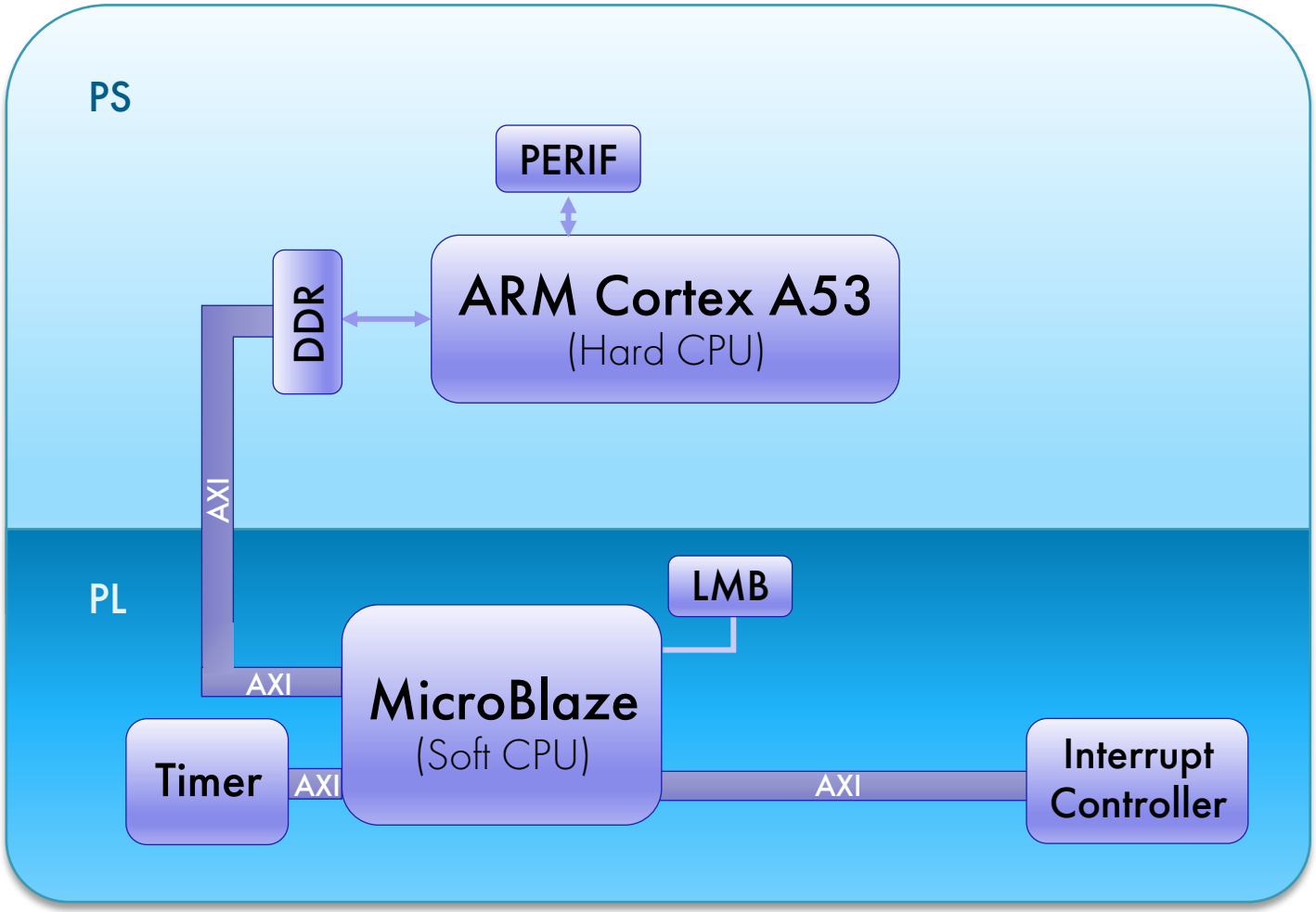
Архитектура проекта



Что нам нужно для запуска Linux:

- ✓ CPU с MMU
- ✓ Оперативная память
- ✓ **Контроллер прерываний**
- ✓ Системный таймер

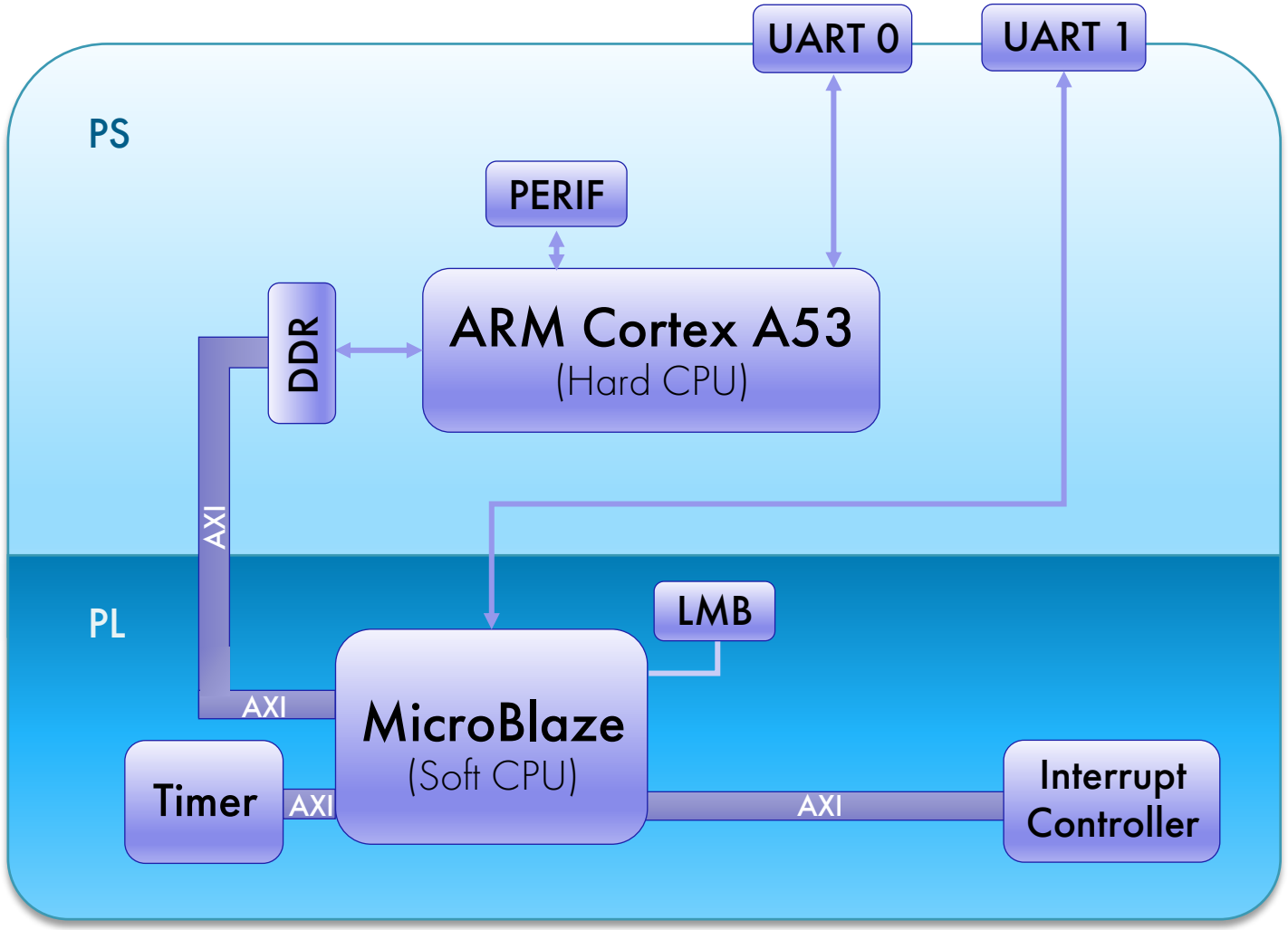
Архитектура проекта



Что нам нужно для запуска Linux:

- ✓ CPU с MMU
- ✓ Оперативная память
- ✓ Контроллер прерываний
- ✓ **Системный таймер**

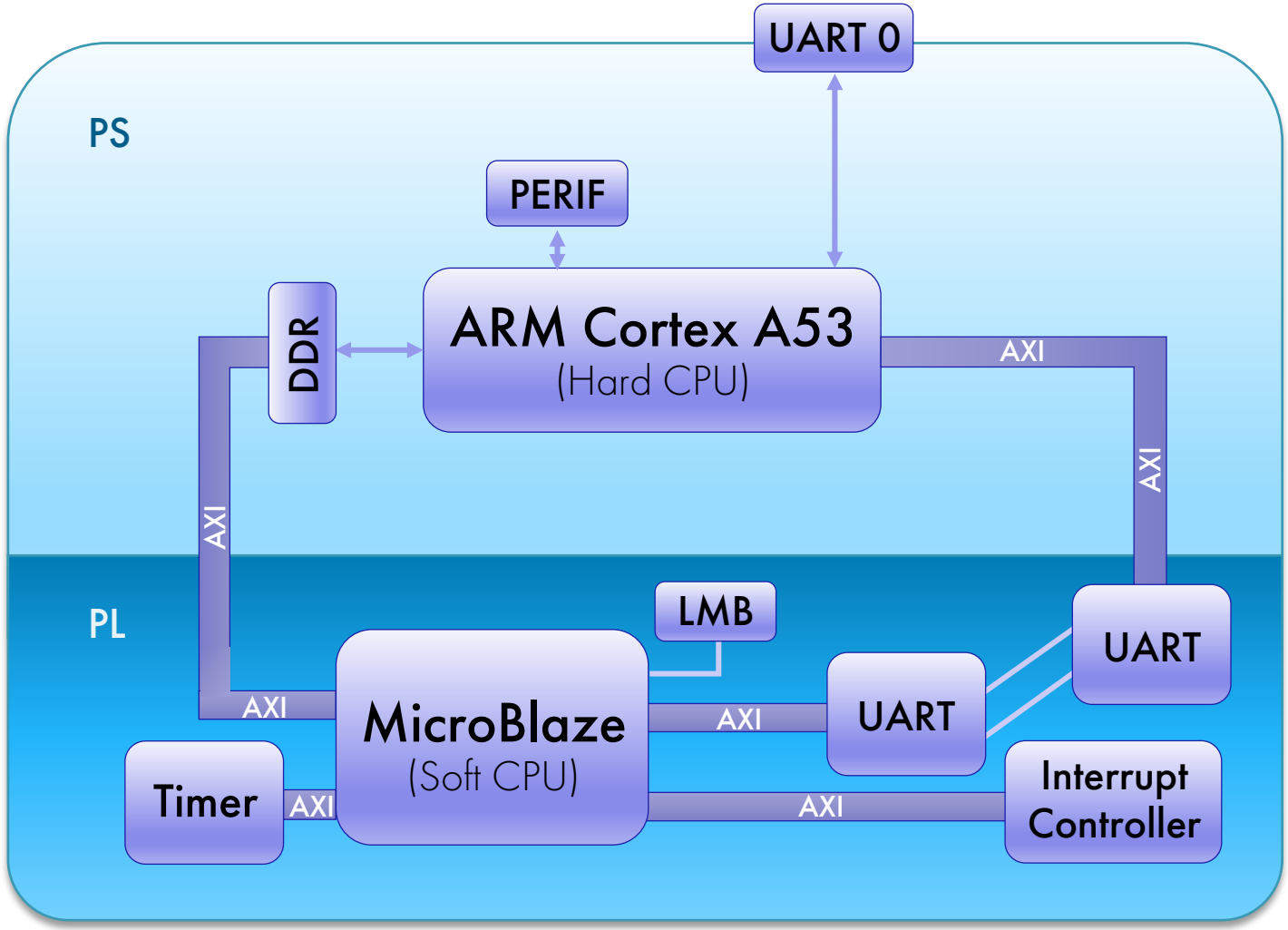
Архитектура проекта



Что нам нужно для запуска Linux:

- ✓ CPU с MMU
- ✓ Оперативная память
- ✓ Контроллер прерываний
- ✓ Системный таймер
- ✓ **Консоль**

Архитектура проекта



Что нам нужно для запуска Linux:

- ✓ CPU с MMU
- ✓ Оперативная память
- ✓ Контроллер прерываний
- ✓ Системный таймер
- ✓ **Консоль**

Архитектура проекта

Архитектура проекта

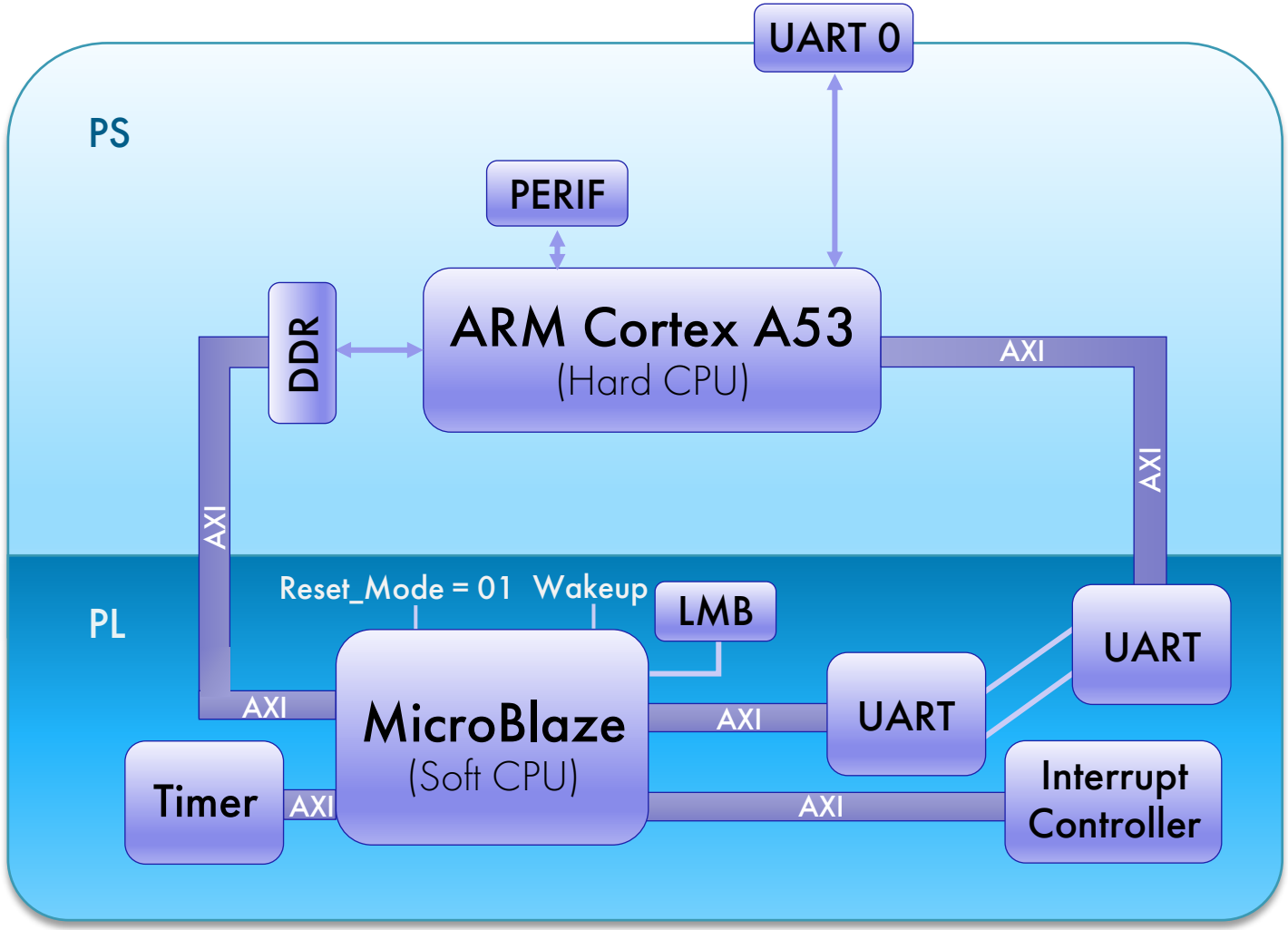
01 – перейти в сон сразу после запуска

Архитектура проекта

01 – стартуем работу!

01 – перейти в сон сразу после запуска

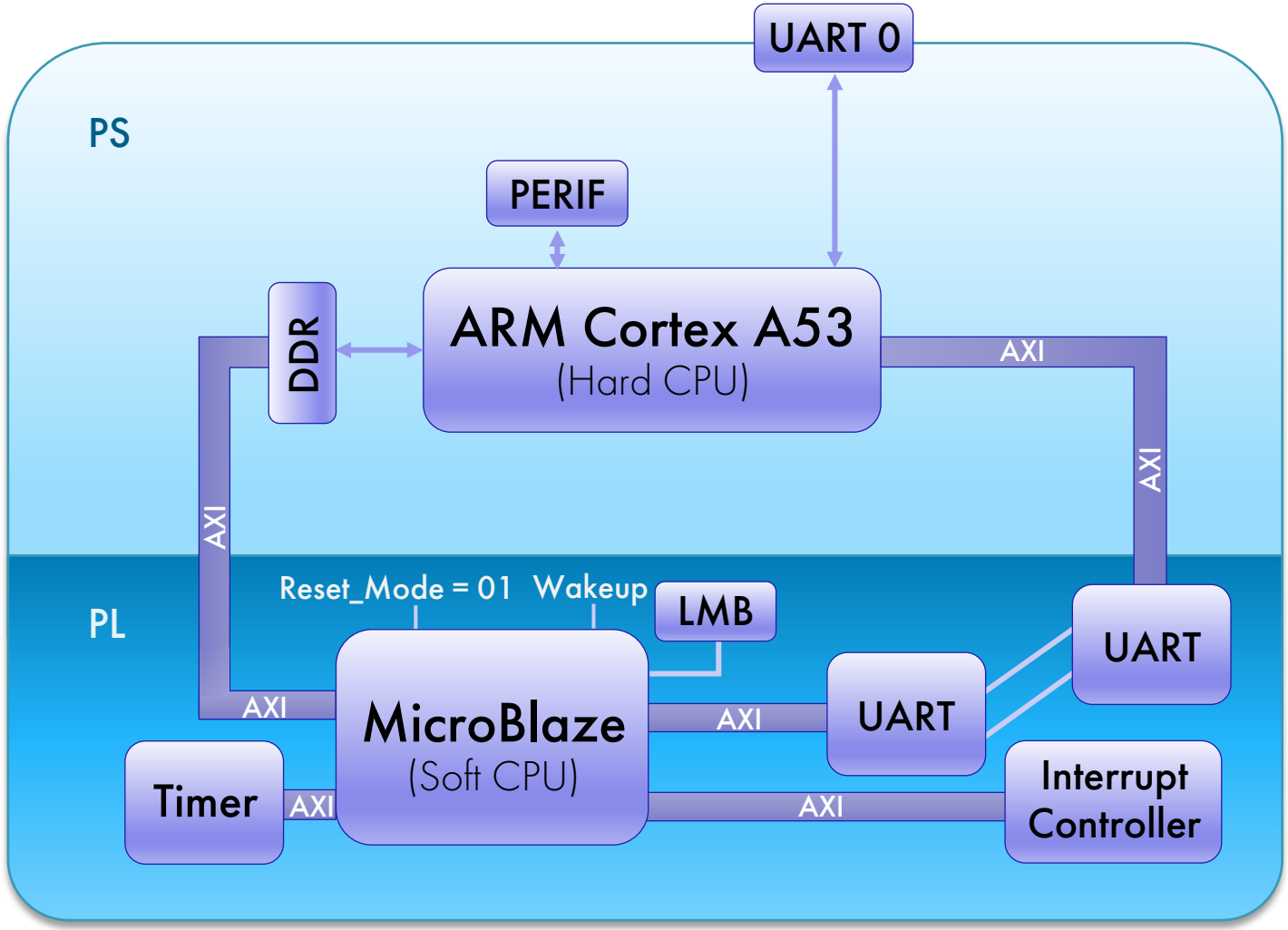
Архитектура проекта



Что нам нужно для запуска Linux:

- ✓ CPU с MMU
- ✓ Оперативная память
- ✓ Контроллер прерываний
- ✓ Системный таймер
- ✓ Консоль
- ✓ **Управление Soft CPU**

Архитектура проекта



Что нам нужно для запуска Linux:

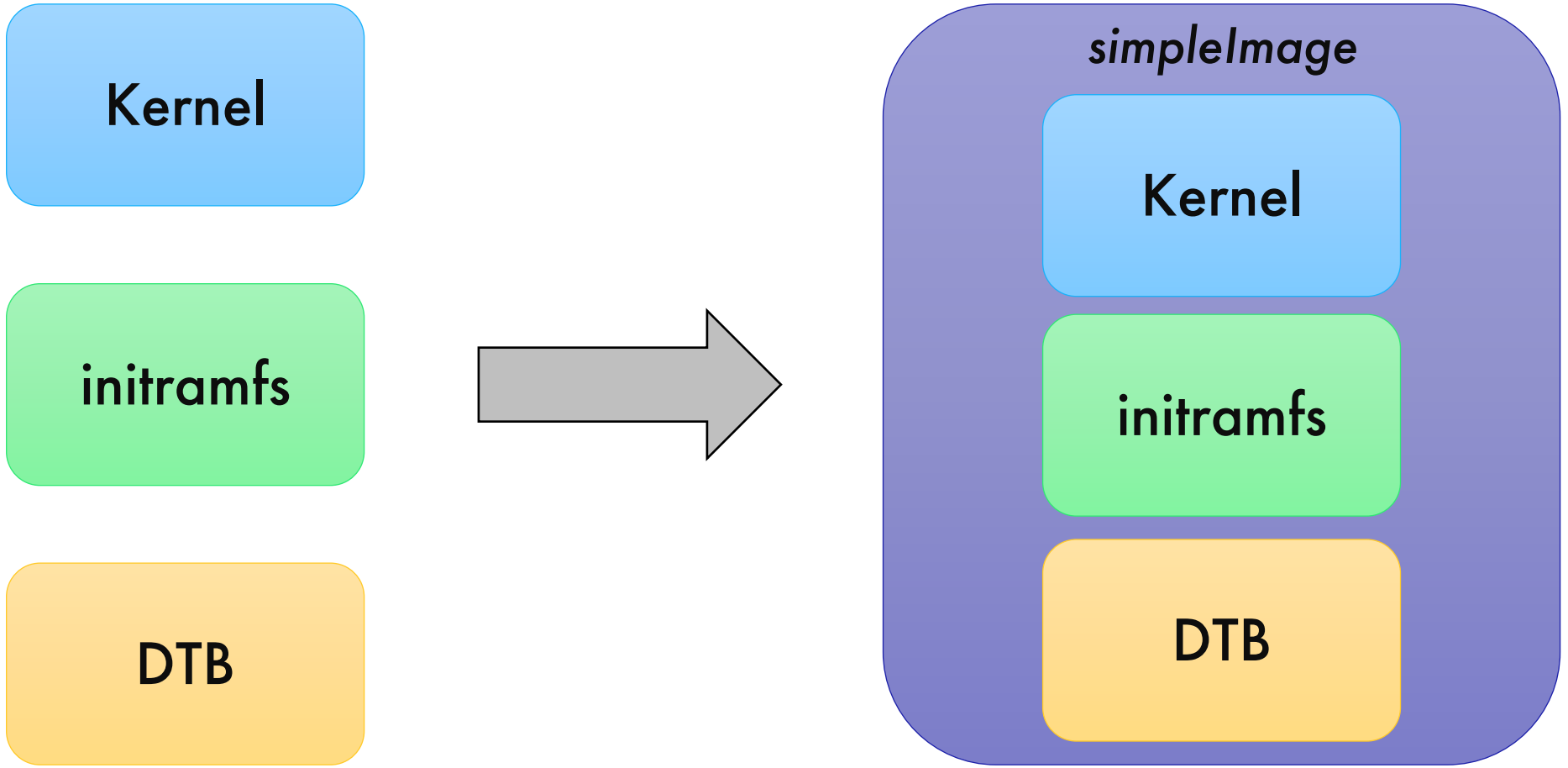
- ✓ CPU с MMU
- ✓ Оперативная память
- ✓ Контроллер прерываний
- ✓ Системный таймер
- ✓ Консоль
- ✓ Управление Soft CPU

Часть 1: проект программируемой логики

Часть 2: Сборка Embedded Linux

Часть 3: запуск и верификация проекта

Компоненты Embedded Linux



Чем собрать Embedded Linux

Две архитектуры

AArch64

HARD CPU
ARM Cortex A53

microblaze-el

SOFT CPU
MicroBlaze

Инструменты для сборки

- Petalinux
- Yocto project
- Buildroot
- Кросс-компиляция совместимым тулчейном

Сборка для Hard CPU (devicetree)

Исключаем одновременный доступ к выделенному региону памяти.

```
reserved-memory {
    #address-cells = <2>;
    #size-cells = <2>;
    ranges;

    reserved: microblaze@0x40000000 {
        no-map;
        reg = <0x0 0x40000000 0x0 0x20000000>;
    };
};
```

Сборка для Hard CPU (Petalinux)

 # petalinux-create

- Создает проект

 # petalinux-config

- Конфигурирует проект основываясь на файле описания аппаратного обеспечения
- Конфигурирует различные компоненты проекта

 # petalinux-build

- Собирает проект

 # petalinux-package

- Подготавливает необходимые загрузочные образы

Сборка для Soft CPU (devicetree)

`xsct% hsi open_hw_design`

- Открывает файл описания аппаратуры

`xsct% hsi set_repo_path`

- Указывает путь к репозиторию компонентов devicetree

`xsct% hsi create_sw_design`

- Создает проект devicetree для заданного процессора

`xsct% hsi generate_target`

- Генерирует файлы devicetree

xsct (Xilinx Software Command-Line Tool) доступен в стандартной поставке Vivado/Vitis (xlnx/Vitis/2024.1/bin/xsct)

Сборка для Soft CPU (кросс-компиляция)

Готовим initramfs

- Сборка фреймворком или вручную
- Можно использовать предварительно-собранный архив
- Опционально копируем в директорию исходного кода ядра Linux

Копируем dts в директорию соответствующей архитектуры

- arch/microblaze/boot/dts
- Необходимо для сборки simpleImage
- Имя файла будет являться частью таргета при построении

* Совместимый тулчейн доступен в стандартной поставке Vivado/Vitis (xlnx/Vitis/2024.1/gnu/microblaze/linux_toolchain/lin64_le/bin/microblazeel-xilinx-linux-gnu-)

Сборка для Soft CPU (кросс-компиляция)

Конфигурируем ядро

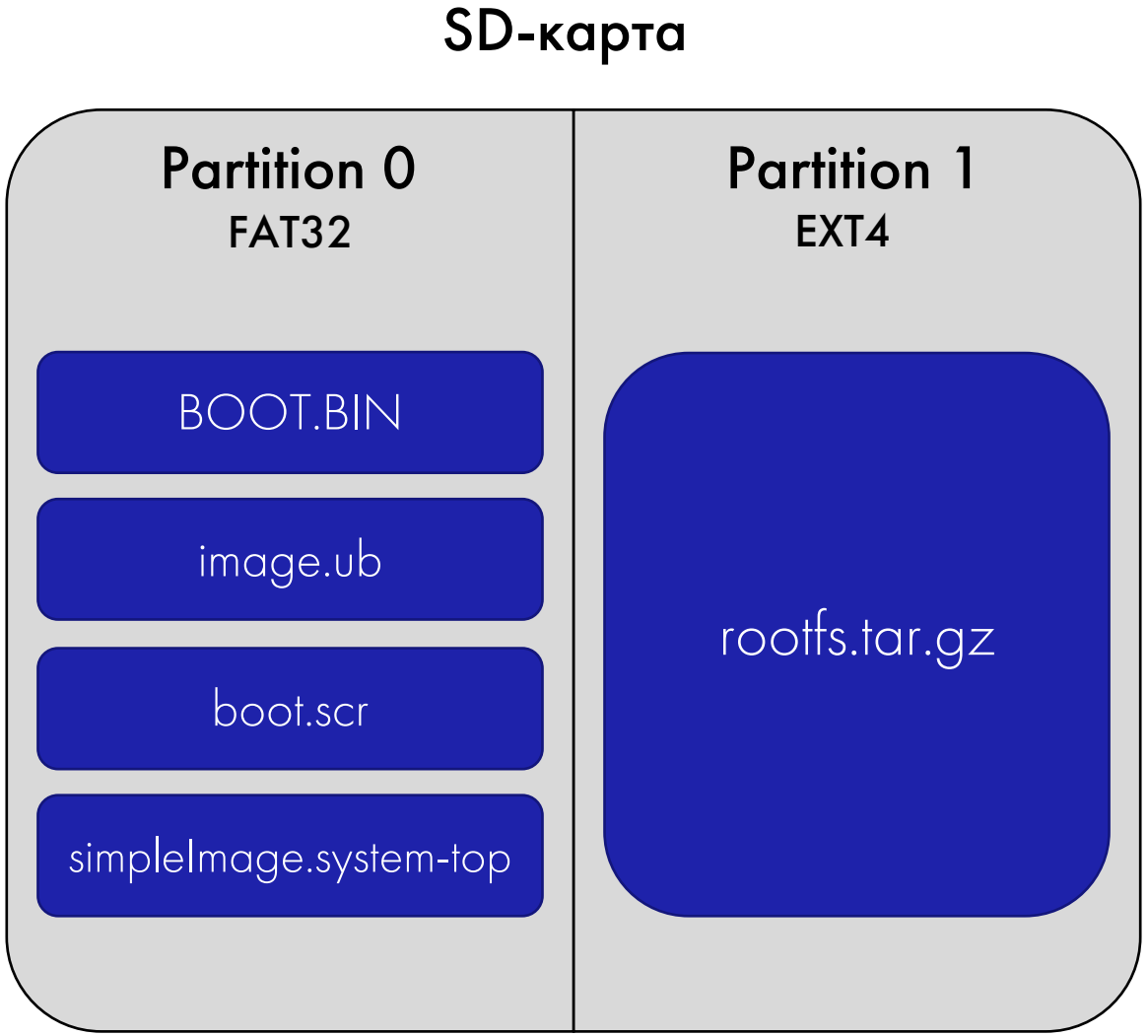
- General setup > initramfs
- Platform options > установить точку входа и сконфигурировать аппаратные компоненты

Собираем проект

```
# make ARCH=microblaze                                  \
       CROSS_COMPILE=microblazeel-xilinx-linux-gnu- \
       simpleImage.system-top
```

Совместимый тулчейн доступен в стандартной поставке Vivado/Vitis (xlnx/Vitis/2024.1/gnu/microblaze/linux_toolchain/lin64_le/bin/microblazeel-xilinx-linux-gnu-)

Подготовка загрузочного носителя



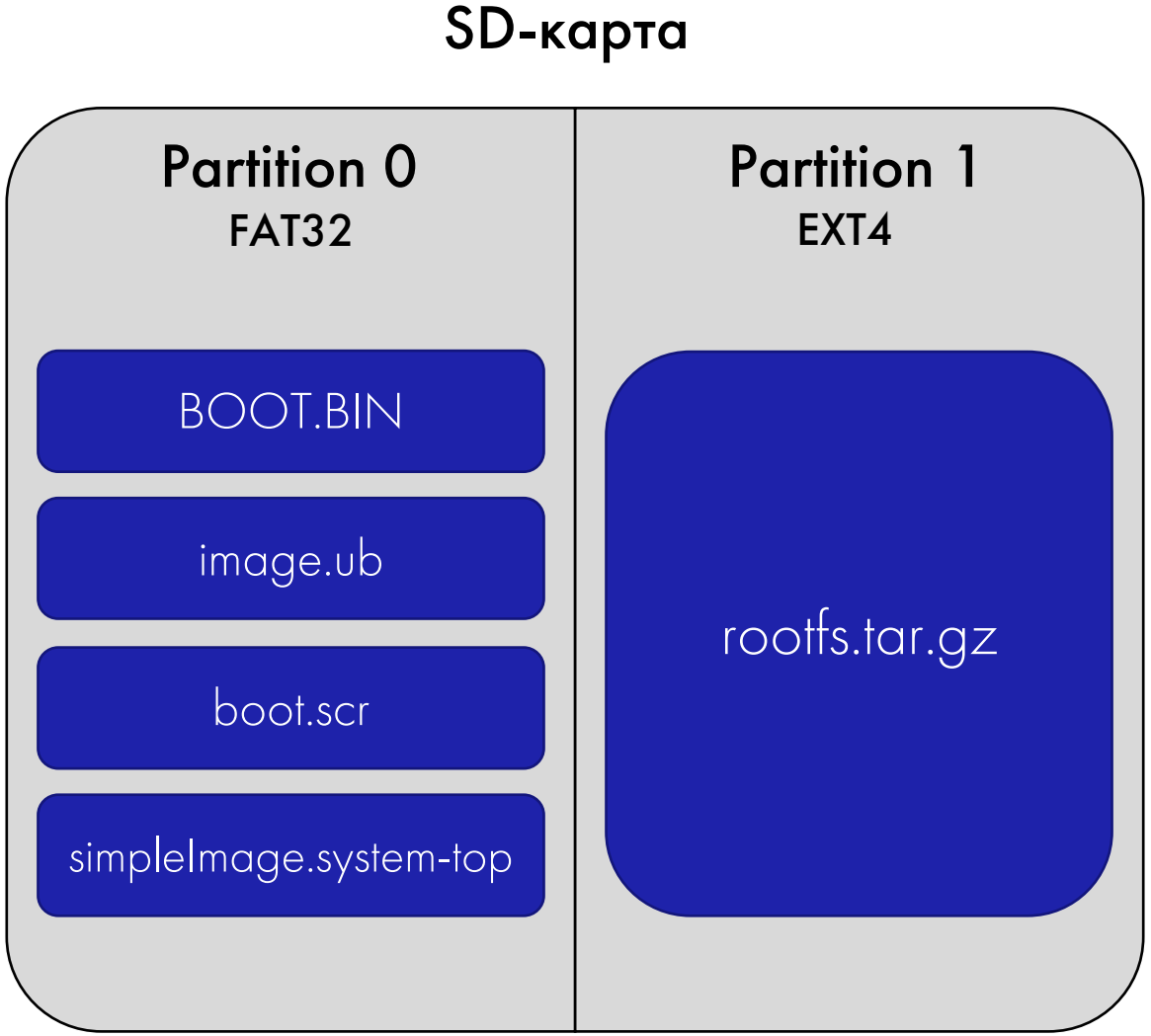
BOOT.BIN

- zynqmp_fsbl.elf
- u-boot.elf
- bl31.elf
- pmufw.elf
- system.bit

Image.ub (Hard CPU)

- Linux kernel
- initramfs
- DTB

Подготовка загрузочного носителя



boot.scr

- Загрузочный скрипт U-Boot

rootfs.tar.gz

- Файловая система, монтируемая вместо initramfs

simpleImage.system-top
(Soft CPU)

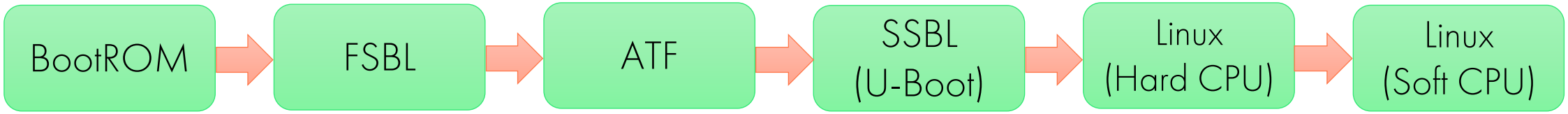
- Linux kernel
- initramfs
- DTB

Часть 1: проект программируемой логики

Часть 2: сборка Embedded Linux

Часть 3: запуск и верификация проекта

Порядок загрузки



- **BootROM** – зашит в постоянную память платы, отвечает за поиск и загрузку FSBL
- **FSBL** - загрузчик первой стадии, отвечает за конфигурацию платформы
- **ATF (ARM Trusted Firmware)** - отвечает за исполнение доверенного ПО
- **SSBL (U-Boot)** - загрузчик второй стадии, отвечает за загрузку образов ОС в ОЗУ
- **Linux (Hard-CPU)** - ОС Hard CPU(ARM Cortex A53)
- **Linux (Soft-CPU)** - ОС Soft CPU (MicroBlaze)

Загрузочный скрипт

```
# dd if=boot.scr of=boot.txt bs=72 skip=1
```

- Убирает из файла заголовок U-Boot

boot.txt

```
fatload mmc 1 0x10000000 image.ub
fatload mmc 1 0x60000000 simpleImage.system-top
cp.b 0x60000000 0x40000000 0x10000000
bootm 0x10000000
```

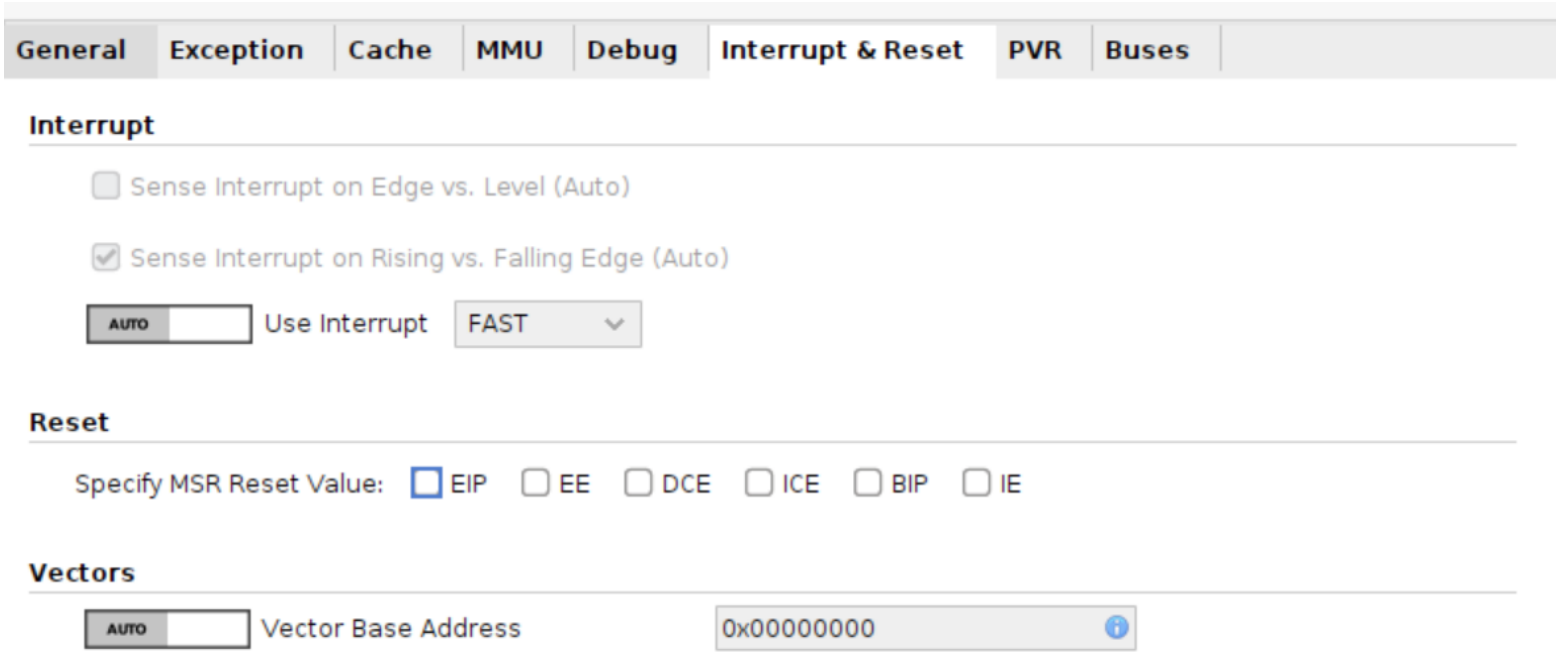
```
# mkimage -A arm -T script -C none -d boot.txt boot.scr
```

- Формирует загрузочный скрипт путем добавления заголовка для U-Boot

Запуск ОС на Soft CPU

Подаем логическую единицу на wakeup порт Microblaze

- Процессор начинает выполнение инструкций по базовому адресу таблицы векторов прерываний



Точка входа в ОС

```
ENTRY(_reset)
    VM_OFF
    brai 0; /* Jump to reset vector */

    /* These are compiled and loaded into high memory, then
     * copied into place in mach_early_setup */

    .section .init.ivt, "ax“

    .org 0x0
    brai CONFIG_MANUAL_RESET_VECTOR
    .org 0x8
    brai TOPHYS(_user_exception);       /* syscall handler */
    .org 0x10
    brai TOPHYS(_interrupt);            /* Interrupt handler */
    .org 0x18
    brai TOPHYS(_debug_exception);      /* debug trap handler */
    .org 0x20
    brai TOPHYS(_hw_exception_handler); /* HW exception handler */
```

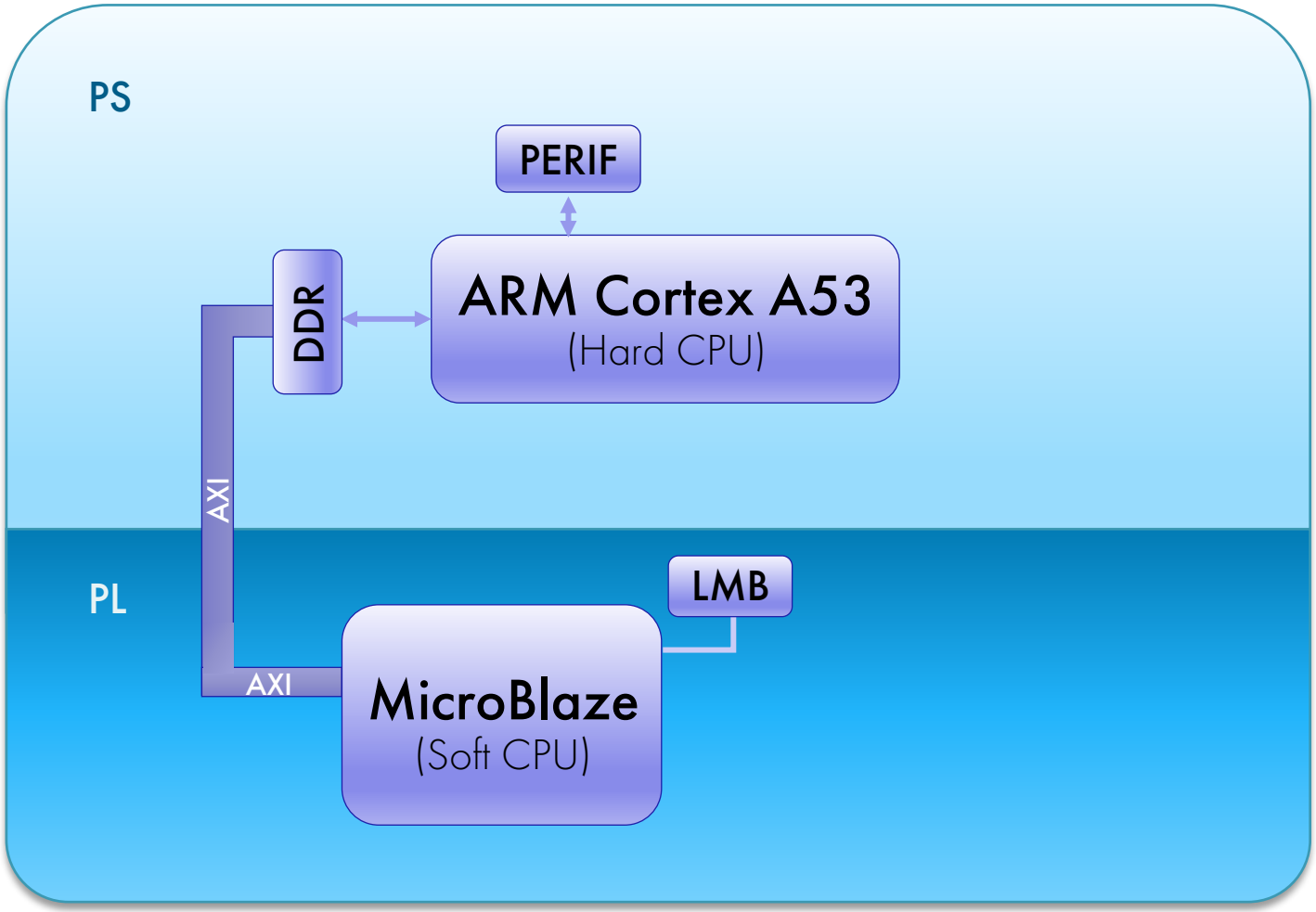
Точка входа в ОС

```
unsigned int offset = 0;

/* Do not copy reset vectors. offset = 0x2 means skip the first
 * two instructions. dst is pointer to MB vectors which are placed
 * in block ram. If you want to copy reset vector setup offset to 0x0 */

#if !CONFIG_MANUAL_RESET_VECTOR
    offset = 0x2;
#endif
    dst = (unsigned long *) (offset * sizeof(u32));
    for (src = __ivt_start + offset; src < __ivt_end; src++, dst++)
        *dst = *src;
```

Архитектура проекта



Что нам нужно для запуска Linux:

- ✓ CPU с MMU
- ✓ **Оперативная память**
- ✓ Контроллер прерываний
- ✓ Системный таймер

Таблица векторов прерываний

```
# reset vector
brai 0x40000000
# user exception handler
brai 0x0
# interrupt handler
brai 0x0
# break handler
brai 0x0
# hw exception handler
brai 0x0
```

Компилируем

```
# microblazeel-xilinx-linux-gnu-as -o vectors.elf vectors.s
```

Ассоциируем файл с блочной памятью в Vivado > Tools > Associate ELF_ files...

Еще раз о процессе загрузки

Init

- Отрабатывает код, зашитый вендором в BootROM, загружает первичный загрузчик и передает ему управление.
- Первичный загрузчик инициализирует платформу, загружает битстрим, передает управление вторичному загрузчику.

Load

- Вторичный загрузчик, размещает образы обеих ОС в ОЗУ основываясь на boot.scr, передает управление на точку входа в ОС Hard-CPU.

Hard

- После загрузки подает логическую “1” на вход “wakeup” Soft-CPU.

Soft

- Начинает выполнение инструкций с адреса 0x0 в BRAM.
- Первая инструкция – переход на точку входа ОС Soft-CPU в ОЗУ.

Результат запуска двух ОС

HARD CPU

```
root@mbpetalinux:~# uname -nrm
mbpetalinux 6.6.10-xilinx-v2024.1-g3af4295e00ef aarch64
root@mbpetalinux:~#
root@mbpetalinux:~# cat /proc/cpuinfo
processor       : 0
BogoMIPS        : 66.66
Features        : fp asimd aes pmull sha1 sha2 crc32 cpuid
CPU implementer : 0x41
CPU architecture: 8
CPU variant     : 0x0
CPU part        : 0xd03
CPU revision    : 4

processor       : 1
BogoMIPS        : 66.66
Features        : fp asimd aes pmull sha1 sha2 crc32 cpuid
CPU implementer : 0x41
CPU architecture: 8
CPU variant     : 0x0
CPU part        : 0xd03
CPU revision    : 4

processor       : 2
BogoMIPS        : 66.66
Features        : fp asimd aes pmull sha1 sha2 crc32 cpuid
CPU implementer : 0x41
CPU architecture: 8
CPU variant     : 0x0
CPU part        : 0xd03
```

SOFT CPU

```
/ # uname -nrm
mb-demo 6.6.0-ge475cc1c9432 microblaze
/ #
/ # cat /proc/cpuinfo
CPU-Family:     MicroBlaze
FPGA-Arch:      UltraScale+ Zynq
CPU-Ver:        11.0, little endian
CPU-MHz:        99.999001
BogoMips:       49.35
HW:
 Shift:         yes
 MSR:           yes
 PCMP:          yes
 DIV:           yes
 MMU:           3
 MUL:           v2
 FPU:           no
 Exc:           op0x0 unal ill iopb dopb zero
Stream-insns:   privileged
Icache:         16kB    line length:    32B
Dcache:         16kB    line length:    16B
Dcache-Policy:  write-through
HW-Debug:       no
PVR-USR1:       00
PVR-USR2:       00000000
Page size:      4096
/ #
```

Выводы

- Проект получил положительные отзывы от целевой аудитории
- Широкий охват прикладных задач позволил эффективно систематизировать знания сотрудников отдела
- Для ряда инженеров проект определил нишу дальнейших исследований и развития компетенций
- Интересные задачи – отличный способ стать автором статьи и попасть на конференцию =)

БУДУЩЕЕ
В НАШИХ
РУКАХ