



СИБИРСКИЙ  
ГОСУДАРСТВЕННЫЙ  
УНИВЕРСИТЕТ  
ТЕЛЕКОММУНИКАЦИЙ  
И ИНФОРМАТИКИ



Центр  
параллельных  
вычислительных  
технологий

Семинар 1

# Архитектура

# СБИС К1879ВМ8Я (NM6408)

**Михаил Георгиевич Курнос**

Цикл семинаров по архитектуре процессоров  
цифровой обработки сигналов НТЦ Модуль NeuroMatrix

# НТЦ Модуль

- С 1995 года создает высокопроизводительные процессорные ядра и аналогово-цифровые системы на кристалле
- <https://module.ru/>

## Процессоры на базе архитектуры NeuroMatrix



# Модули



Включен в перечень по 719 и 878  
Постановлению Правительства РФ



**Модуль NM Card**

Спецвычислители  
подключение в слот PCIe, эффективны для реализации  
нейронных сетей, решения задач цифровой обработки  
сигналов и изображений



**Модуль NM Card Mini**



Включен в перечень по 719 и 878  
Постановлению Правительства РФ



**Модуль NM Quad**

Спецвычислитель  
на основе четырех  
DSP-процессоров K1879BM8Я



Включен в перечень по 719 и 878  
Постановлению Правительства РФ

**СБИС K1879BM8Я** – гетерогенный многоядерный процессор  
цифровой обработки сигналов на базе оригинальной  
архитектуры NeuroMatrix с управляющих RISC-ядром

# Программно-аппаратные комплексы



**Встраиваемый бортовой вычислитель  
NM Pilot**

Универсальный процессор RK3588 (RockChip)  
+ нейропроцессор K1879BM8Я NeuroMatrix



**Портативный компьютер  
NM Vision**

Универсальный процессор RK3399K (RockChip)  
+ нейропроцессор K1879BM8Я NeuroMatrix

# Программное обеспечение RC Module (Linux)

- **Модуль ядра (драйвер), библиотека загрузки и запуска кода**
  - Neuromatrix\_accelerators\_support.deb
  - /usr/local/rc\_module
- **NMC SDK** – средства разработки для NMC (asm/disasm, C/C++, ld, gdb, stdlib), симулятор QEMU
  - <https://module.ru/products/5/neuromatrix-nmc-sdk>
  - /opt/NMC-SDK/doc, /usr/local/bin/nmc-\*, /usr/local/lib/, /usr/local/lib/gcc/nmc/\*, /usr/local/nmc/\*
- **NMDL** (NeuroMatrix DeepLearning) - библиотека реализации нейронных сетей
  - <https://module.ru/products/5/neuromatrix-deep-learning>
- **NMDL Plus** – NMDL + python, новые нейронные сети и операторы
  - <https://module.ru/products/5/neuromatrixr-deep-learning-plus>

# RC Module GitHub

- <https://github.com/RC-MODULE>
- **Neuro Matrix Performance Primitives**
  - <https://github.com/RC-MODULE/nmpp>
  - <https://rc-module.github.io/nmpp/modules.html>
- <https://github.com/RC-MODULE/nmblas>
- <https://github.com/RC-MODULE/nmdnn>
- <https://github.com/RC-MODULE/media.hpp>
- **Jpeg coder implementation for NeuroMatrix**
  - <https://github.com/RC-MODULE/nmJpeg>
- <https://github.com/RC-MODULE/nmprofiler>
- **Step by step tutorial of DSP application development for NeuroMatrix**
  - <https://github.com/RC-MODULE/sobel-steps?tab=readme-ov-file>

# Модуль NM Card Mini

Включен в перечень по 719 и 878  
Постановлению Правительства РФ

- **Модуль NM Card Mini** – спец. вычислитель (ускоритель, DSP-процессор), подключаемый в слот PCI Express на системной плате компьютера/сервера
  - <https://module.ru/products/2-moduli/nm-card-mini>
- **Состав модуля**
  - СБИС K1879BM8Я (16 ядер NMC4, 5 ядер ARM Cortex-A5)
  - 5GB DDR3L SDRAM
  - Ethernet 100 Мб/с
  - PCIe 2.0 x4
  - 4 коммуникационных порта E-Link (до 16 Гб/с) для построения многопроцессорных систем
- Максимальная потребляемая мощность **<= 25 Вт** (типовая 12 Вт)
- **Назначение** – задачи цифровой обработки больших массивов данных, поддержка чисел с плавающей запятой одинарной и двойной точности



# Установка ПО поддержки модуля

```
# Поиск устройства на шине PCI
```

```
$ lspci
```

```
01:00.0 Unassigned class [ff00]: Cadence Design Systems, Inc. Device 0002
```

```
# Установка драйвера (модуль ядра) для модуля
```

```
$ sudo apt install ./Neuromatrix_accelerators_support.deb
```

```
$ tree /usr/local/rc_module
```

```
├── board-nm_card # Библиотека загрузки и обмена
│   ├── bin
│   ├── include
│   └── lib
├── doc
├── driver # Модуль ядра nm_card
├── example
├── flash_prog
└── neuromatrix_temperature_monitor # Утилита мониторинга температуры
```

```
# Проверка загрузки драйвера и функционирование модуля
```

```
$ ls /dev/NM*
```

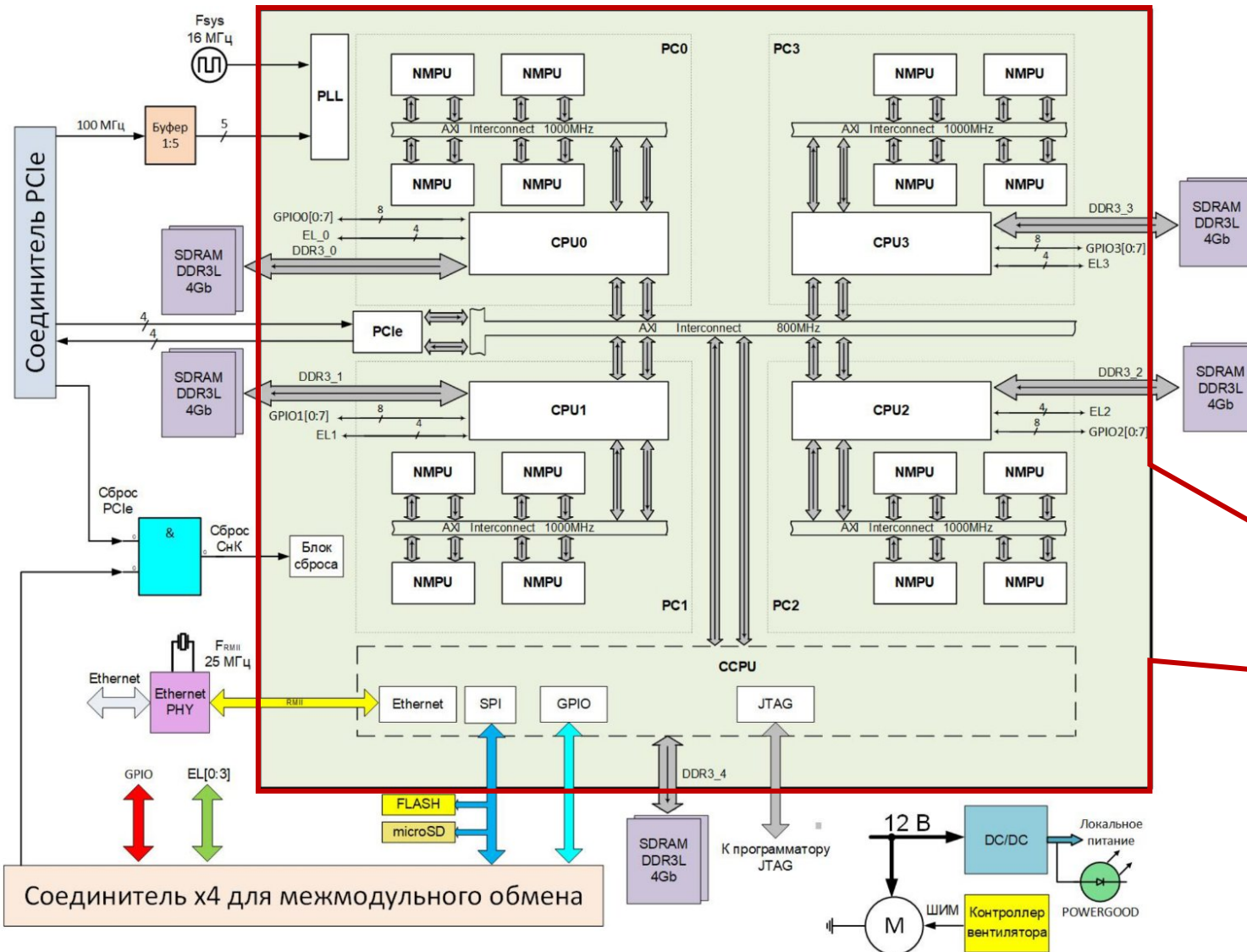
```
/dev/NM-Card_00
```

```
$ lsmod | grep nm_card
```

```
nm_card                40960  0
```

```
$ /usr/local/rc_module/neuromatrix_temperature_monitor/bin/NeuroMatrix_Temperature_Monitor_Console
```

# Модуль NM Card Mini



- **СБИС K1879BM8Я (NM6408)**
  - 16 ядер NMC4
  - 5 ядер ARM Cortex-A5
- **SDRAM: 5GB DDR3L**
- **NIC: Ethernet 100 Mб/с**
- **PCIe 2.0 x4**
- **4 коммуникационных порта E-Link для межмодульного обмена (до 16 Гб/с)**

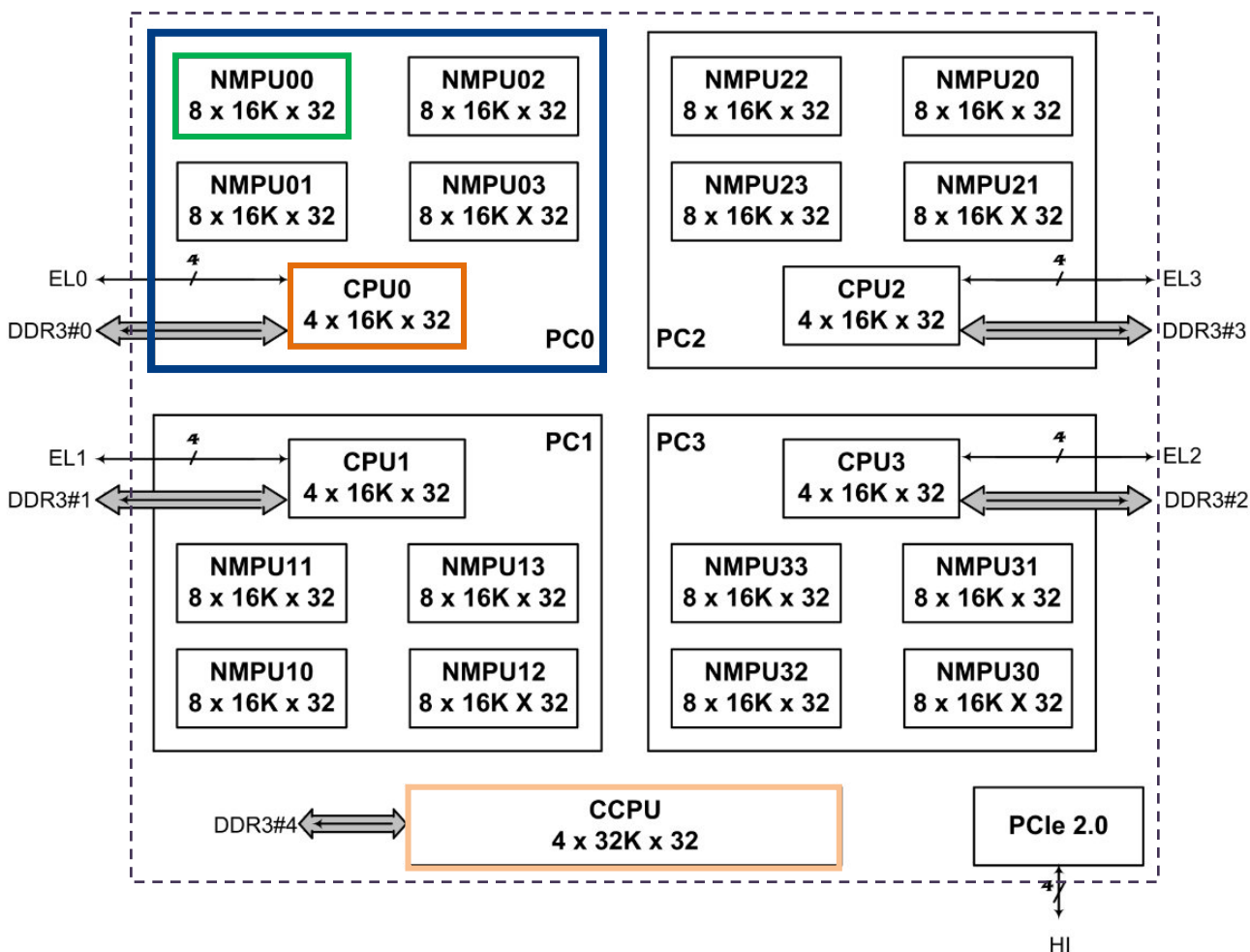


# СБИС K1879BM8Я (SoC NM6408)



- Гетерогенная многопроцессорная система на кристалле (СнК, SoC), 512 GFLOPS

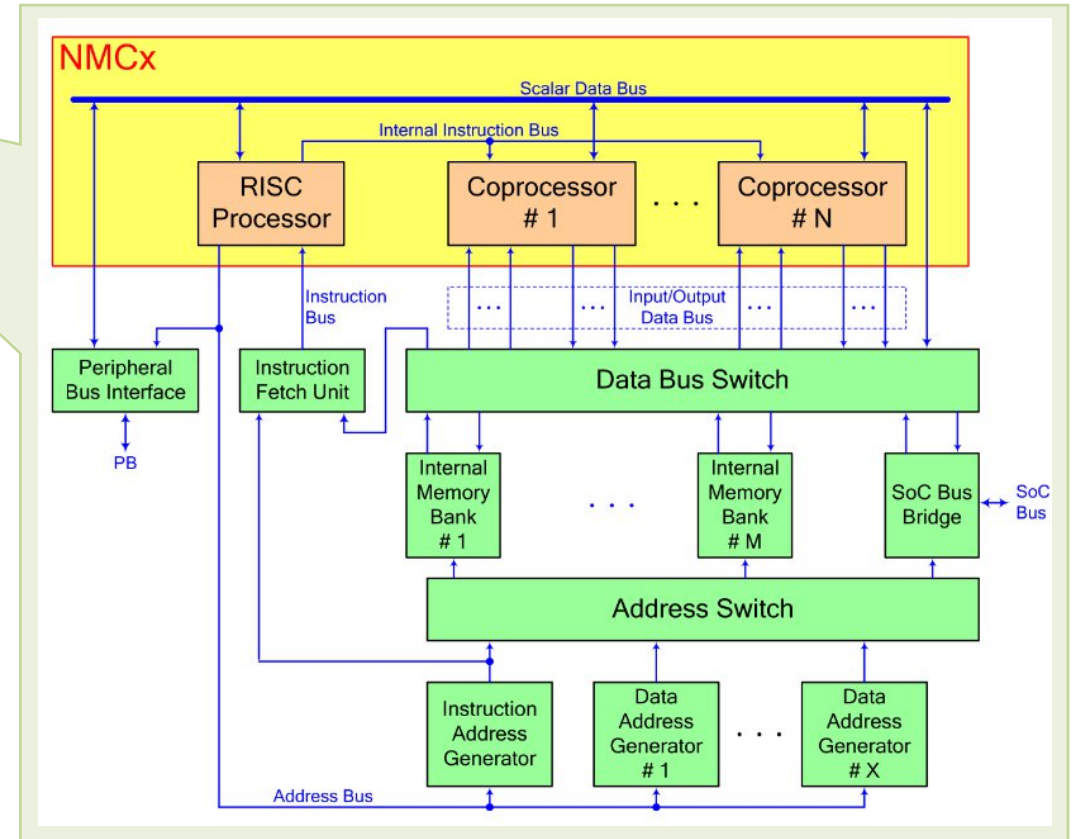
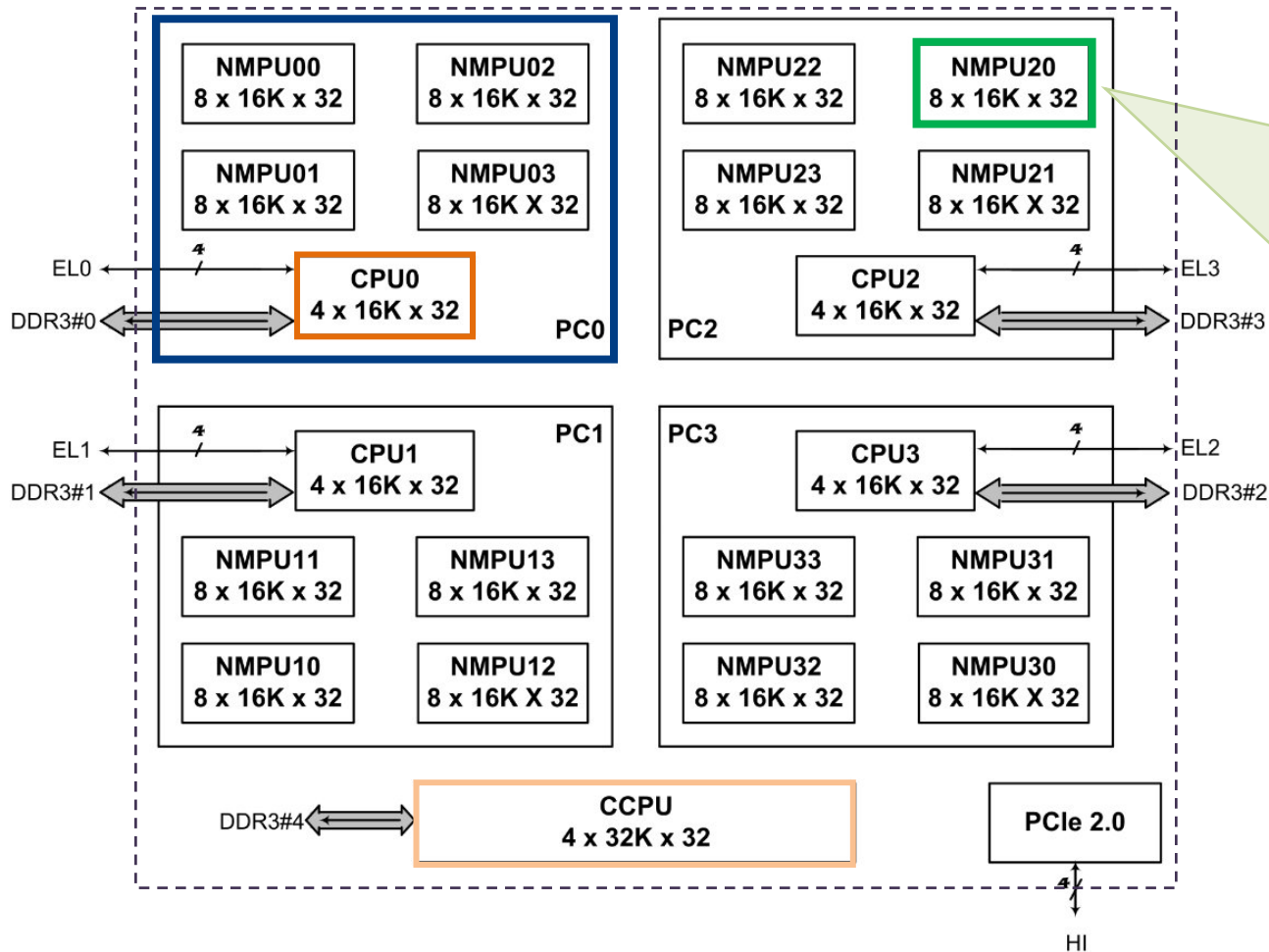
- <https://www.module.ru/products/1/26-18798>
- 16 процессорных ядер NeuroMatrix Core 4 (DSP)
- 5 ядер ARM Cortex-A5 (управление системой)



- **Процессорный узел (ПУ)** – ядро ARM/NMC и 4 или 8 банков памяти (16Kx32 бита)
- **Процессорный кластер (PC – Processing Cluster)** – 4 ПУ на базе NMC, 1 ПУ ARM, контроллер DDR3, 128 GFLOPS
- **NMPU (NeuroMatrix Processing Unit)** – ПУ на базе ядра NMC4 (1 GHz), 8 банков памяти, 32 GFLOPS
- **CPU (Cortex Processing Unit)** – управляющий ПУ на базе ядра ARM (800 MHz), 4 банка памяти, доступ к DDR3, доступ к другим модулям через каналы EL
- **CCPU (Central Cortex Processing Unit)** – центральный управляющий ПУ на базе ARM (600 MHz), 4 банка внутренней памяти, управление системой и доступ к памяти DDR3 по 32-разрядной шине данных
- **Host** – системная плата с процессором x86-64 (Elbrus, ARM) к шине PCIe, которого подключен модуль
- **ARM-часть** – ПУ на ядре ARM SoC NM6408
- **NMC-часть** – ПУ на ядре NMC SoC NM6408
- **Board** – SoC NM6408 + его ресурсы

# Ядро NMC4 (32 GFLOPS)

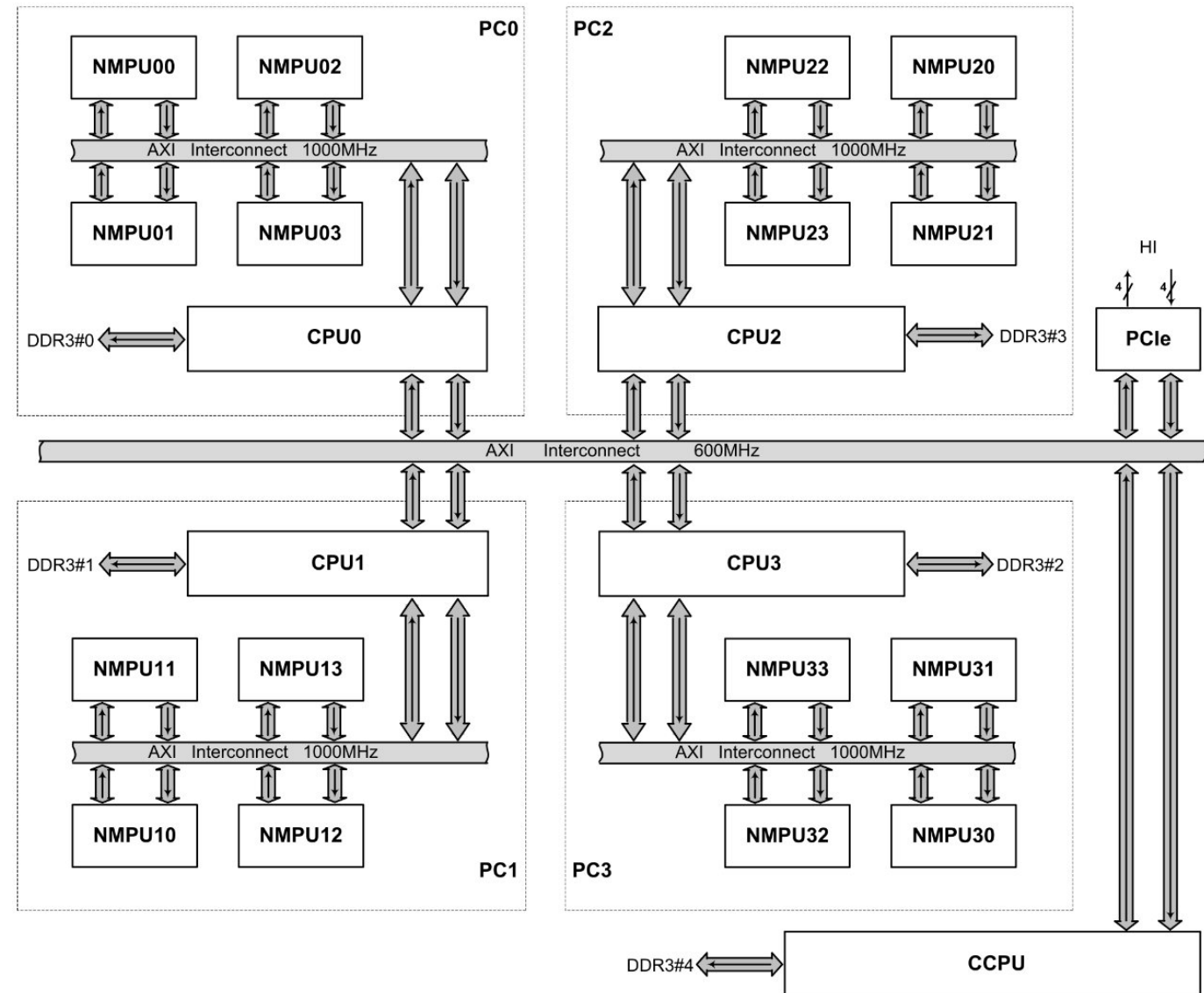
- Управляющее RISC-ядро (декодирование VLIW инструкций)
- Сопроцессоры цифровой обработки данных (векторно-матричные)



- Управляющее RISC-ядро (декодирование инст.)
- Сопроцессоры цифровой обработки данных

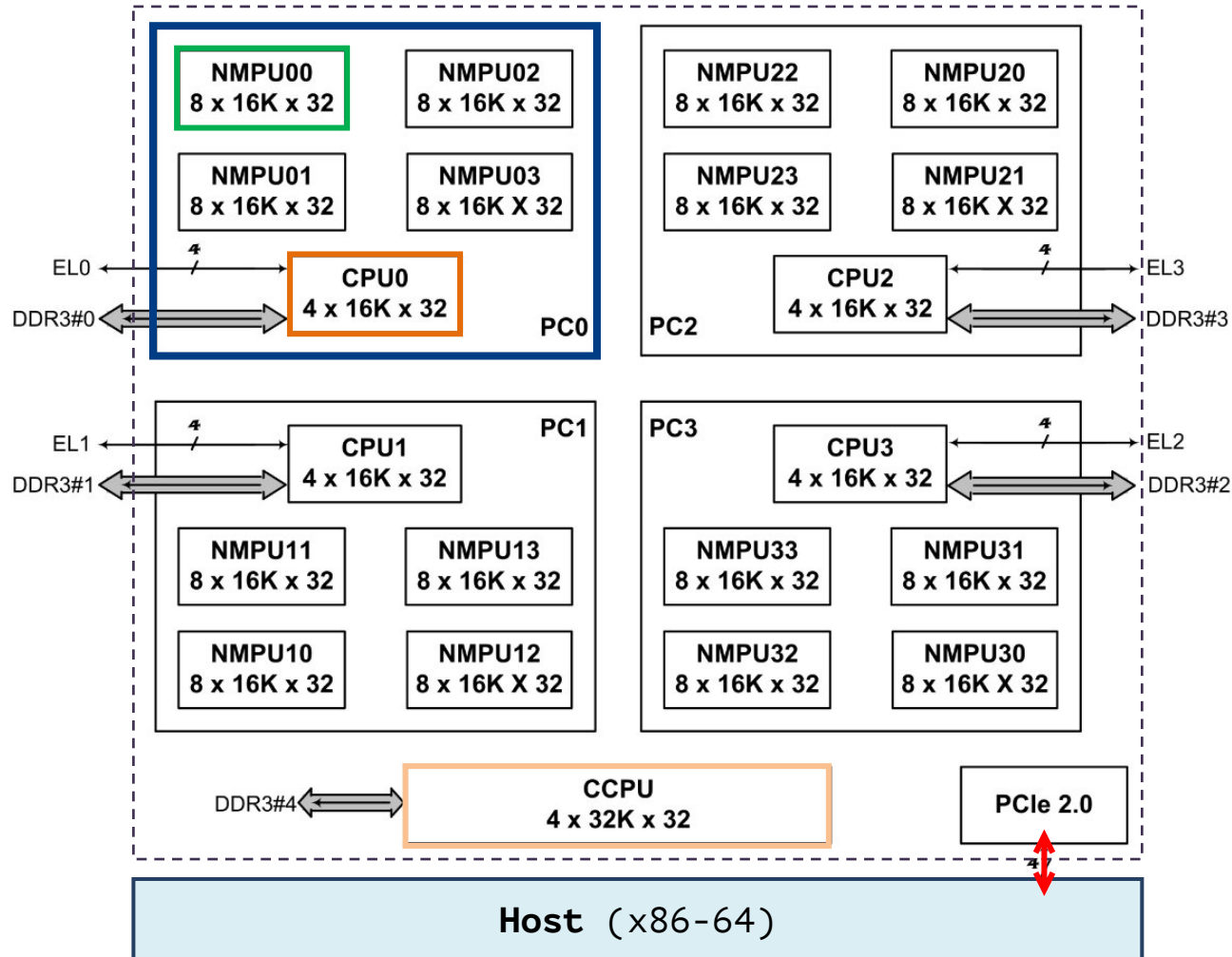
# Память SoC NM6408

- Внутренняя память и DDR3 включена в единое адресное пространство, доступное всем ядрам ARM и NMC
- Единое адресное пространство в СнК охватывает 4 ГБ
- Все NMPU имеют доступ к внутренней памяти друг друга и DDR3
- NMC и ARM формируют 32-разрядные адреса



# Библиотека загрузки и обмена SoC NM6408

- **libnm\_card\_load.so** (/usr/local/rc\_module/board-nm\_card/lib/libnm\_card\_load.so)
  - host-часть библиотеки для загрузки и запуска кода с хоста на ПУ ARM или NMC



## Сценарии кросс-компиляции кода и его запуска

- Кросс-компиляция кода на хосте x86-64 и запуск с хоста на ПУ NMC
- Кросс-компиляция кода на хосте x86-64 и запуск с хоста на ПУ ARM

# Средства кросс-компиляции для SoC NM6408

# Устанавливаем на хосте x86-64 средства кросс-компиляции для ARM Cortex-A5

# ARM Cortex-A5: ARMv7, 32-разрядная ISA ARMv7, in-order

```
$ sudo apt install gcc-arm-none-eabi
```

```
$ arm-none-eabi-gcc --version
```

```
arm-none-eabi-gcc (15:13.2.rel1-2) 13.2.1 20231009
```

# Устанавливаем на хосте x86-64 пакет NMC SDK

```
$ sudo apt install ./NMC-SDK-1.4.246.783a74b.x86_64.deb
```

```
$ nmc-gcc --version
```

```
nmc-gcc (NMC SDK GCC ) 4.8.3
```

# Запуск кода на ARM-ядре SoC NM6408

```
#include <stdlib.h>
#include <stdio.h>
#include "nm6408load_arm.h" // ARM часть библиотеки загрузки

int fact(int n)
{
    if (n == 1)
        return 1;
    int res = fact(n - 1) * n;
    return res;
}

int main(int argc, char *argv[])
{
    int n = 10;
    if (argc) {
        printf("ARM%d args - num = %d, ptr = 0x%x\n", acl_getClusterID(), argc, argv);
        if (argc > 1)
            n = atoi(argv[1]);
    }

    for (int i = 0; i < argc; i++)
        printf("ARM%d argv[%d] (addr 0x%x) = %s\n", acl_getClusterID(), i, *(argv + i), argv[i]);

    printf("ARM%d, board %s: fact(%d)\n", acl_getClusterID(), acl_getBoardName(), n);
    return fact(n);
}
```

# Запуск кода на ARM-ядре SoC NM6408

```
$ cat build.sh
```

```
RCM_HOME=/usr/local/rc_module
```

```
RCM_BOARD="nm_card"
```

```
RCM_BOARD_PATH=$RCM_HOME/board-$RCM_BOARD
```

```
export PATH=$RCM_HOME/$RCM_BOARD/bin:$PATH
```

```
export LD_LIBRARY_PATH=$RCM_HOME/$RCM_BOARD/lib:$LD_LIBRARY_PATH
```

```
unset LANG
```

```
make ARM_CROSS_COMPILE="arm-none-eabi-" NMC_CROSS_COMPILE="nmc-" \  
BOARD=$RCM_BOARD BOARD_PATH=$RCM_BOARD_PATH
```

```
$ file arm/central_arm_part.elf
```

```
arm/central_arm_part.elf: ELF 32-bit LSB executable, ARM, EABI5 version 1 (SYSV), statically linked...
```

```
$ nm_card_run arm/central_arm_part.elf -p -c-1 -n-1 --args="10"
```

```
Batch loader for NM_Card v2.1.240408.a4de; (C) 2023 RC Module Inc.
```

```
LibLoad v2.1.240408.a4de
```

```
Firmware v0.2.0.0
```

```
Serial n144
```

```
Start user program on core -1.-1...
```

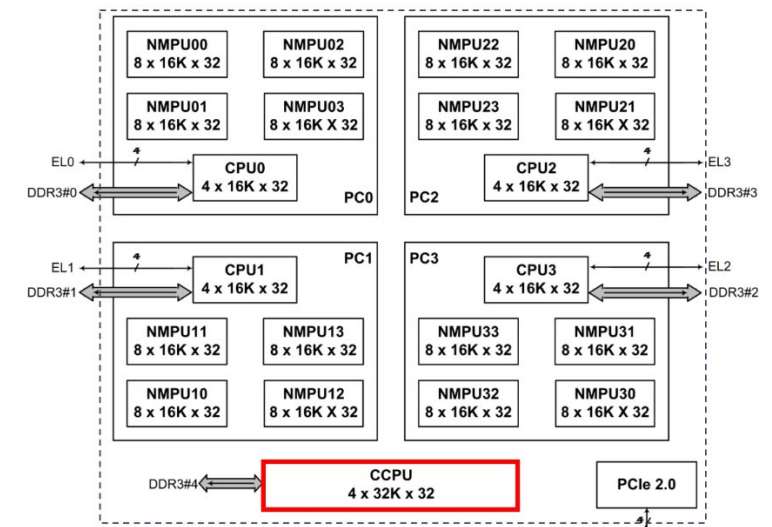
```
ARM4 args - num = 2, ptr = 0x8920
```

```
ARM4 argv[0] (addr 0x8900) = arm/central_arm_part.elf
```

```
ARM4 argv[1] (addr 0x891c) = 10
```

```
ARM4, board MC127.05: fact(10)
```

```
arm/central_arm_part.elf :: return 3628800 = 0x375F00.
```



# Запуск кода на NMC-ядре SoC NM6408

```
#include <stdlib.h>
#include <stdio.h>
#include "nm6408load_nmc.h" // NMC часть библиотеки загрузки

int fact(int n)
{
    if (n == 1)
        return 1;
    int res = fact(n - 1) * n;
    return res;
}

int main(int argc, char *argv[])
{
    int n = 10;
    if (argc) {
        printf("NMC%d:%d args - num = %d, ptr = 0x%x\n", ncl_getClusterID(), ncl_getCoreID(), argc, argv);
        if (argc > 1)
            n = atoi(argv[1]);
    }

    for (int i = 0; i < argc; i++)
        printf("NMC%d:%d argv[%d] (addr 0x%x) = %s\n", ncl_getClusterID(), ncl_getCoreID(),
            i, *(argv + i), argv[i]);

    printf("Board %s NMC%d:%d, fact(%d)\n", ncl_getBoardName(), ncl_getClusterID(), ncl_getCoreID(), n);
    return fact(n);
}
```

# Запуск кода на NMC-ядре SoC NM6408

```
$ cat build.sh
```

```
RCM_HOME=/usr/local/rc_module
```

```
RCM_BOARD="nm_card"
```

```
RCM_BOARD_PATH=$RCM_HOME/board-$RCM_BOARD
```

```
export PATH=$RCM_HOME/$RCM_BOARD/bin:$PATH
```

```
export LD_LIBRARY_PATH=$RCM_HOME/$RCM_BOARD/lib:$LD_LIBRARY_PATH
```

```
unset LANG
```

```
make ARM_CROSS_COMPILE="arm-none-eabi-" NMC_CROSS_COMPILE="nmc-" \
```

```
BOARD=$RCM_BOARD BOARD_PATH=$RCM_BOARD_PATH
```

```
$ file file nmc/nmc0_part.abs
```

```
nmc/nmc0_part.abs: ELF 32-bit LSB executable, *unknown arch 0xfffffa32* version 1 (SYSV), statically linked
```

```
$ nm_card_run nmc/nmc0_part.abs -p -c0 -n0
```

```
Batch loader for NM_Card v2.1.240408.a4de; (C) 2023 RC Module Inc.
```

```
LibLoad v2.1.240408.a4de
```

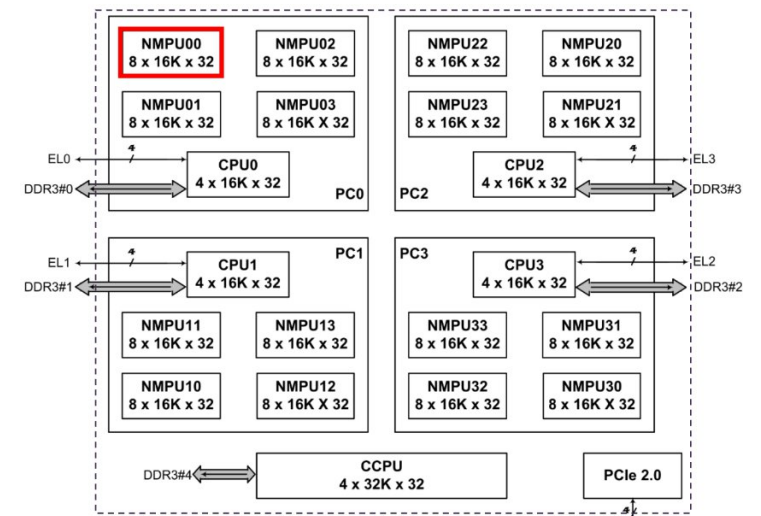
```
Firmware v0.2.0.0
```

```
Serial n144
```

```
Start user program on core 0.0...
```

```
Board MC127.05 NMC0:0, fact(10)
```

```
nmc/nmc0_part.abs :: return 3628800 = 0x375F00.
```



# API для запуска кода с хоста

- **Библиотека загрузки и выполнения (хост часть, x86-64)**
  - `/usr/local/rc_module/board-nm_card/lib/libnm_card_load.so`
  - `/usr/local/rc_module/board-nm_card/include/nm_card_load.h`
- `PL_GetBoardCount()` – определение количества доступных экземпляров модуля
- `PL_GetBoardDesc()` – получение дескриптора доступа к экземпляру модуля
- `PL_ResetBoard()` – перезагрузка экземпляра модуля
- `PL_LoadInitCode()` – загрузка кода начальной инициализации
- `PL_GetAccess()` – получение дескриптора доступа к процессорному узлу модуля (ARM, NMC)
  - `core.cluster_id`: -1 - CCPU, 0-3 – номер кластера
  - `core.nm_id`: -1 – ARM ядро, 0-3 – NMC ядро
- `PL_LoadProgramFile(char *filename)` – загрузка и запуск ELF-файла (скомпилированного!)
- `PL_GetResult()` – получение возвращаемого значения пользовательской программы
- `PL_GetStatus()` – состояние выполнения программы
- `PL_ReadMemBlock()` – чтение блока из разделяемой памяти модуля
- `PL_WriteMemBlock()` – запись блока данных в разделяемую память модуля

# API для запуска кода с хоста (x86-64)

```
#include "nm_card_load.h"
#include "io_host.h"

int main() {
    unsigned int boardCount;
    PL_GetBoardCount(&boardCount);

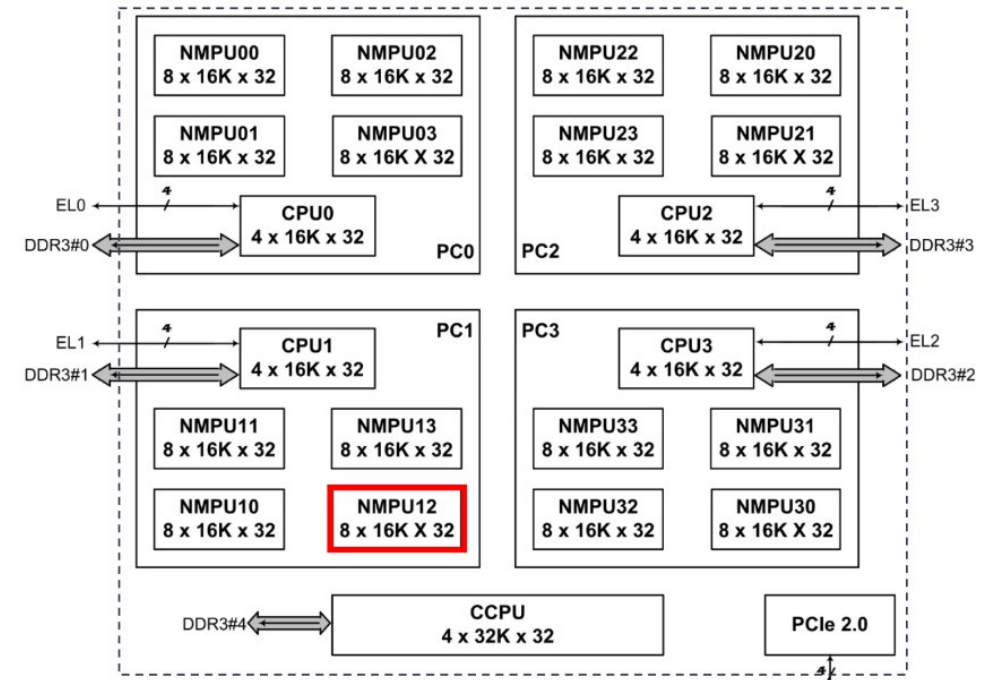
    PL_Board *board;
    PL_GetBoardDesc(0, &board);
    PL_ResetBoard(board);
    PL_LoadInitCode(board);

    const char *filename = "./nmc/nmc0_part.abs";
    PL_CoreNo core = { .nm_id = 2, .cluster_id = 1 }; // Запуск на ПУ NMC 2, кластер 1
    PL_Access *access;
    PL_GetAccess(board, &core, &access);
    PL_LoadProgramFile(access, filename);

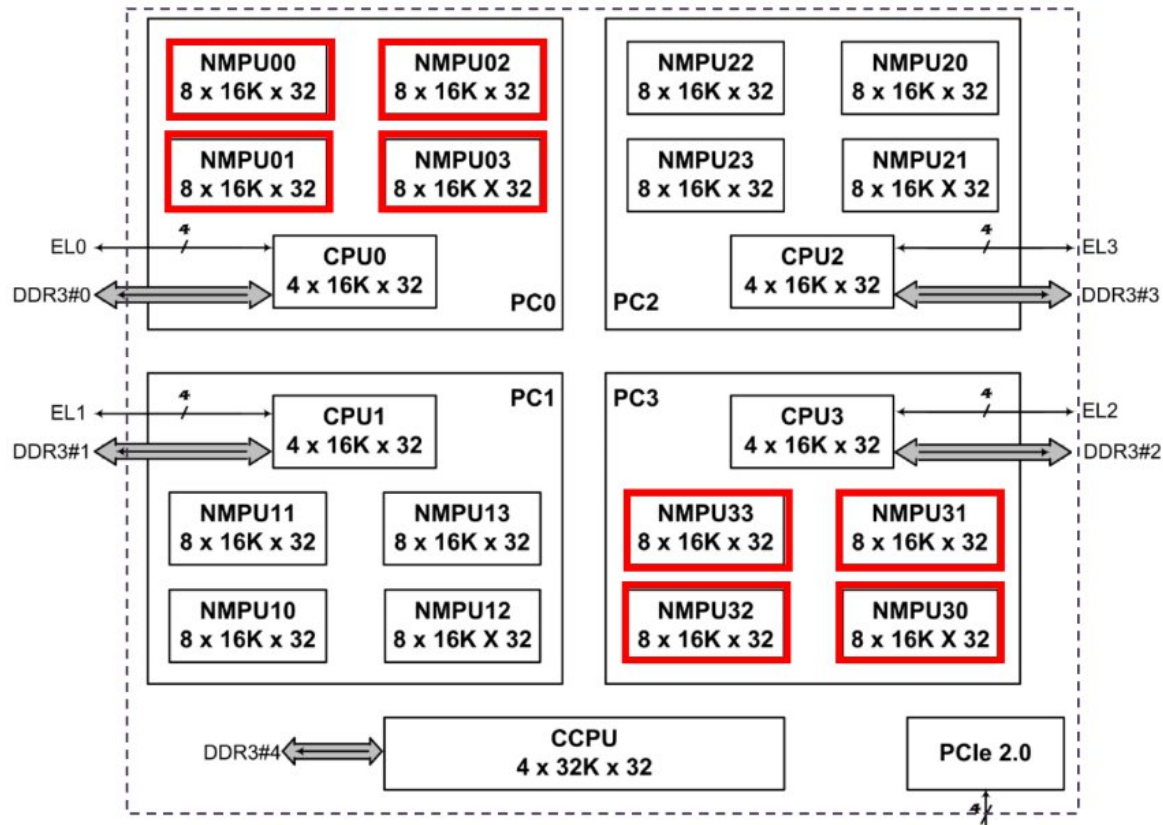
    IO_Service *service = IO_ServiceStart(access, NULL, NULL, NULL);
    PL_Word status, result;
    int is_finished = 0;
    do {
        PL_GetStatus(access, &status);
        if (status == PROGRAM_FINISHED) is_finished = 1;
    } while (!is_finished);

    PL_GetResult(access, &result);
    cout << "Result: " << result << "\n";

    IO_ServiceStop(&service, NULL);
    PL_CloseAccess(access);
    PL_CloseBoardDesc(board);
    return 0;
}
```



# Запуск кода на множестве ПУ NMC



1. Скомпилировать код для NMPU с учетом их номеров и карт памяти
2. Получить доступ ко всем NMPU:  
`PL_GetAccess(board, &core, &access)`
3. Запустить на программы на всех NMPU:  
`PL_LoadProgramFile(access, filename)`
4. Дождаться завершения программ на NMPU

## ■ Примеры

- `/usr/local/rc_module/example/simple/host_pthread.c`

# Что дальше?

- Карта памяти SoC
- Динамическое выделение памяти (DDR3, локальные банки)
- Распределение вычислений между ПУ на базе NMC, синхронизация выполнения
- Обмен информацией: хост – SoC, ARM – NMC, NMC – NMC, ARM – ARM
- Система команд NMC (VLIW SIMD, структура матрично-векторных сопроцессоров)
- Отладка, профилирование и оптимизация кода
- Прикладные библиотеки (DSP, BLAS, NN)
  
- **Кластер СибГУТИ на базе модулей НТЦ Модуль**
  - 2 сервера (2 x NM Quad), 1 сервер с NM Card Mini
  - Доступ через систему очередей SLURM
  - <https://wiki.csc.sibsutis.ru/en/ClusterNMQuad>