Лекция 6 Разработка и отладка программ для встраиваемых систем

План курса «Встраиваемые микропроцессорные системы»:

Лекция 1: Введение. Язык программирования С

Лекция 2: Язык программирования С, применение для встраиваемых систем

Лекция 3: Стандартная библиотека языка С

Лекция 4: Ядро ARM Cortex-M3. Микроконтроллер Миландр К1986ВЕ92QI

Лекция 5: Этапы разработки микропроцессорных систем

Лекция 6: Разработка и отладка программ для встраиваемых систем

Лекция 7: Архитектура программного обеспечения

Лекция 8: Периферийные модули: Timer, DMA, ADC, DAC

Лекция 9: Периферийные модули: CAN, USB, Ethernet, SDIO

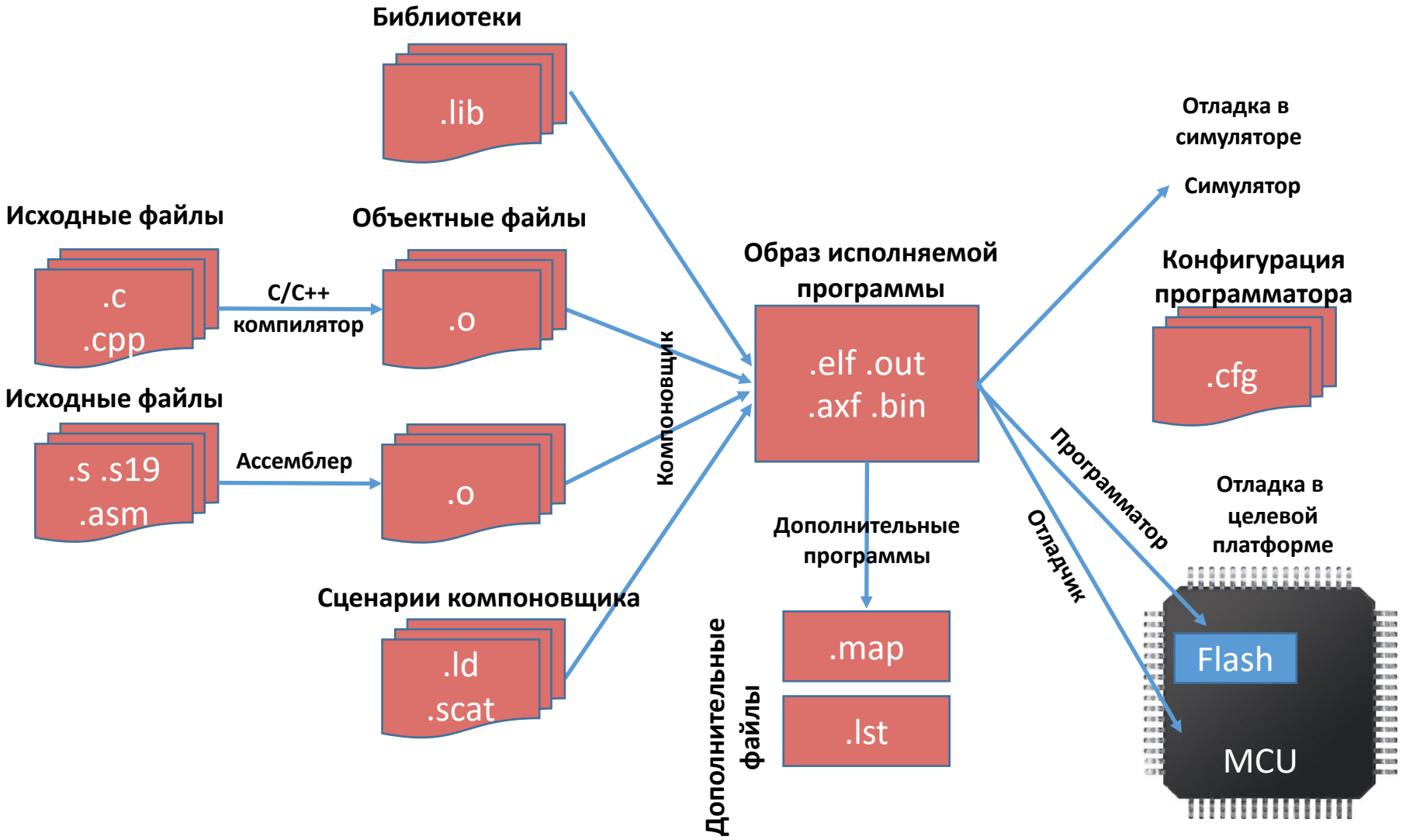
Программное обеспечение, используемое при разработке встраиваемой системы

1. Встраиваемое программное обеспечение – программное обеспечение для исполнения в микроконтроллере;
2. Инструментальное программное обеспечение – комплекс программ для разработки и отладки встраиваемого программного обеспечения;
3. Программное обеспечение общего назначения – вспомогательное программное обеспечение, например браузер или текстовый редактор.

Инструментальное программное обеспечение

1. Текстовый редактор с подсветкой синтаксиса (text editor);
2. Ассемблер (assembler);
3. Компилятор (compiler);
4. Компоновщик (линковщик, linker);
5. Генератор исходных текстов (source code generator);
6. Программно-логический симулятор (simulator);
7. Отладчик (debugger);
8. Драйвер для отладочного устройства;
9. Драйвер для программатора;
10. Менеджер проекта;
11. Графические интерпретаторы отладки/симуляции.

Процесс сборки встраиваемого программного обеспечения



Редактор кода или интегрированная среда разработки

- **Интегрированная среда разработки (IDE – Integrated Development Environment).**
  Все инструментальное программное обеспечение собрано в одной программе.
  Настройка проекта в графическом интерфейсе.
  Все инструментальное программное обеспечение собрано в IDE.
  Пример: Keil MDK, Atmel Studio, Visual Studio, Eclipse, Arduino IDE.

- **Редактор кода и сборка проекта в командной строке.**
  Запуск проекта например в утилите make и по сценарию из текстового файла Makefile.
  Необходимое программное обеспечение вызывается из командной строки.
  Пример: редактор кода (vi, Emacs, Notepad++, atom, VS Code), набор программ для компиляции и отладки (gcc, clang, armcc), программа для сборки (make), отладчик (OpenOCD).

Интегрированные среды разработки для Cortex-M

Коммерческие среды:

Keil Microcontroller Development Kit (MDK-ARM)

IAR Embedded Workbench for ARM Cortex-M

Atmel Studio (только для Cortex-M от Atmel)

Texas Instruments Code Composer Studio (только для Cortex-M от TI)

Open-source среды:

VS Code + GCC (GNU Compiler Collection) + OpenOCD (open on-chip debugger)

Система контроля версий

Зачем отслеживать изменения?

Чтобы искать ошибки в новых версиях

Как хранить изменения?

Файлы/директории с версией/датой в названии

Как синхронизировать изменения при работе в команде?

Хранилище файлов – Google Drive, Dropbox, … или общая (shared) директория

Для отслеживания и синхронизации изменения применяют системы контроля версий:

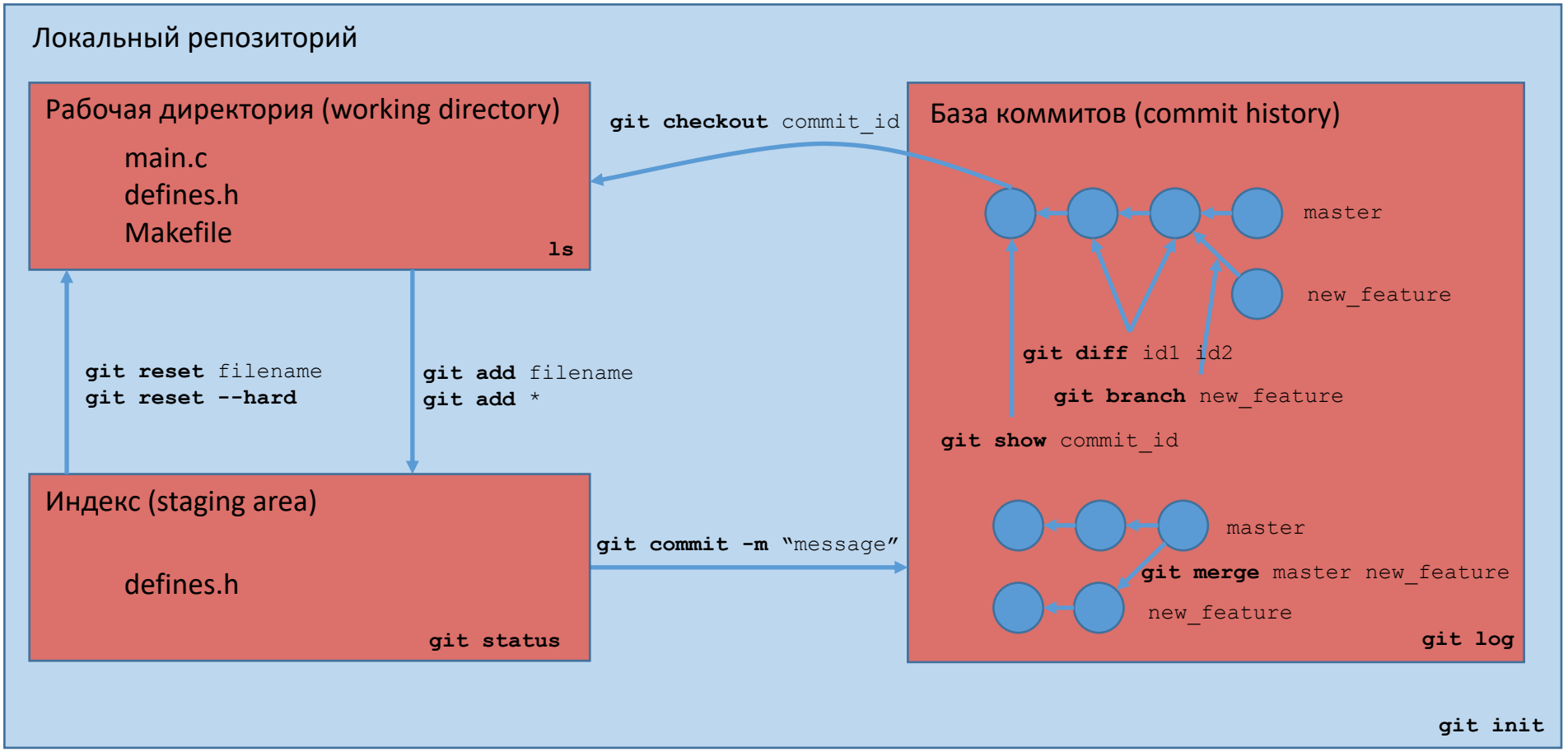
git – распределенная система (наиболее популярная)

svn – централизованная система

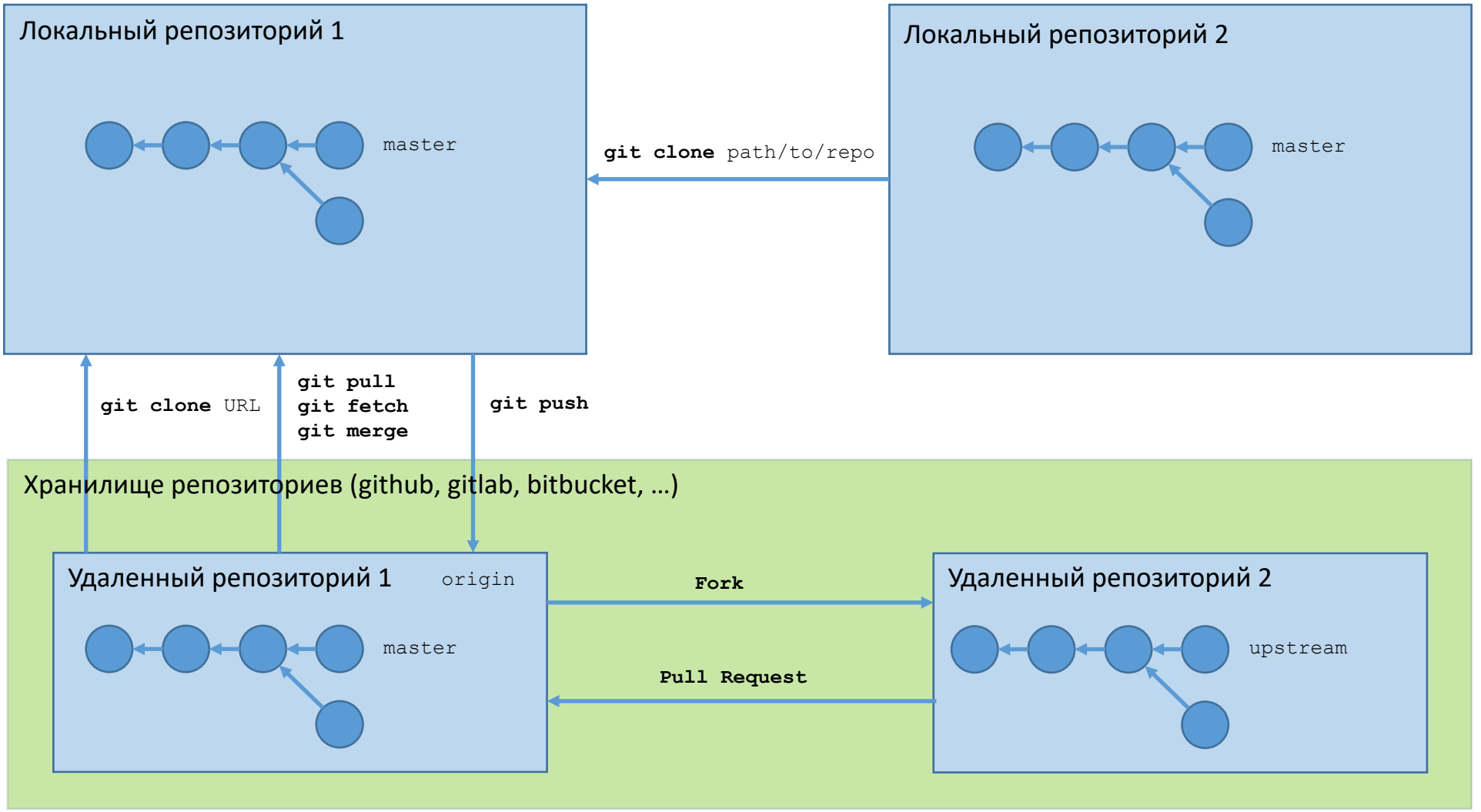
hg – гибридная система

Основной способ работы с системой контроля версия это командная строка, но существуют графические интерфейсы, например TortoiseGit или плагины для редакторов кода.

Система контроля версий git



Система контроля версий git



Система контроля версий git

Добавить в глобальные настройки имя и email,
которые будут ассоциироваться с вашими коммитами:

```
git config --global user.name "Sam Smith"
git config --global user.email sam@example.com
```

Создать новый локальный репозиторий в текущей директории:

```
git init .
```

Сделать копию локального репозитория:

```
git clone /path/to/repository
```

Сделать копию удаленного репозитория:

```
git clone username@host:/path/to/repository
```

Добавить один или все файлы в индекс:

```
git add <filename>
git add *
```

Сделать коммит в локальный репозиторий (но не в удаленный):

```
git commit -m "Commit message"
```

Сделать коммит и добавление всех изменений в индекс одной командой:

```
git commit -a
```

Отравить изменения ветки master в удаленный репозиторий origin:

```
git push origin master
```

Список файлов в которых есть изменения и которые нужно
добавить в индекс или сделать коммит:

```
git status
```

Добавление удаленного репозитория по имени origin или upstream:

```
git remote add origin <server>
git remote add upstream <server>
```

Список всех удаленных репозиториев:

```
git remote -v
```

Создать новую ветку и переключиться на нее:

```
git checkout -b <branchname>
```

Переключиться на другую ветку:

```
git checkout <branchname>
```

Список всех веток в локальном репозитории и указание на текущую ветку:

```
git branch
```

Удалить ветку:

```
git branch -d <branchname>
```

Отправить ветку в удаленный репозиторий origin:

```
git push origin <branchname>
```

Отправить все ветки в удаленный репозиторий origin:

```
git push --all origin
```

Удалить ветку в уделенном репозитории origin:

```
git push origin :<branchname>
```

Получить изменения из удаленного репозитория origin и объединить с
рабочей директорией:

```
git pull origin
```

Система контроля версий git

Получить изменения с удаленного репозитория origin:

```
git fetch origin
```

Объединить текущую активную ветку с branchname:

```
git merge <branchname>
```

Показать все конфликты слияния:

```
git diff
```

Показать конфликты слияния для базового файла:

```
git diff --base <filename>
```

Предпросмотр изменений перед слиянием:

```
git diff <sourcebranch> <targetbranch>
```

После ручное разрешения конфликтов слияния файл нужно добавить в индекс:

```
git add <filename>
```

Коммитам можно добавлять тэги, например о том, что был релиз:

```
git tag 1.0.0 <commitID>
```

Посмотреть ID коммитов:

```
git log
```

Отправить все тэги в удаленный репозиторий origin:

```
git push --tags origin
```

Изменения которые были добавлены в индекс, так же как и новые файлы будут сохранены:

```
git checkout -- <filename>
```

Подтянуть все изменения из удаленного репозитория и сбросить локальный репозиторий до последнего коммита:

```
git fetch origin
```

```
git reset --hard origin/master
```

Поиск в рабочей директории слова foo():

```
git grep "foo()"
```

Свойства программ симуляторов и аппаратных средств отладки

Основные свойства:

1. Загрузка программы в память;
2. Чтение содержимого любой ячейки памяти и любого регистра ЦП;
3. Изменение содержимого любой ячейки памяти и любого регистра ЦП;
4. Запуск и останов программы в произвольной контрольной точке (точке останова - breakpoint), возможность исполнения программы по шагам.

Дополнительные свойства:

1. Установка множества точек останова;
2. Установка условных контрольных точек (conditional breakpoint) и точек наблюдения (watchpoint);
3. Установка динамических точек останова (count breakpoint);
4. Полная символьная отладка (синхронизация исходных файлов и счетчика команд);
5. Загрузка внешних файлов с данными, сохранение информации в файлы;
6. Графическое отображение данных из памяти.

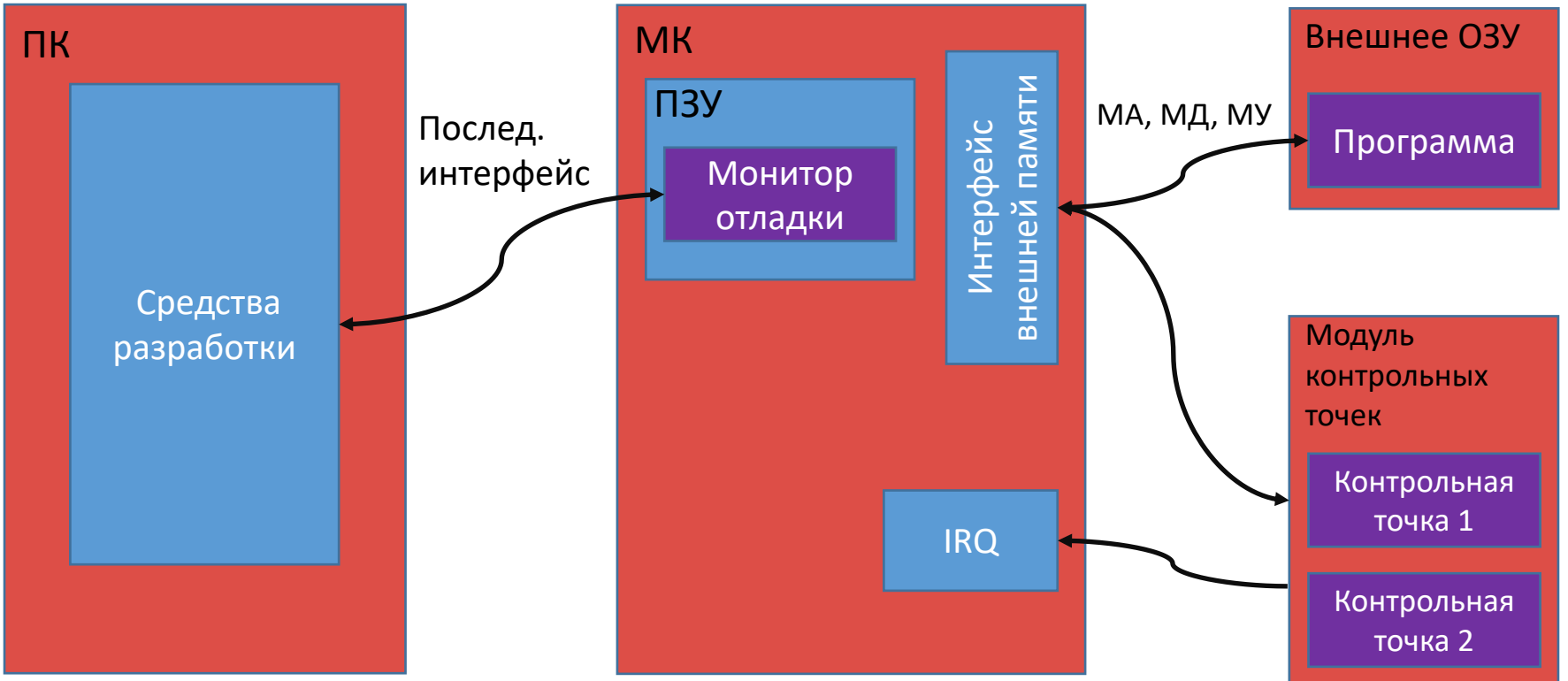
Для симуляторов

1. Симулятор реального времени;
2. Большое количество периферии.

Принципы организации аппаратной отладки

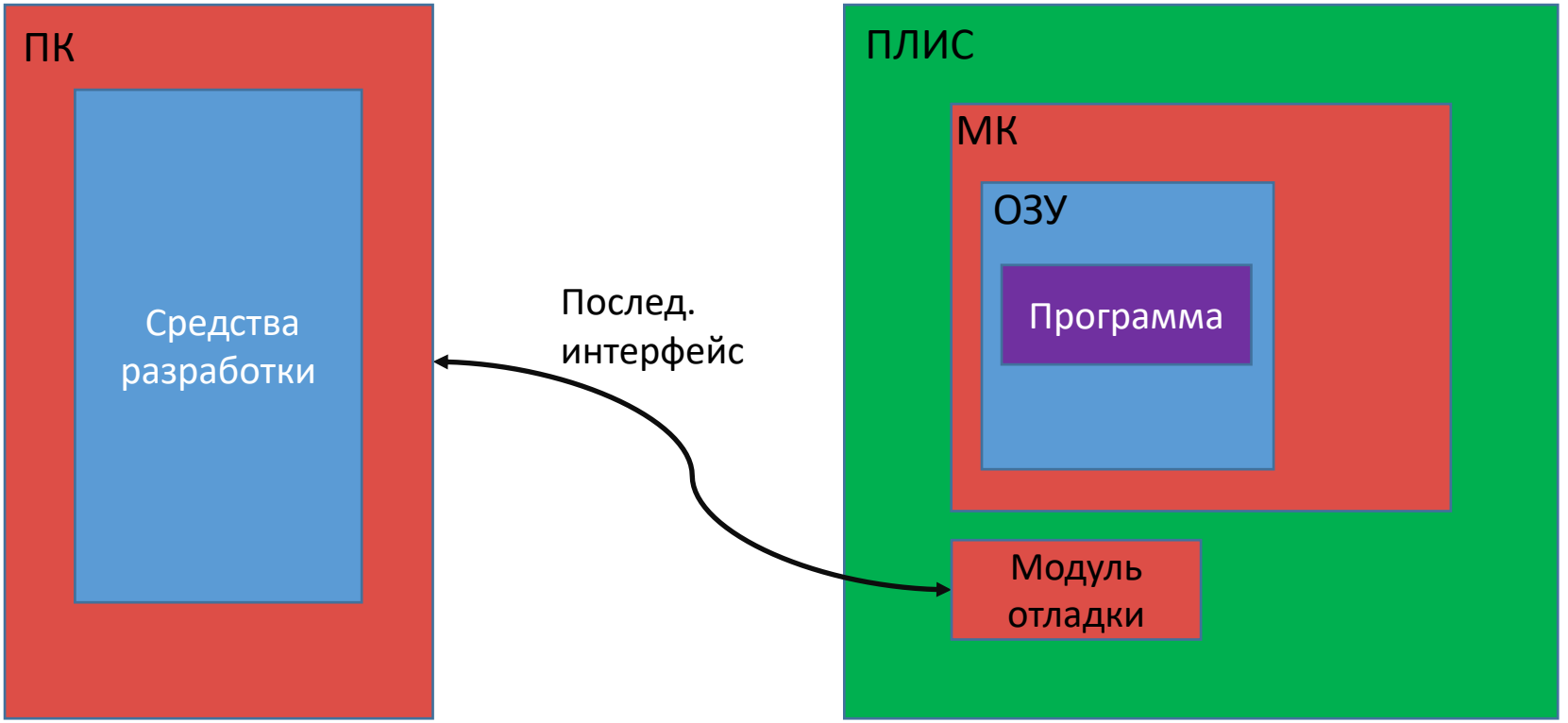
1. Отладка через интерфейс внешней памяти;
2. Отладка на ПЛИС;
3. Внутрисхемная отладка;
4. Отладка через монитор отладки.

Отладка через интерфейс внешней памяти



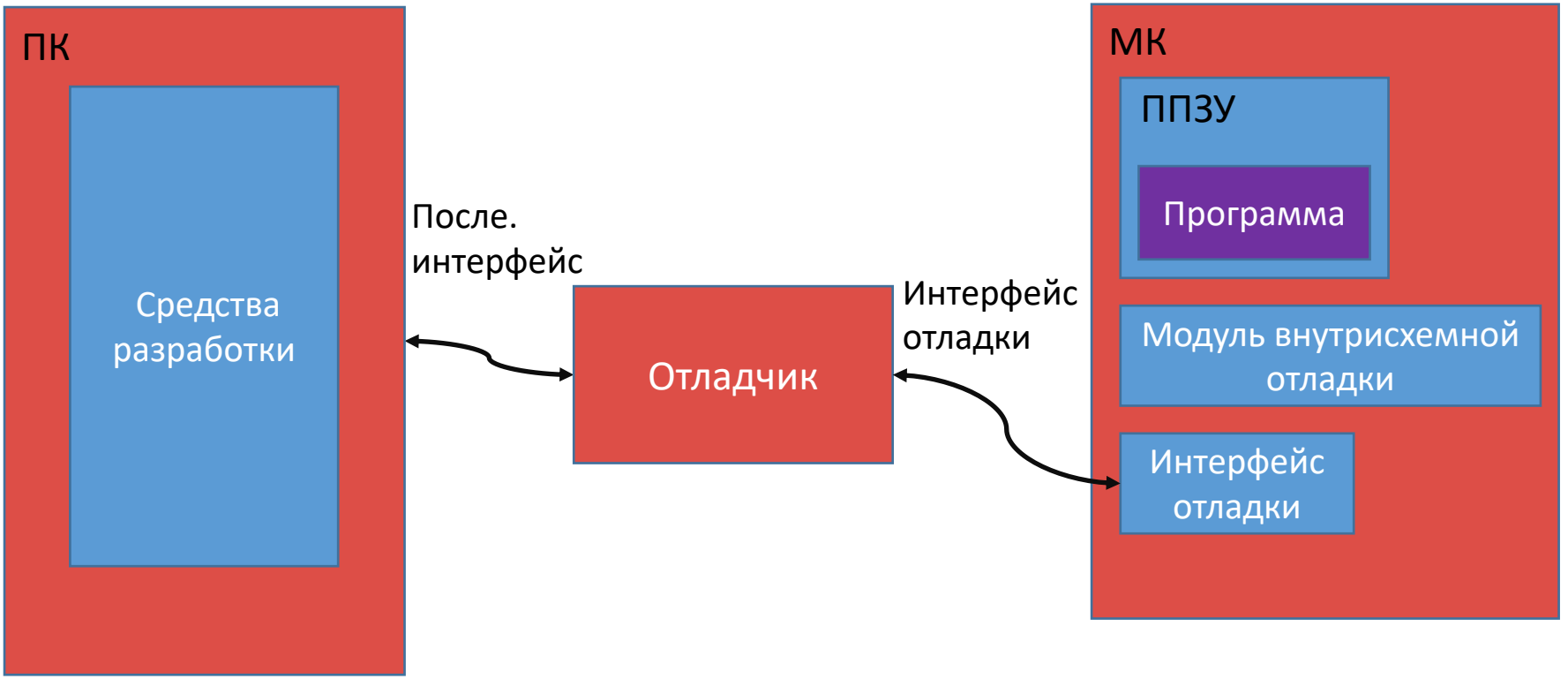
МК имеет интерфейс внешней памяти и способен исполнять программу, размещенную во внешней памяти. В ПЗУ МК располагается программа монитор отладки, которая взаимодействует с персональным компьютером через последовательный интерфейс (USB, UART). Применяется для МК с однократно программируемой или внутренней ПЗУ малого объема.

Отладка на ПЛИС



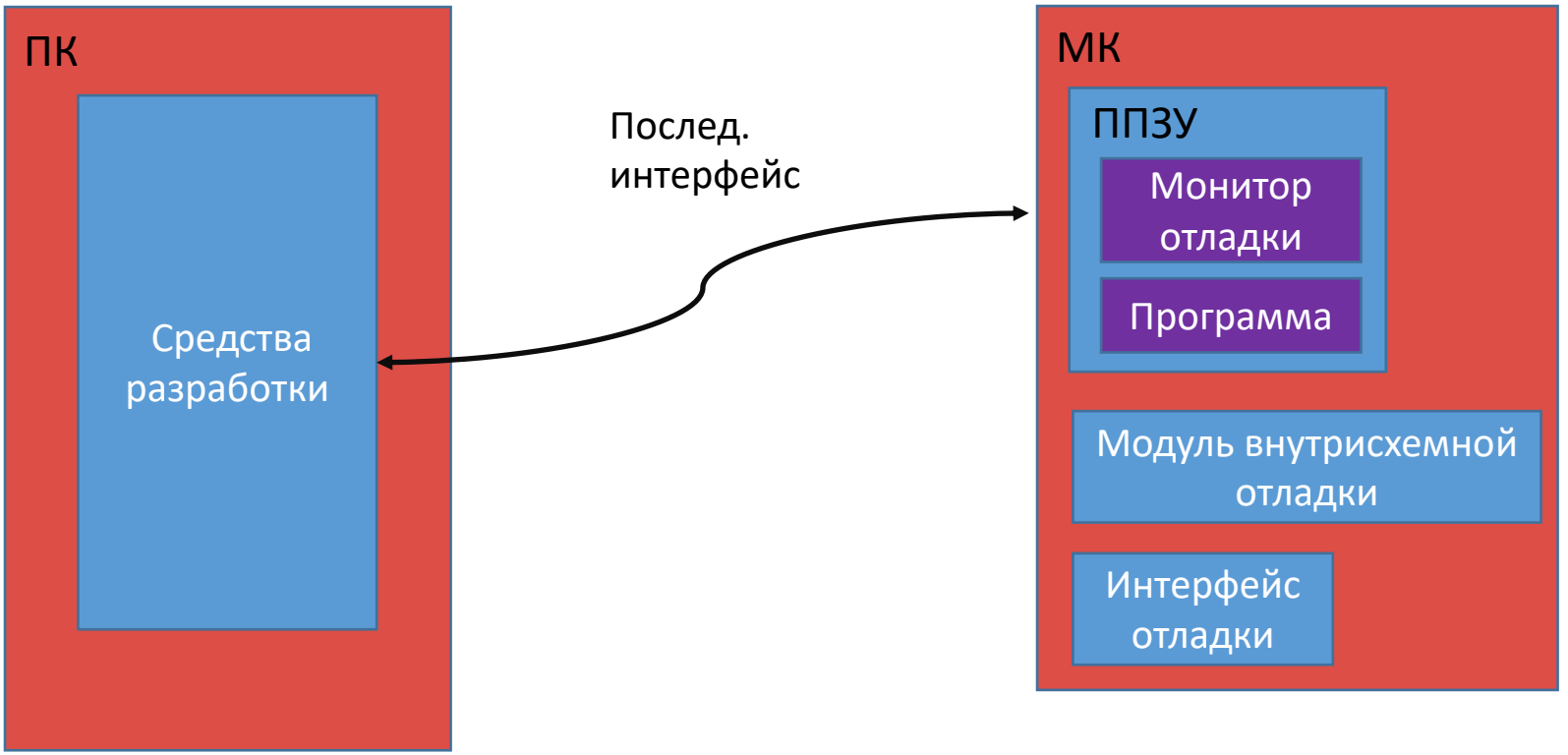
МК (ядро, периферия и память) полностью реализован на ПЛИС. В ПЛИС сконфигурирован модуль отладки, который взаимодействует с персональным компьютером через последовательный интерфейс (USB, UART). Может применяться для отладки массовых микроконтроллеров с однократно программируемой ПЗУ.

Внутрисхемная отладка



В МК реализован специальный модуль внутрисхемной отладки (OCD – on chip debugger). Специальное устройство отладчик (debug adapter) подключается через интерфейс отладки к МК с одной стороны и через последовательный интерфейс (USB) к ПК с другой стороны.

Внутрисхемная отладка с монитором отладки



В МК реализован специальный модуль внутрисхемной отладки, однако он не используется. В МК предварительно загружается программа монитор отладки, которая взаимодействует с персональным компьютером через последовательный интерфейс (USB, UART). Дополнительное устройство отладчик не требуется.

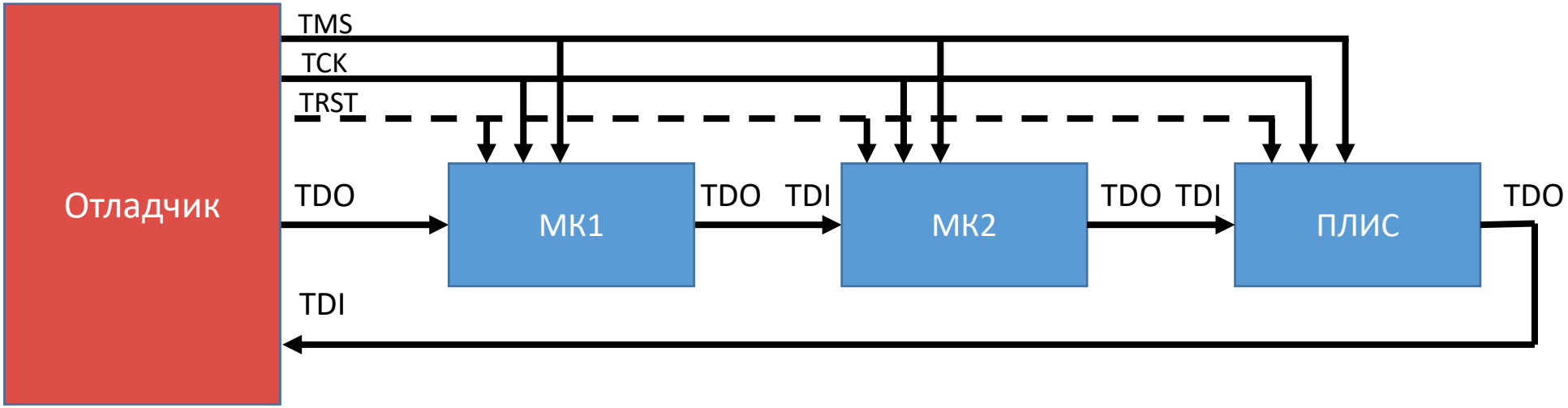
Примеры интерфейсов внутрисхемной отладки

1. **JTAG – Joint Test Action Group**
   Отраслевой стандарт интерфейса внутрисхемной отладки. Применяется во всех современных микроконтроллерах. Изначально разработан для тестирования устройств на плате (пограничное сканирование).
2. **SWD – Serial Wire Debug**
   Интерфейс отладки от компании ARM для МК с ядром Cortex-M.

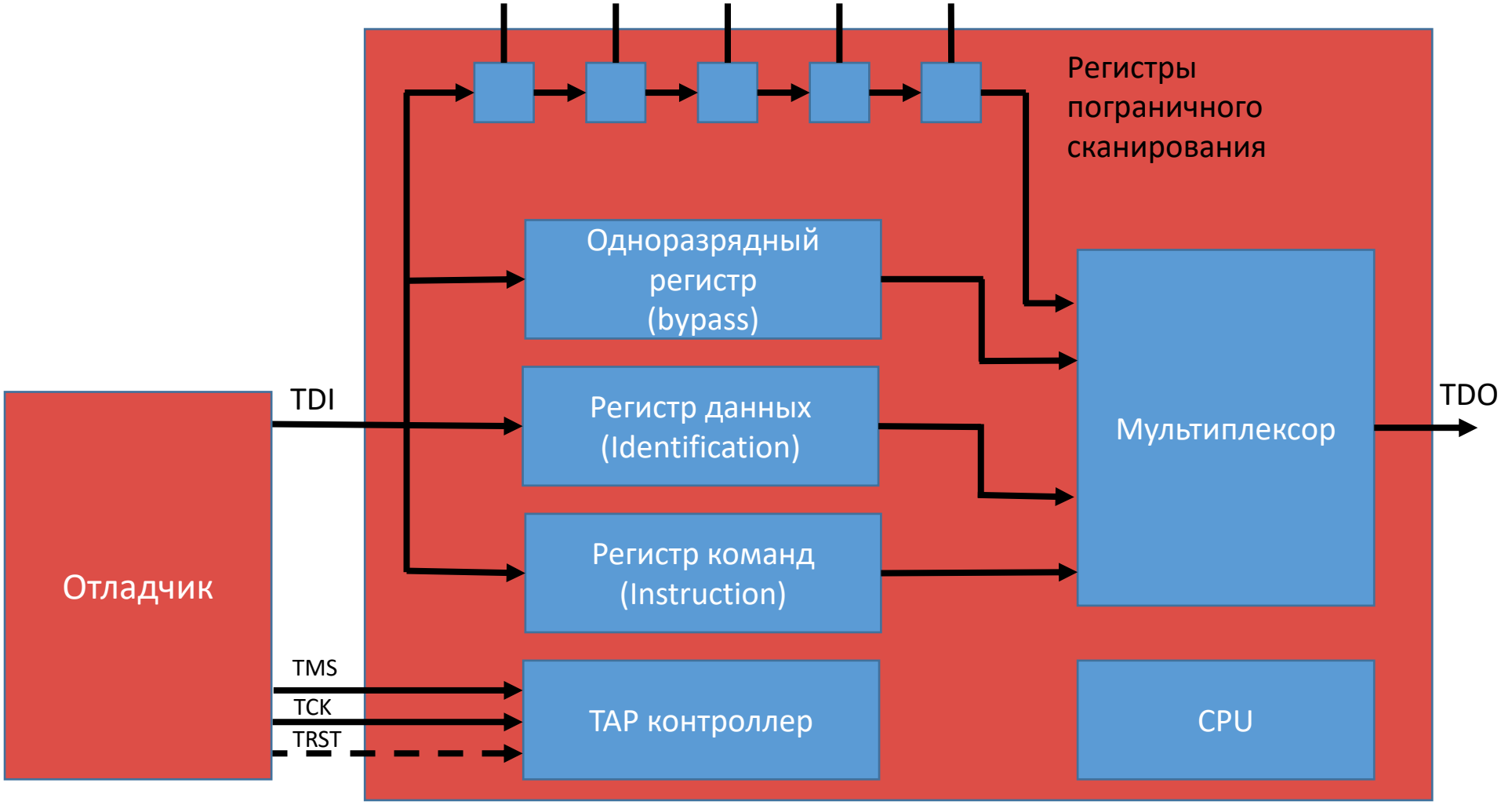
Внутрисхемная отладка: Интерфейс JTAG

Daisy Chaining («Гирлянда») – подключение нескольких устройств к одному отладчику через JTAG

JTAG – Joint Test Action Group
TDI – Test Data Input
TDO – Test Data Output
TMS – Test Mode Select
TCK – Test Clock
TRST – Test Reset



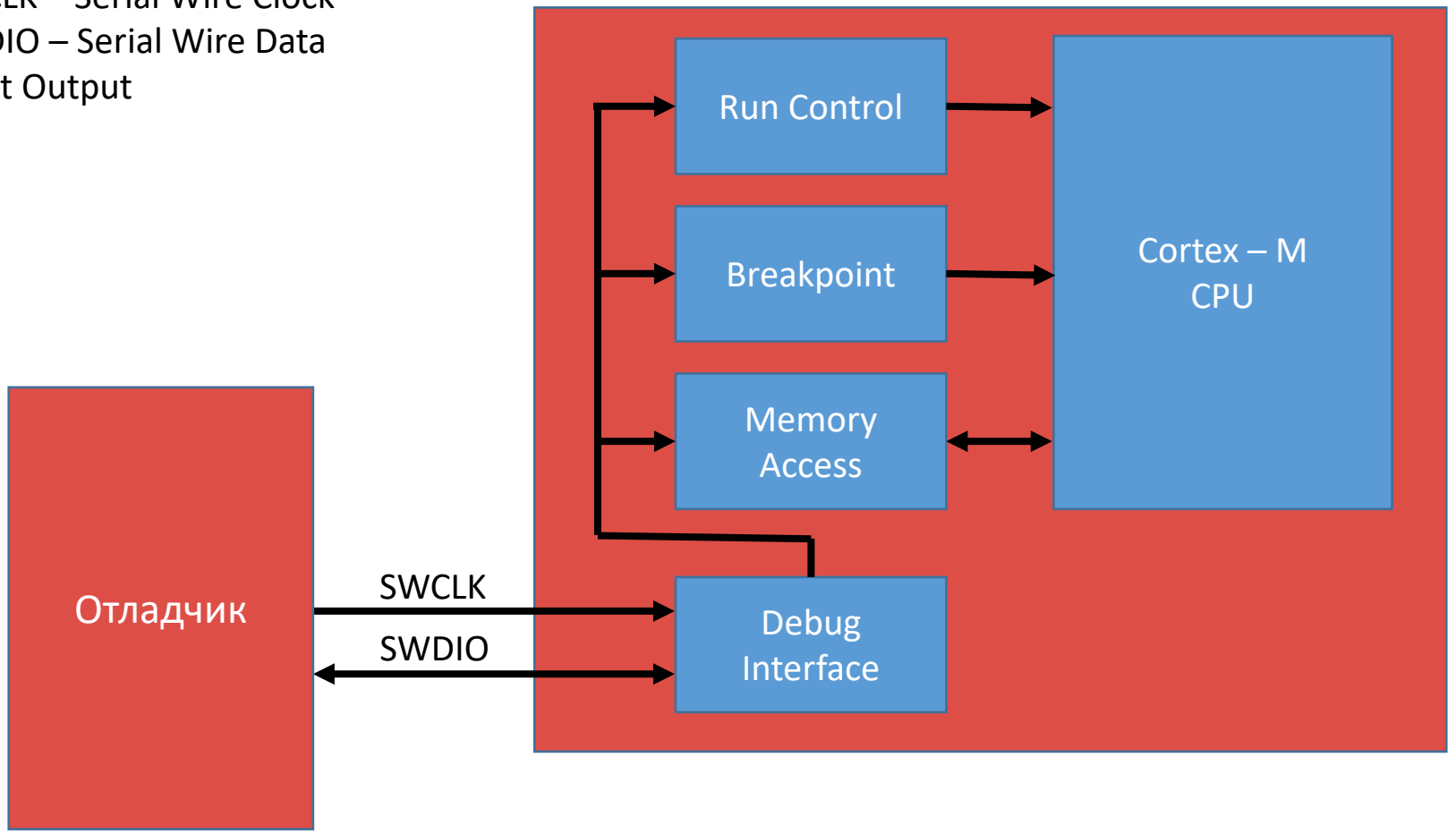
Внутрисхемная отладка: Интерфейс JTAG



Внутрисхемная отладка: Интерфейс SWD

SWD – Serial Wire Debug ARM Cortex-M

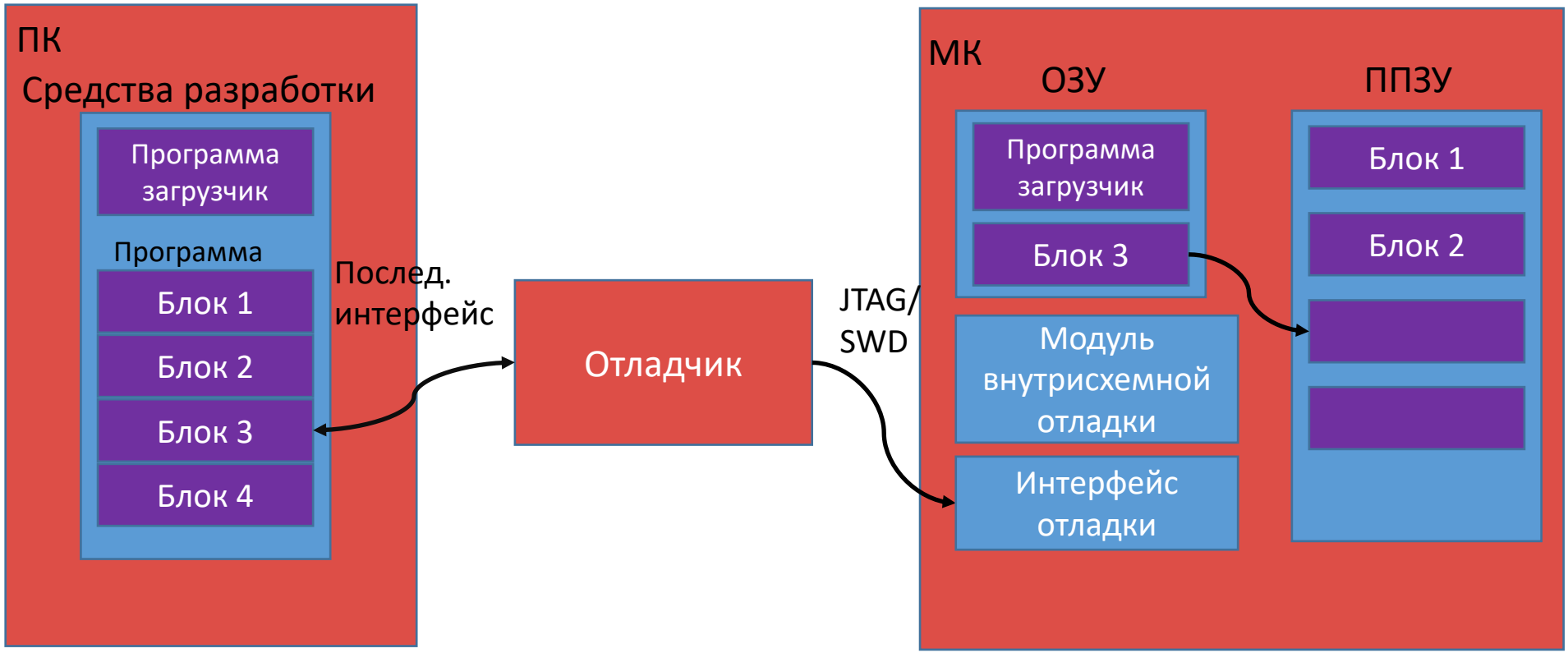
SWCLK – Serial Wire Clock
SWDIO – Serial Wire Data
Input Output



Загрузка программы

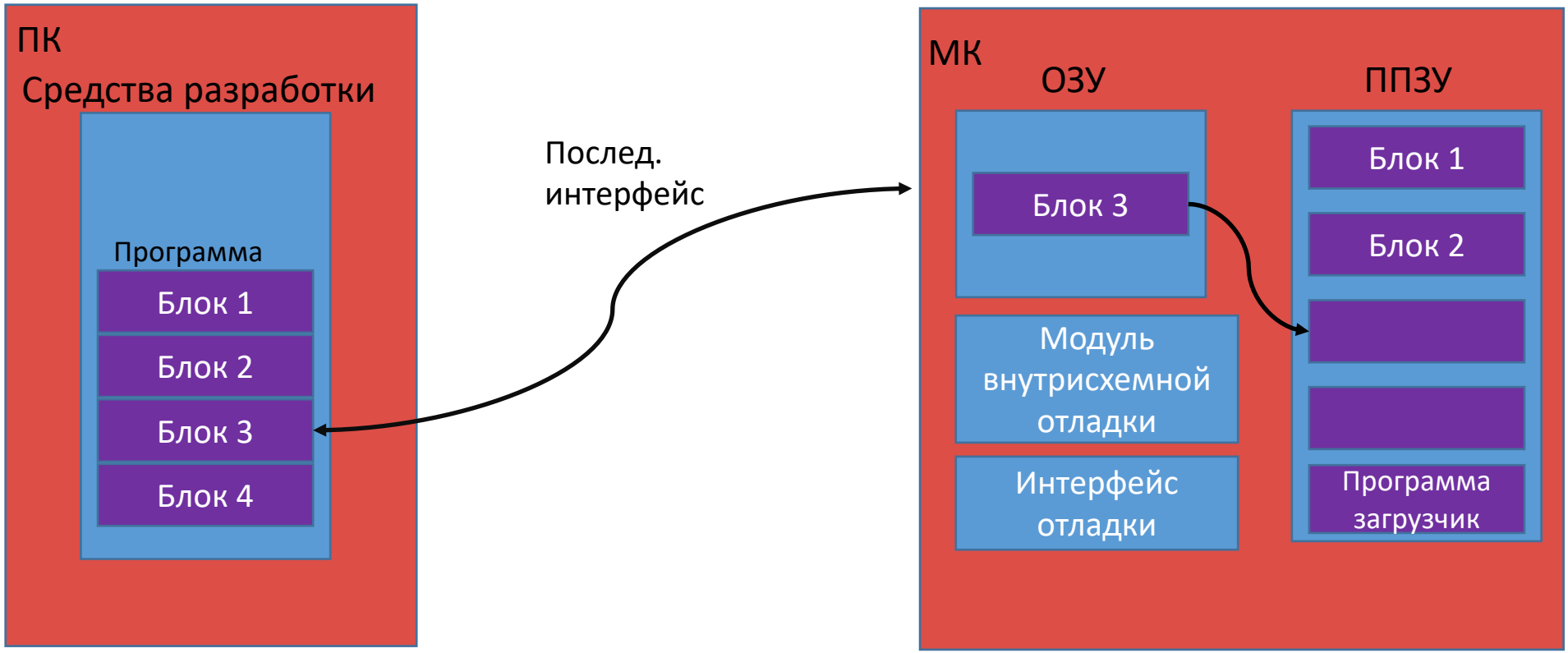
1. Через внутрисхемный отладчик и программу загрузчик;
2. Через последовательный интерфейс и программу загрузчик (bootloader);
3. Непосредственно в интегральную схему памяти с помощью специального программатора для МК с внешней памятью.

Загрузка программы в микроконтроллер через внутрисхемный отладчик



Отладчик загружает в ОЗУ МК специальную программу загрузчик, которая осуществляться программирование ППЗУ МК. Вся программа может не поместиться в ОЗУ МК, поэтому программирование идет блоками.

Загрузка программы в микроконтроллер через загрузчик



Предварительно (производителем или разработчиком) в МК загружена программа загрузчик (bootloader), которая с помощью последовательного интерфейса загружает пользовательскую программу в ОЗУ и программирует ППЗУ.

Трассировка

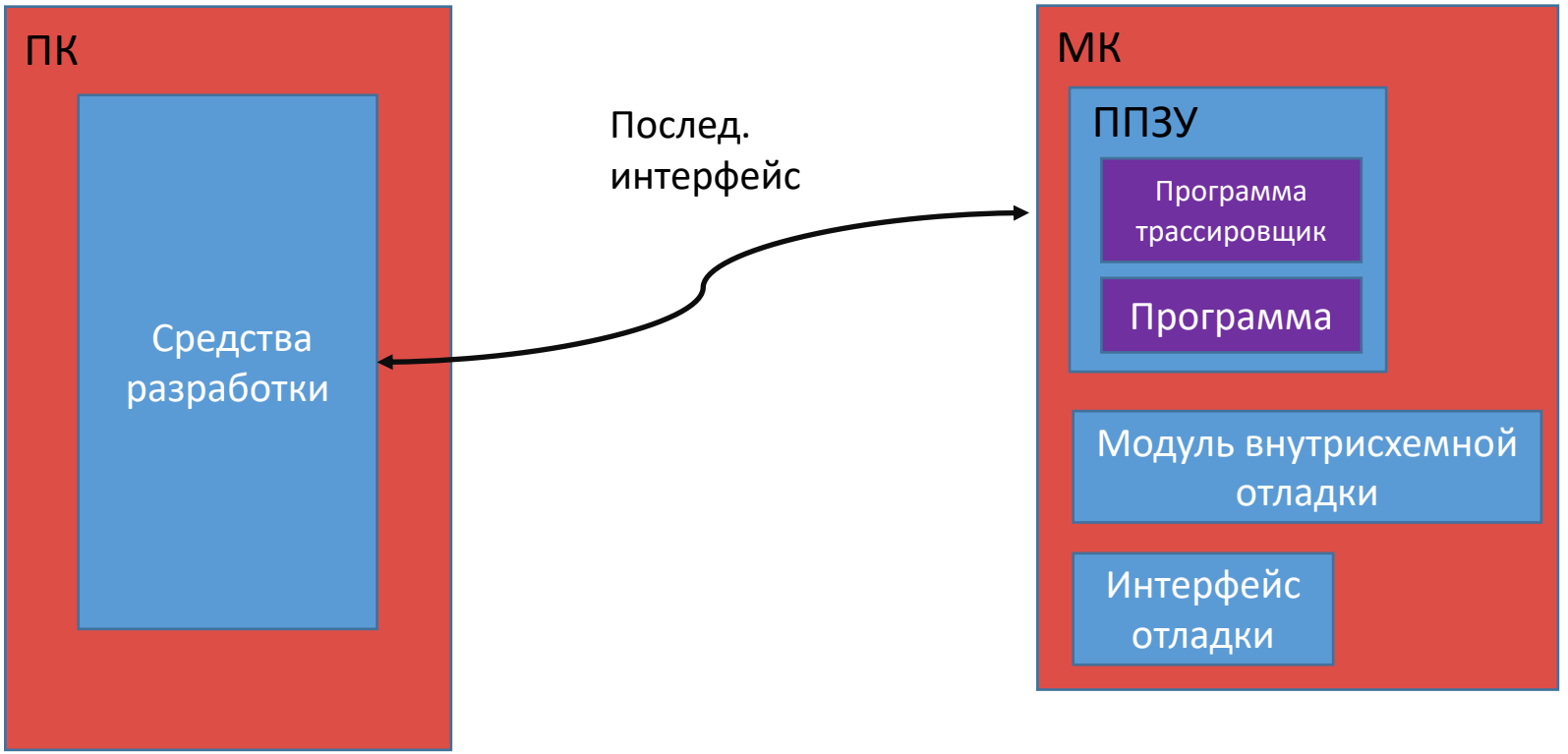
Трассировка – дает дополнительные возможности при отладке:

1. Печать пользовательской отладочной информации (printf, trace, log);
2. Подсчет количества и длительности выполнения прерываний (Exception Trace);
3. Профилирование - время исполнения подпрограмм (Profiling);
4. Измерение покрытия кода - статистика использования кода (Code Coverage);
5. Отображение потока исполнения программы (Instruction Trace);
6. Отображение памяти в реальном времени (Access Data).

Средства трассировки:

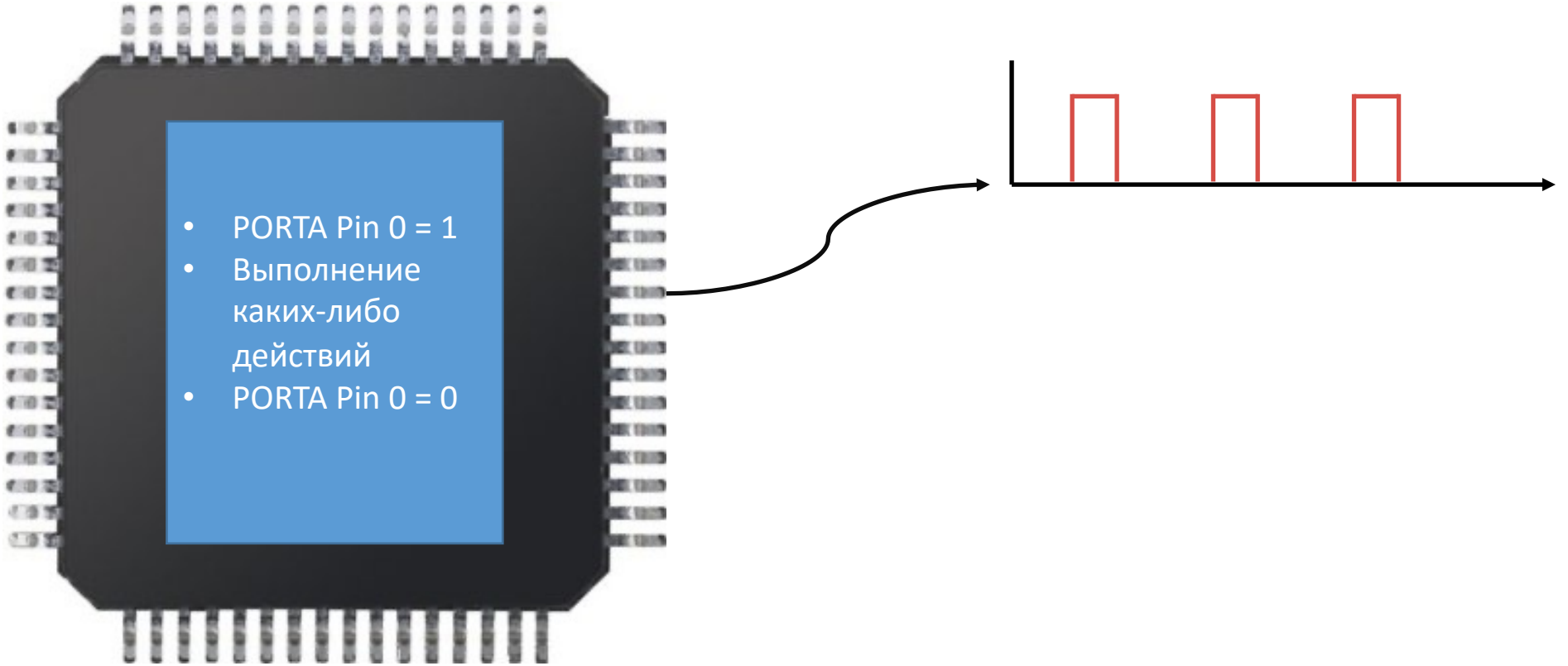
1. Через программу трассировки;
2. Через порты ввода/вывода общего назначения;
3. Через аппаратный трассировщик.

Программная трассировка



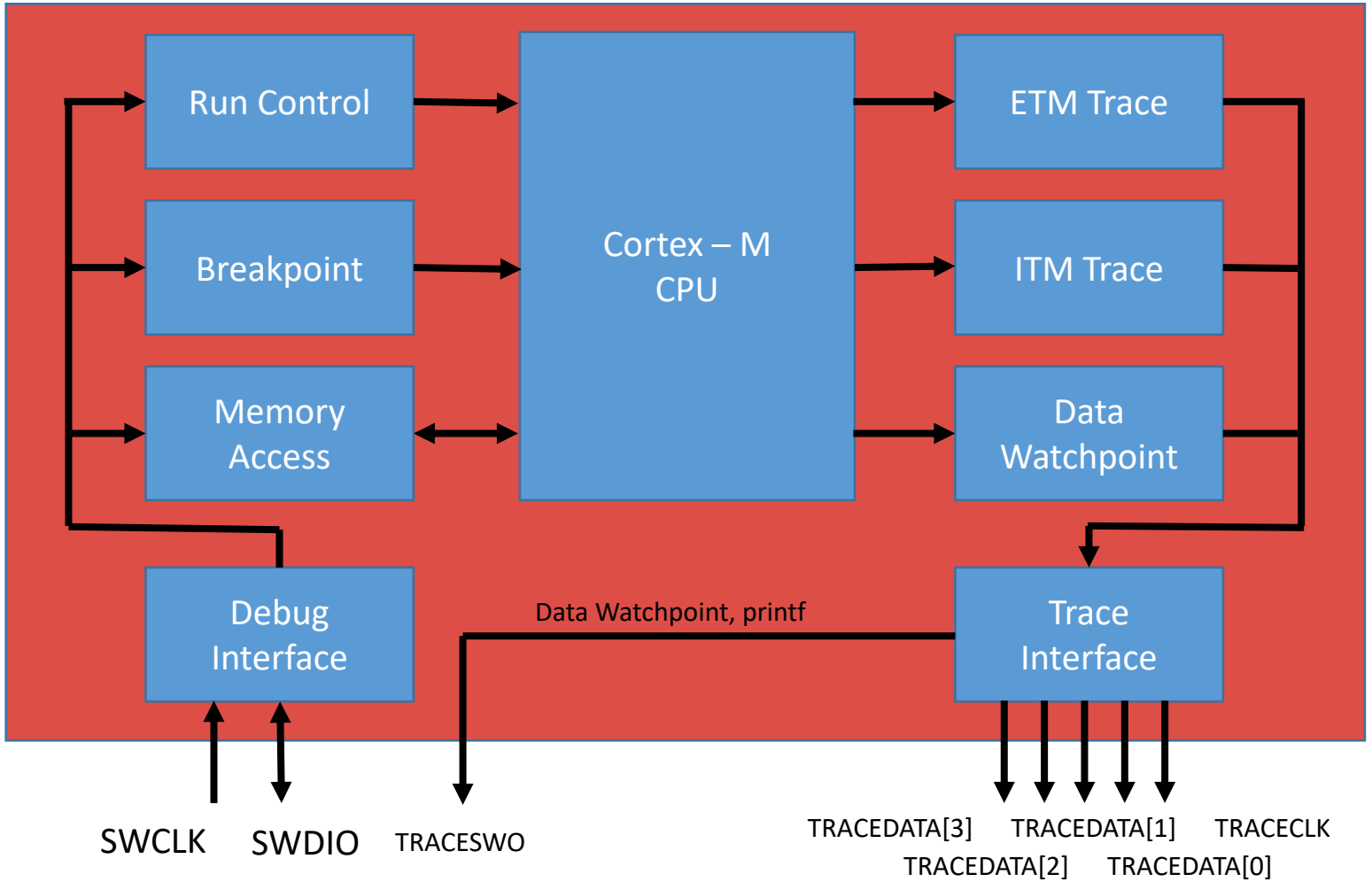
В МК загружается программа трассировщик. Простейшим трассировщиком являются функции текстового ввода/вывода (printf, scanf) через последовательный интерфейс (UART, USB). Остальные задачи трассировки (подсчет количества прерываний, профилирование и т.д.) можно решить добавлением переменных счетчиков и периодическим выводом значений через printf. Основной недостаток: функции текстового ввода/вывода влияют на время исполнения программы (медленный UART и USB).

Трассировка через порты ввода/вывода общего назначения



Трассировка через порты ввода/вывода позволят проводить профилирование, подсчет и фиксацию событий практически не влияя на ход исполнения программы.

Аппаратная трассировка: SWD, TRACEPORT



В МК реализован модуль и интерфейс трассировки. Для подключения к ПК требуется специальное устройство отладчик/трассировщик. Можно проводить трассировку в реальном времени не влияя на ход выполнения программы.

Загрузка/отладка минимальными аппаратными средствами

1. Отладка/трассировка:
   - В реальном масштабе времени: светодиоды, свободные порты ввода вывода + осциллограф;
   - В остальных случаях: средства ввода/вывода текстовой информации, например, printf и scanf через последовательный интерфейс UART или USB.

2. Загрузка программы через программу-загрузчик (bootloader):
   - Современные МК имеют прошитый на заводе программу загрузчик (bootloader). Активация программы производится установкой комбинации логических сигналов на портах ввода/вывода при подаче питания. Интерфейсом взаимодействия могут являться UART, USB, CAN, Ethernet.

Заключение

1. Выбор инструментального программного обеспечения (IDE или toolchain) зависит от многих факторов: скорость разработки, бюджет, квалификация разработчика;
2. По возможности следует избегать отладки (исполнение кода по шагам, контрольные точки и т.д.). Исполнение программы по шагам и применение точек останова во встраиваемых системах реального времени может приводить к серьезным авариям при отладке;
3. Если отладка необходима, то в большинстве случаев достаточно отладки при помощи портов ввода/вывода (для систем реального времени) и печати текстовой информации (для всего остального);
4. Применение аппаратных отладчиков требуется в исключительных случаях, когда печать текстовой информации вносит недопустимые изменения в ход выполнения программы, а информации полученной через порты ввода/вывода недостаточно для выявления ошибки.

Правильно спроектированное программное обеспечение (декомпозиция на подпрограммы, модули с подпрограммами, уровни абстракции) с максимально полным покрытием программы тестами позволяет создавать надежное программное обеспечение не прибегая к отладке.