

Лекция 6 Разработка и отладка программ для встраиваемых систем

План курса «Встраиваемые микропроцессорные системы»:

Лекция 1: Введение. Язык программирования C

Лекция 2: Язык программирования C. Стандартная библиотека языка C

Лекция 3: Применение языка C для встраиваемых систем

Лекция 4: Микроконтроллер

Лекция 5: Этапы разработки встраиваемых систем

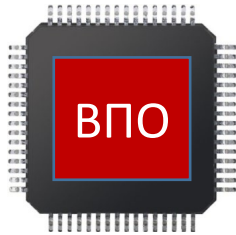
Лекция 6: Разработка и отладка программ для встраиваемых систем

Лекция 7: Архитектура программ для встраиваемых систем

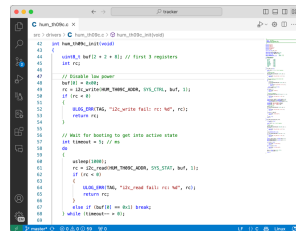
Лекция 8: Периферийные модули: DMA, USB, Ethernet

Программное обеспечение, используемое при разработке встраиваемой системы

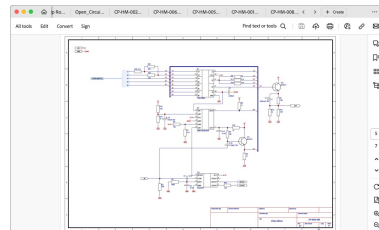
1. Встраиваемое программное обеспечение (ВПО) – программное обеспечение для исполнения в микроконтроллере;



2. Инструментальное программное обеспечение – комплекс программ для разработки и отладки встраиваемого программного обеспечения, например компилятор, редактор кода;



3. Программное обеспечение общего назначения – вспомогательное программное обеспечение, например браузер, схемный редактор.



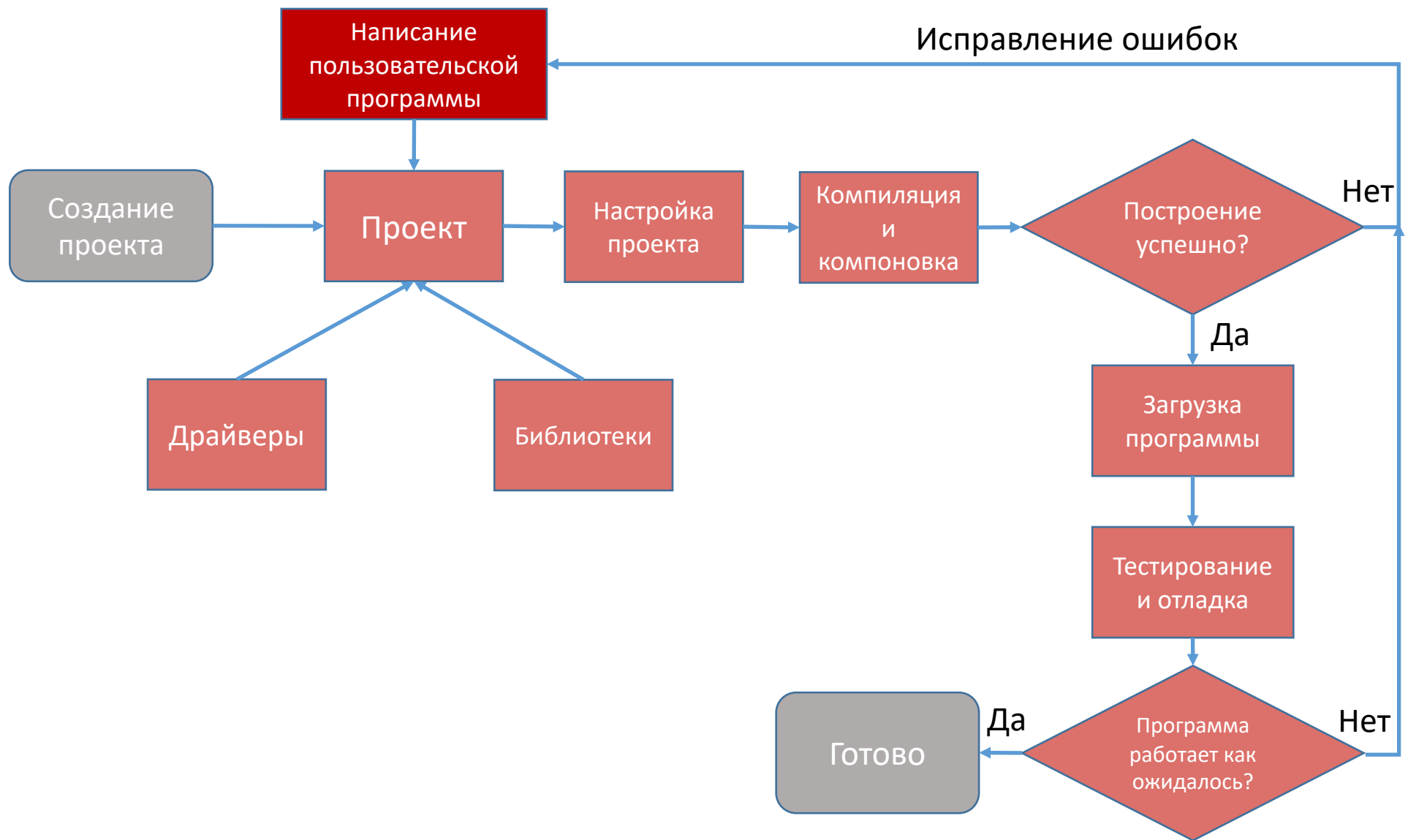
Инструментальное программное обеспечение

1. Текстовый редактор (text editor);
2. Ассемблер (assembler);
3. Компилятор (compiler);
4. компоновщик (линковщик, linker);
5. Генератор исходных текстов (source code generator);
6. Большая языковая модель (LLM);
7. Менеджер проекта;
8. Система контроля версия (version control);
9. Отладчик (debugger);
10. Симулятор (simulator);
11. Драйвер для отладочного устройства/программатора (adapter driver);
12. Графические интерпретаторы отладки/симуляции.

Аппаратные инструменты, используемые при разработке встраиваемой системы

1. Отладочные платы и макетные платы;
2. Отладчик/программатор (debug adapter);
3. Преобразователи интерфейсов (UART-USB, RS232-USB, CAN-USB);
4. Источник питания;
5. Мультиметр;
6. Осциллограф;
7. Логический анализатор;
8. Паяльная станция.

Упрощенный процесс разработки встраиваемого программного обеспечения



Пример: средства для разработки и отладки микропроцессорной системы для K1986BE92QI

1. Средства разработки программного обеспечения (интегрированная среда разработки Keil MDK-ARM или набор средств разработки (toolchain));
2. Отладочная плата или прототип устройства (отладочная плата K1986BE92QI);
3. Программатор/отладчик (Phyton JEM-ARM-V2);
4. Драйверы периферии микроконтроллера (Standard Peripheral Library для K1986BE92QI);
5. Примеры;
6. Документация;
7. Дополнительное оборудование (преобразователи интерфейсов (USB-RS232), осциллограф, генератор импульсов, источники питания и т.д.).

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.

Примеры коммерческих сред для Cortex-M:

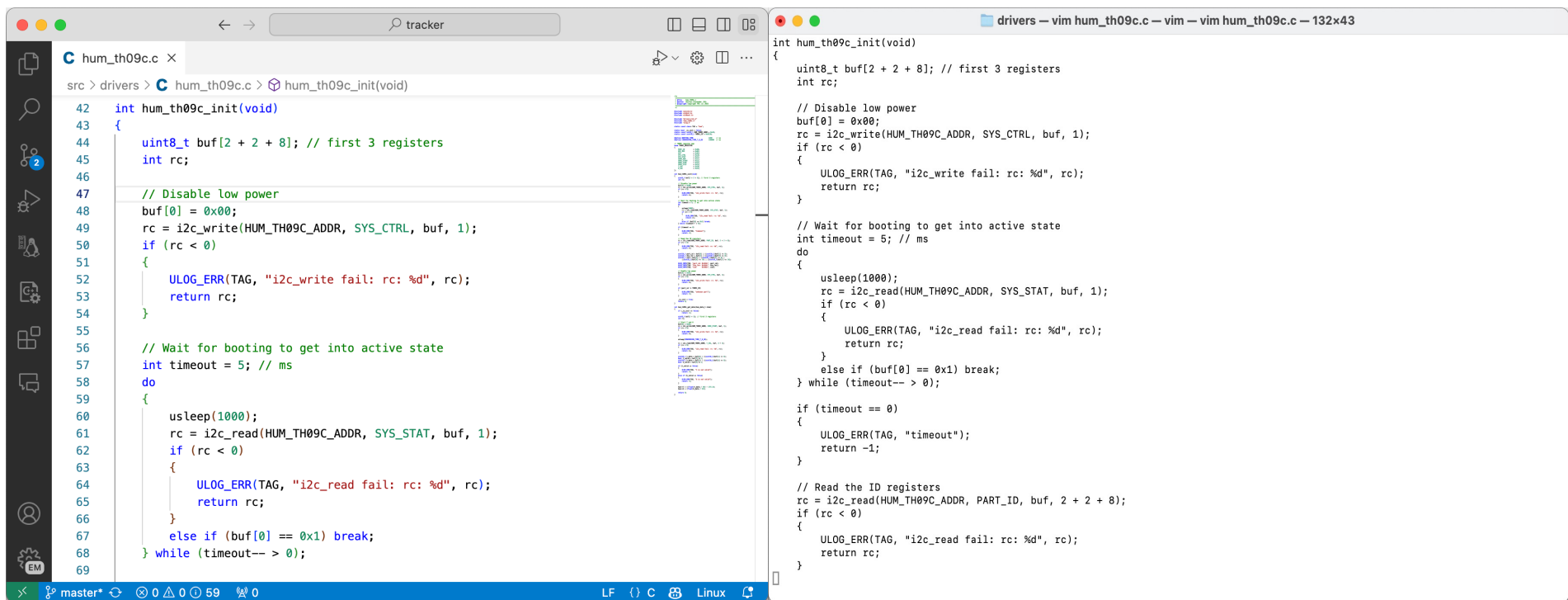
- Keil Microcontroller Development Kit (MDK-ARM);
 - IAR Embedded Workbench for ARM Cortex-M;
 - Atmel Studio (только для Cortex-M от Atmel);
 - Texas Instruments Code Composer Studio (только для Cortex-M от TI).
- **Редактор кода и сборка проекта в командной строке.**
Запуск проекта в утилите make по 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).

Текстовый редактор

Современные текстовые редакторы имеют режим подсветки синтаксиса (ключевых слов) выбранного языка.

Также до сих пор применяются консольные текстовые редакторы (vim, nano).



```
src > drivers > C hum_th09c.c > vim hum_th09c_init(void)
42 int hum_th09c_init(void)
43 {
44     uint8_t buf[2 + 2 + 8]; // first 3 registers
45     int rc;
46
47     // Disable low power
48     buf[0] = 0x00;
49     rc = i2c_write(HUM_TH09C_ADDR, SYS_CTRL, buf, 1);
50     if (rc < 0)
51     {
52         ULOG_ERR(TAG, "i2c_write fail: rc: %d", rc);
53         return rc;
54     }
55
56     // Wait for booting to get into active state
57     int timeout = 5; // ms
58     do
59     {
60         usleep(1000);
61         rc = i2c_read(HUM_TH09C_ADDR, SYS_STAT, buf, 1);
62         if (rc < 0)
63         {
64             ULOG_ERR(TAG, "i2c_read fail: rc: %d", rc);
65             return rc;
66         }
67         else if (buf[0] == 0x1) break;
68     } while (timeout-- > 0);
69 }
```

```
vim hum_th09c.c — vim — vim hum_th09c.c — 132x43
int hum_th09c_init(void)
{
    uint8_t buf[2 + 2 + 8]; // first 3 registers
    int rc;

    // Disable low power
    buf[0] = 0x00;
    rc = i2c_write(HUM_TH09C_ADDR, SYS_CTRL, buf, 1);
    if (rc < 0)
    {
        ULOG_ERR(TAG, "i2c_write fail: rc: %d", rc);
        return rc;
    }

    // Wait for booting to get into active state
    int timeout = 5; // ms
    do
    {
        usleep(1000);
        rc = i2c_read(HUM_TH09C_ADDR, SYS_STAT, buf, 1);
        if (rc < 0)
        {
            ULOG_ERR(TAG, "i2c_read fail: rc: %d", rc);
            return rc;
        }
        else if (buf[0] == 0x1) break;
    } while (timeout-- > 0);

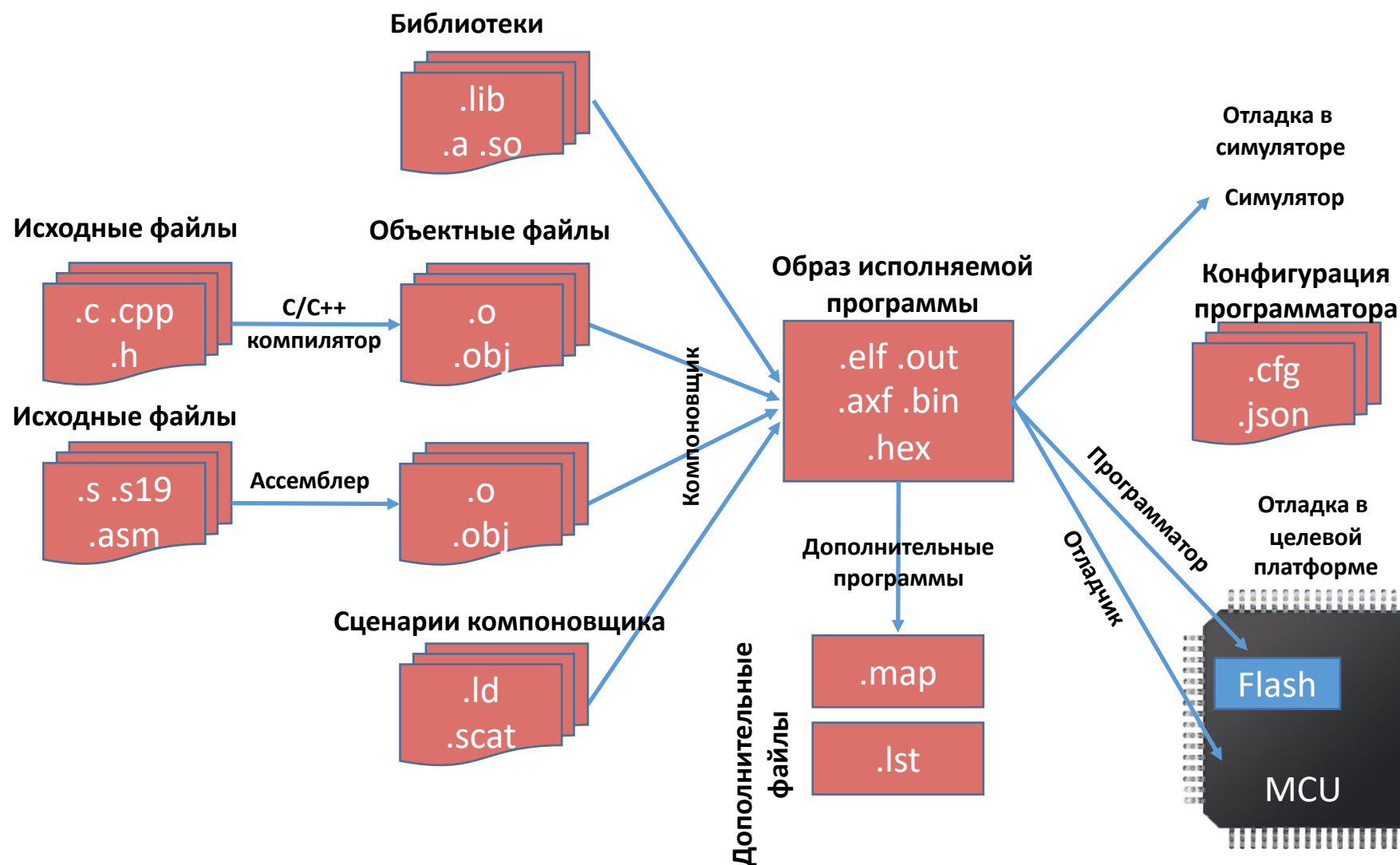
    if (timeout == 0)
    {
        ULOG_ERR(TAG, "timeout");
        return -1;
    }

    // Read the ID registers
    rc = i2c_read(HUM_TH09C_ADDR, PART_ID, buf, 2 + 2 + 8);
    if (rc < 0)
    {
        ULOG_ERR(TAG, "i2c_read fail: rc: %d", rc);
        return rc;
    }
}
```

Visual Studio Code

vim

Процесс сборки встраиваемого программного обеспечения



Компилятор и компоновщик

Компилятор — это программа, которая преобразует исходный код, написанный на одном языке программирования, в объектный код или код на другом языке.

Ассемблер — это программа, преобразующая ассемблерный код в объектный код.

Компоновщик — это программа, которая собирает отдельные модули программы (объектный модули) и библиотеки, совмещая их в один исполняемый файл.

Объектный код — код еще не является полностью готовым для выполнения, так как он может содержать ссылки на внешние ресурсы и библиотеки, которые должны быть разрешены.

Машинный код — это низкоуровневый код, который может быть непосредственно исполнен процессором. Он представляет собой полностью скомпонованную программу, включающую в себя все необходимые модули и ссылки.

Образ (image) – содержит машинный код код всех модулей и библиотек, пригодный для загрузки во встраиваемую систему.

Наборы инструментов

Часто компиляторы, сборщики и другие программы собирают в набор инструментов (toolchain).

Название	GCC	Описание
Компилятор	<code>gcc, g++</code>	Преобразует код на C/C++ в объектный код
Ассемблер	<code>as</code>	Преобразует ассемблерный код в объектный код
Компоновщик	<code>ld</code>	Объединяет несколько объектных файлов и библиотек, разрешает ссылки и создает выходной файл
Binutils	<code>size readelf objdump objcopy strings ar nm</code>	Набор полезных программ. Например, программа <code>size</code> выводит размеры секций образа программы, а программа <code>string</code> выводит все ASCII строки находящиеся в образе.
Runtime library/C standard library	<code>glibc eglibc uClibc</code>	Библиотека исполнения языка C / стандартная библиотека C поставляться отдельно от набора инструментов или может быть заменена на другую.

Кросс компиляторы C

Компиляция для встраиваемых систем требует другого компилятора (**кросс компилятора**). Компиляция исходного кода в самой встраиваемой системе нецелесообразна и часто невозможна.

Для кросс компиляторов gcc есть соглашение о префиксе:

`arch-vendor-(os-)abi-gcc`

Например:

`arm-none-linux-gnueabi-gcc` – ARM архитектура, none – без вендора, linux – операционная система, gnueabi – интерфейс приложений (ABI) для встраиваемого Linux.

`arm-none-eabi-gcc` - ARM архитектура, none – без вендора, интерфейс приложений для встраиваемых систем (Embedded ABI).

`riscv-none-eabi-gcc` – RISC-V архитектура, none – без вендора, интерфейс приложений для встраиваемых систем (Embedded ABI).

Также все остальные компоненты набора инструментов будут иметь префикс.

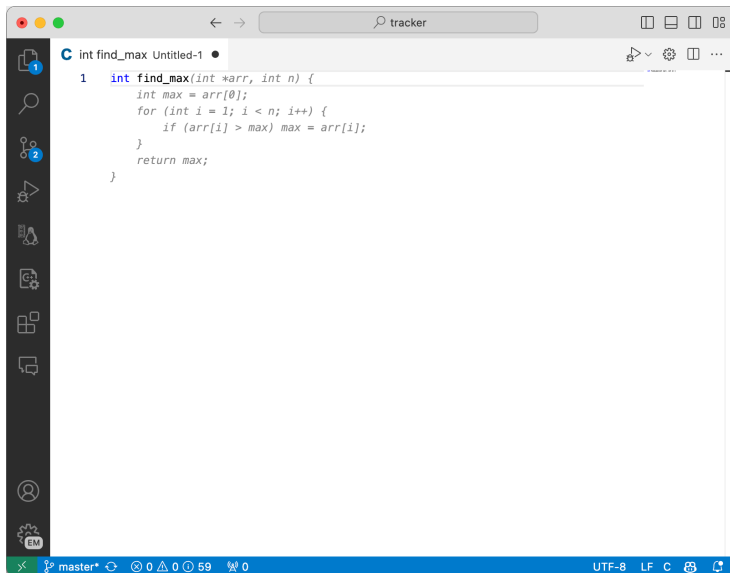
В остальном кросс компилятор работает также как основной компилятор платформы разработки

Генератор исходных текстов

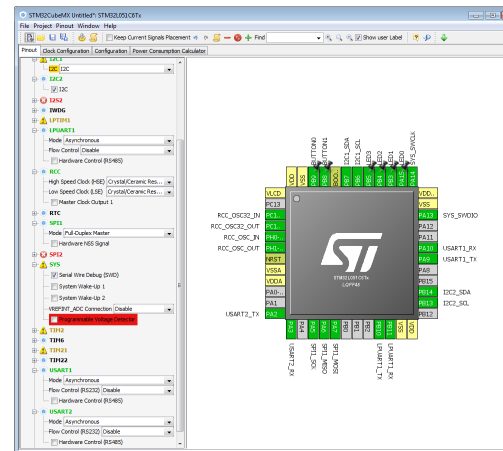
Кодогенераторы на основе шаблонов — они используют шаблоны с плейсхолдерами, которые заполняются данными для генерации кода. Например, STMCubeMX для генерации кода инициализации МК или универсальный шаблонизатор StringTemplate.

Средства автоматизации проектирования программного обеспечения — эти инструменты могут генерировать код из UML-диаграмм или других моделей. Например, Matlab Coder или Matlab Embedded Coder.

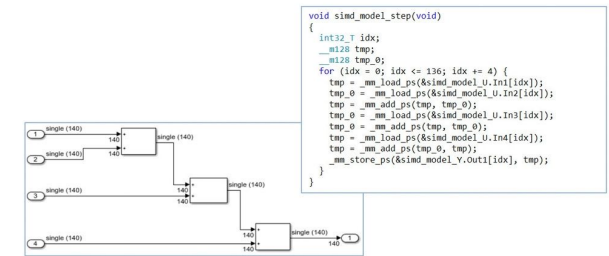
Системы искусственного интеллекта — используют машинное обучение (LLM) для предложения кода исходя из контекста уже написанного кода и других источников. Например, Github Copilot или Tabnine.



GitHub Copilot



STMCubeMX



Matlab Embedded Coder

Менеджер проекта

Менеджер проекта — это компонент или инструмент, который помогает разработчику управлять различными аспектами программного проекта, такими как структура файлов и папок, настройки сборки, управление зависимостями и версиями, а также интеграция с системами контроля версий. Это упрощает навигацию по проекту и его модификацию, а также может включать в себя функции для автоматизации рутинных задач и облегчения совместной работы в команде.

В IDE менеджером проекта является сама графическая оболочка.

Если IDE не применяется, то структура директорий и сценарий сборки Makefile.

Менеджером проекта выполняется **сборка проекта** (build), когда нужные файлы и программы запускаются в нужной последовательности.

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:

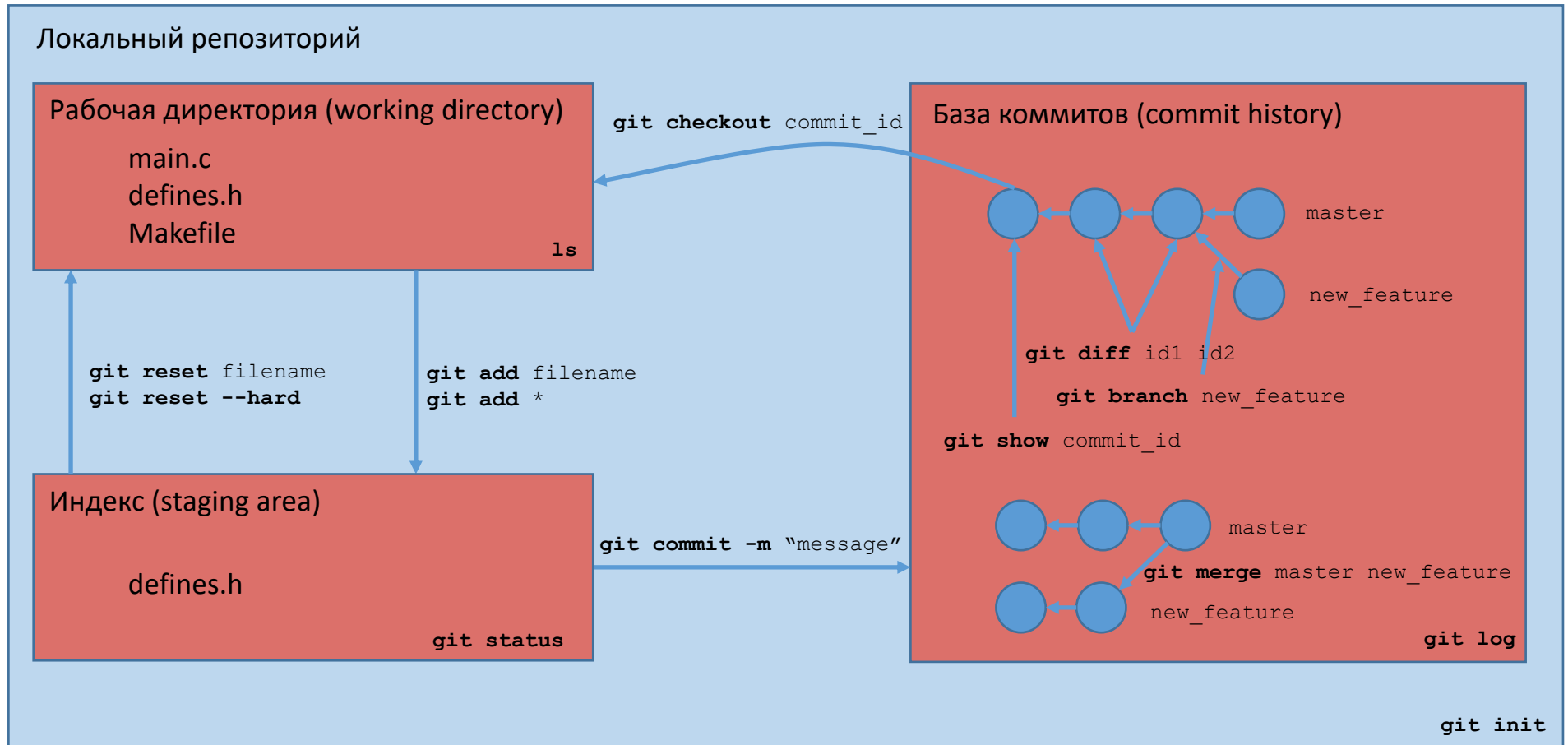
`git` – распределенная система (наиболее популярная)

`svn` – централизованная система

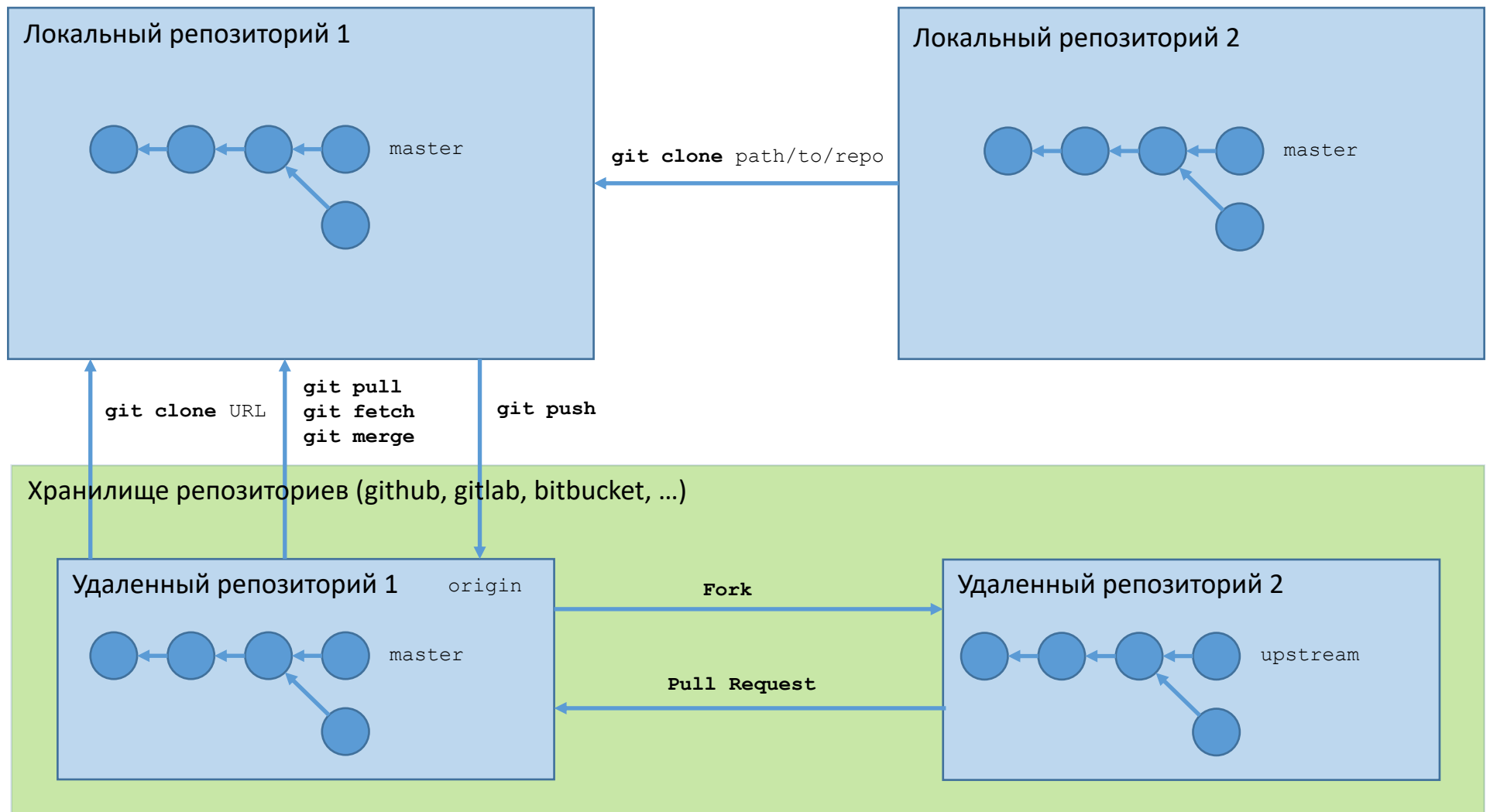
`hg` – гибридная система

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



Отладчик — это программа, предназначенная для тестирования и отладки других программ, позволяя программистам выполнять программу пошагово, просматривать и изменять значения переменных и регистров центрального процессора.

Следует различать программу отладчик (debugger) и адаптер отладки (debug adapter), который также называют отладчиком или программатором.

Симулятор — это программа, которая создаёт приближенное к реальности виртуальное окружение, имитирующее поведение программы или устройства. Симуляторы обычно используются для моделирования работы аппаратного обеспечения, операционных систем, сетей или других программных сред, что позволяет разработчикам тестировать и отлаживать свои приложения в контролируемой среде без необходимости использования реального оборудования.

Программа отладчик подключается к симулятору или к адаптеру отладки для выполнения отладки программного обеспечения.

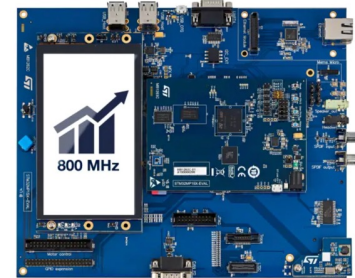
Отладка программы на симуляторе и отладочной плате

Программные средства

1. Симулятор – симуляция части или всей микропроцессорной системы на ПК. Пример QEMU, Renode, симулятор встроенный в Keil uVision;
2. Запуск фрагментов программы не связанных с аппаратными средствами контроллера или изолированных от аппаратных средств на ПК.

Аппаратные средства (подключение к МПС через адаптер отладки)

1. Отладка с помощью средств прототипирования:
 - Starter Kit – отладочная плата для оценки некоторых возможностей микропроцессорной системы, часто проблемно ориентирована;
 - Demonstration Board – отладочная плата для оценки основных возможностей микропроцессорной системы;
 - Evaluation Board – отладочная плата с максимальным количеством периферии и возможностью создания прототипа устройства на ее основе;
2. Программирование и отладка на основе собственных, специально разработанных средств.



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.

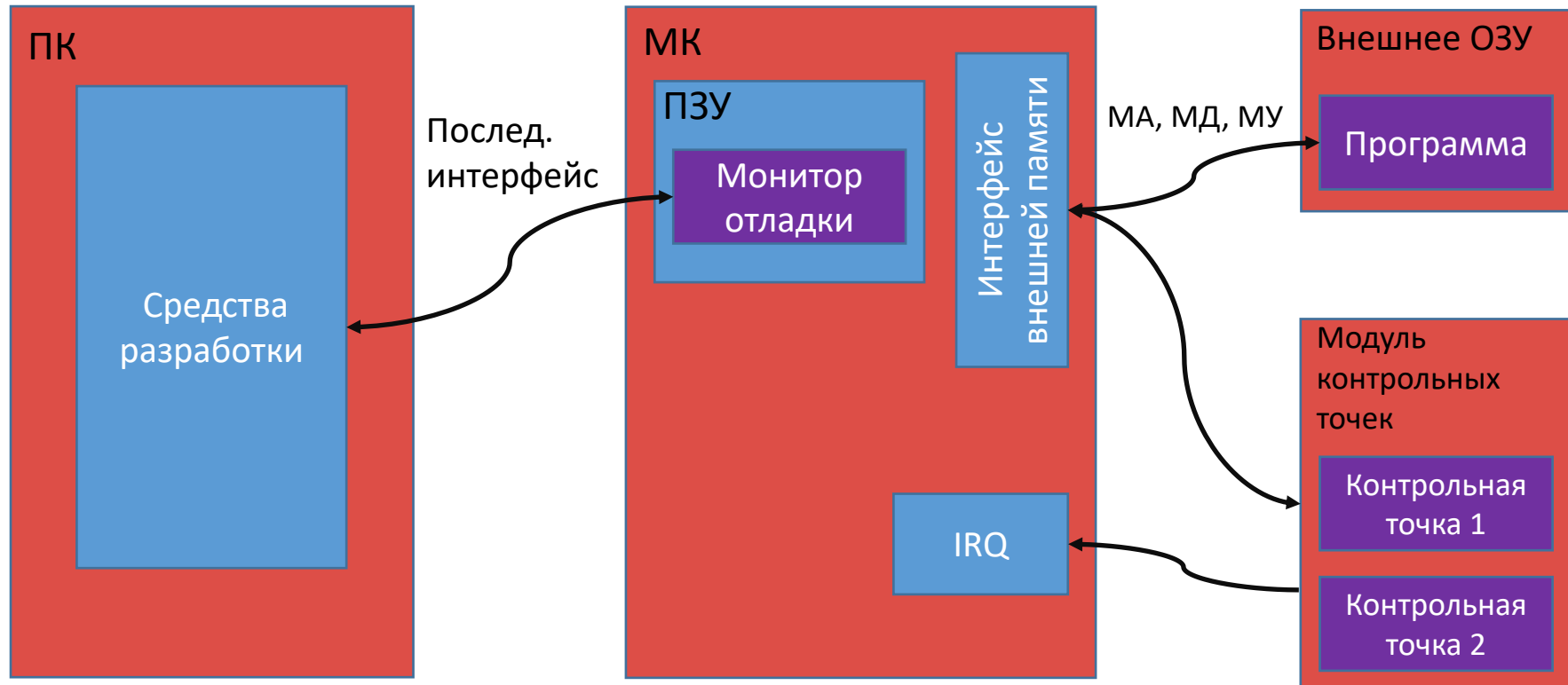
Для симуляторов

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

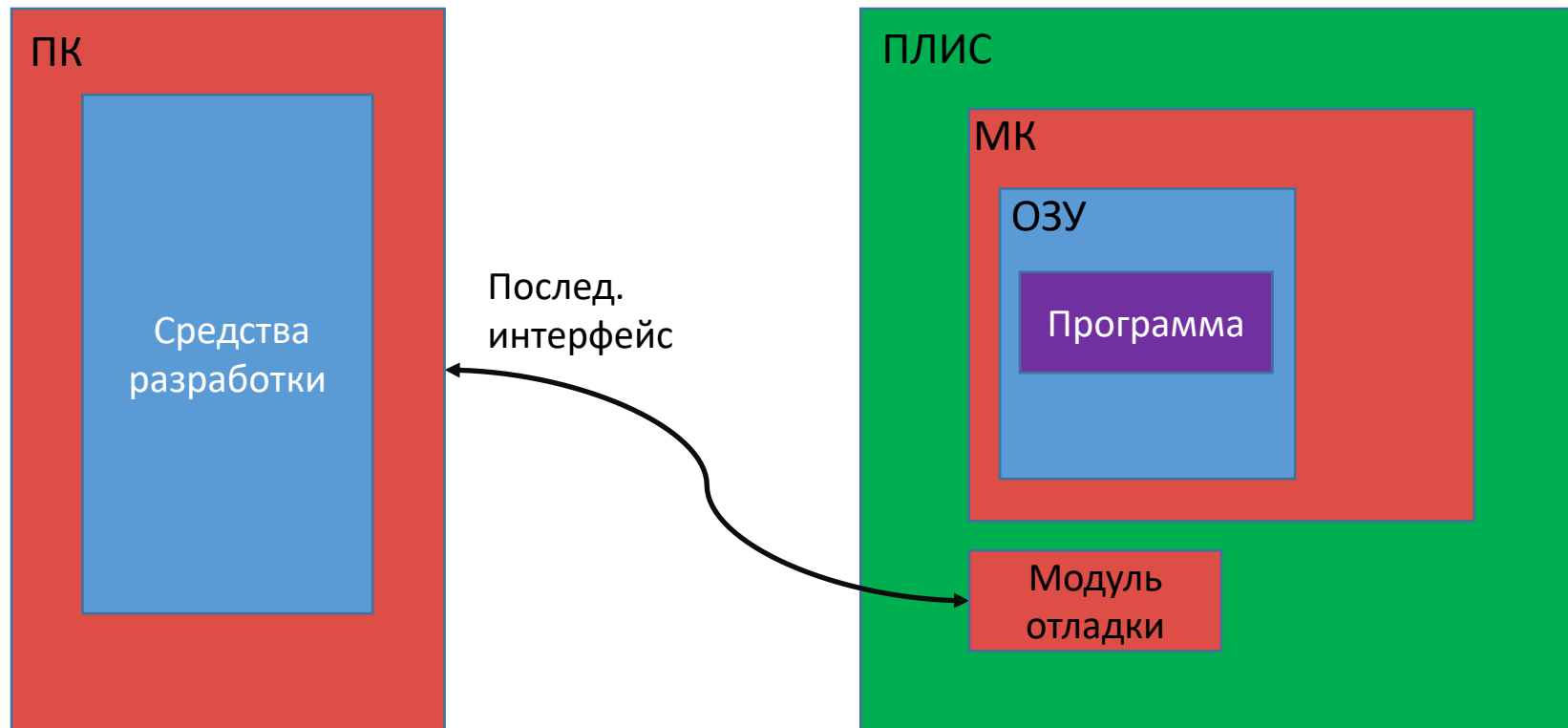
Принципы организации аппаратной отладки

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

Отладка через интерфейс внешней памяти

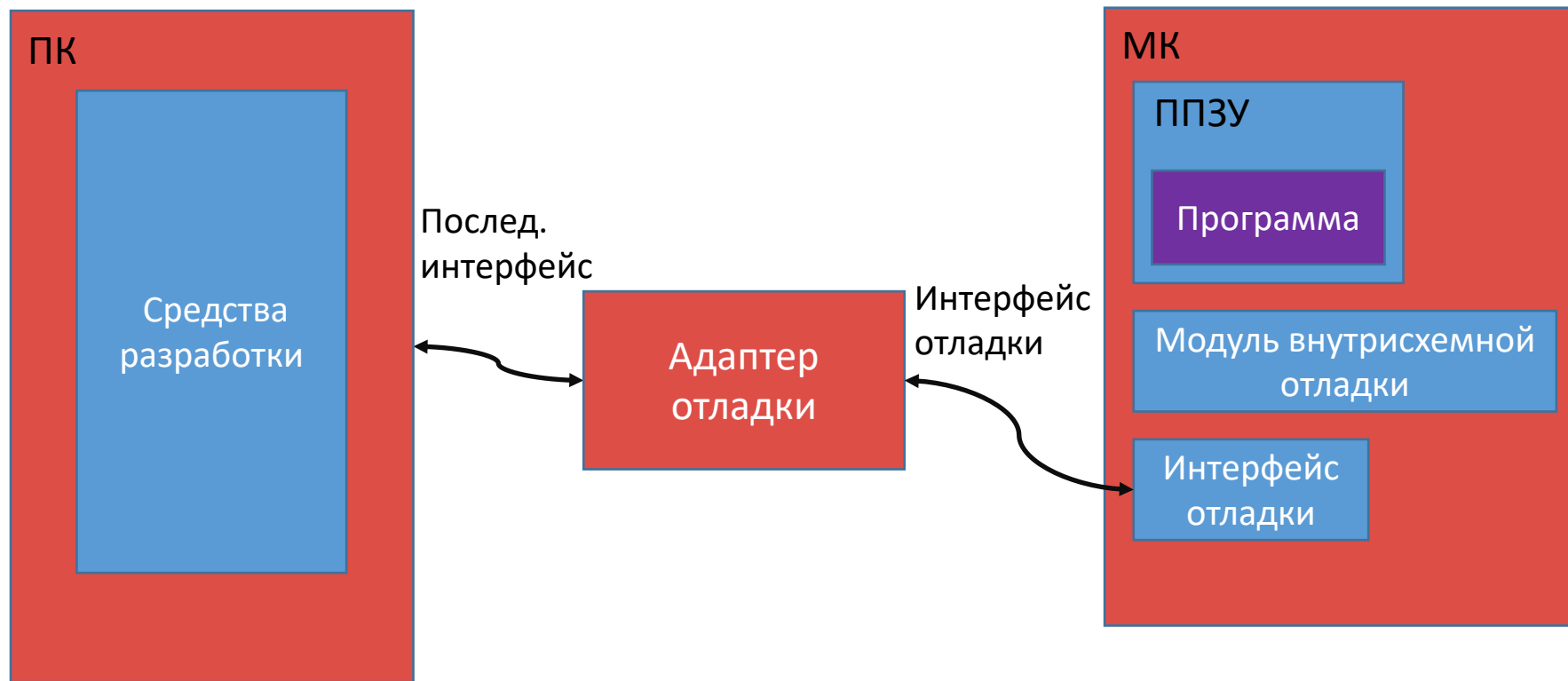


МК имеет интерфейс внешней памяти и способен исполнять программу, размещенную во внешней памяти. В ПЗУ МК располагается программа монитор отладки, которая взаимодействует с персональным компьютером через последовательный интерфейс (USB, UART). Применяется для МК с однократно программируемой или внутренней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



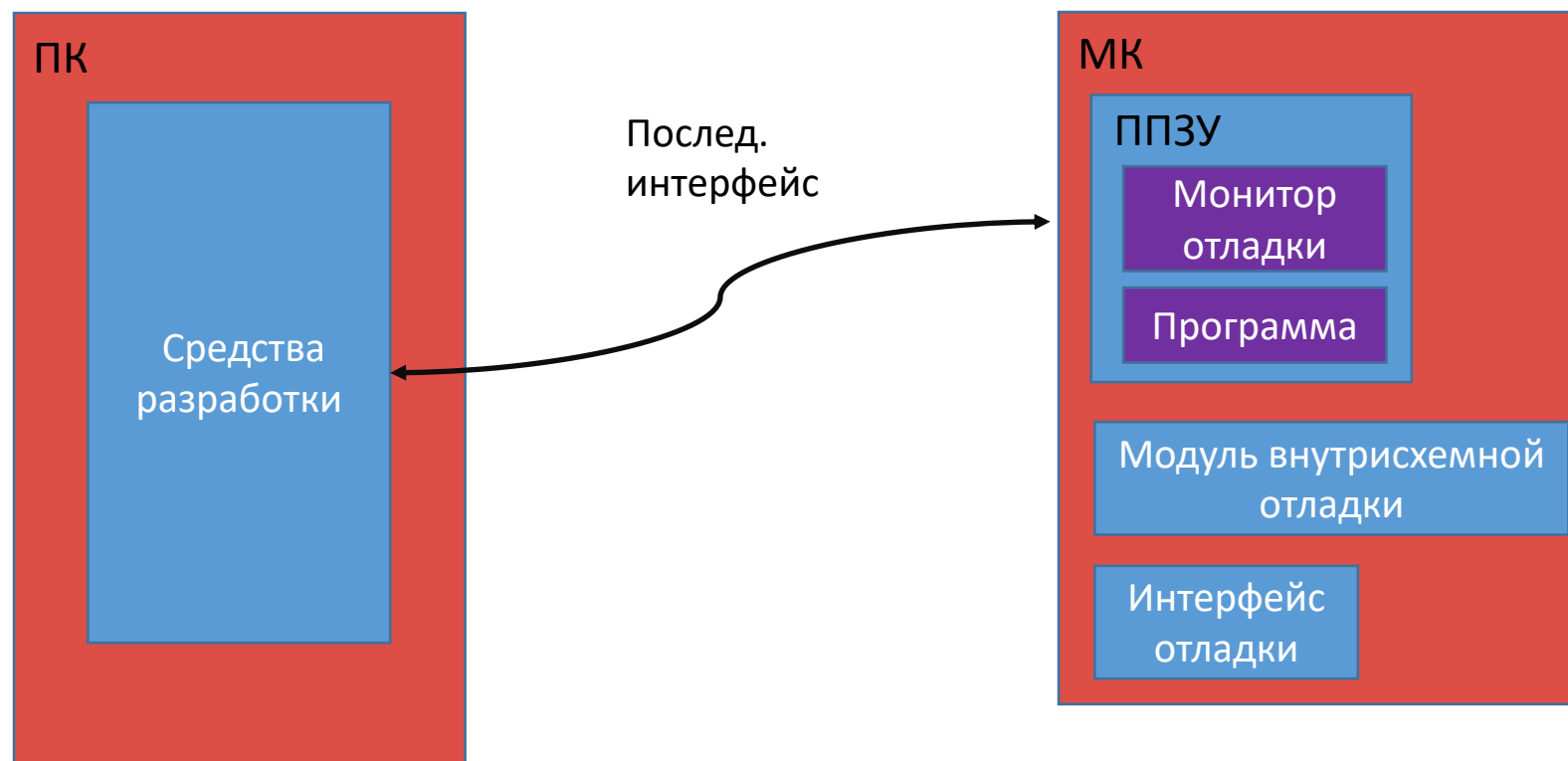
МК (ядро, периферия и память) полностью реализован на ПЛИС (Soft Core). В ПЛИС сконфигурирован модуль отладки, который взаимодействует с персональным компьютером через последовательный интерфейс (USB, UART). Может применяться для отладки массовых микроконтроллеров с однократно программируемой ПЗУ.

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, CAN). Дополнительное устройство отладчик не требуется.

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

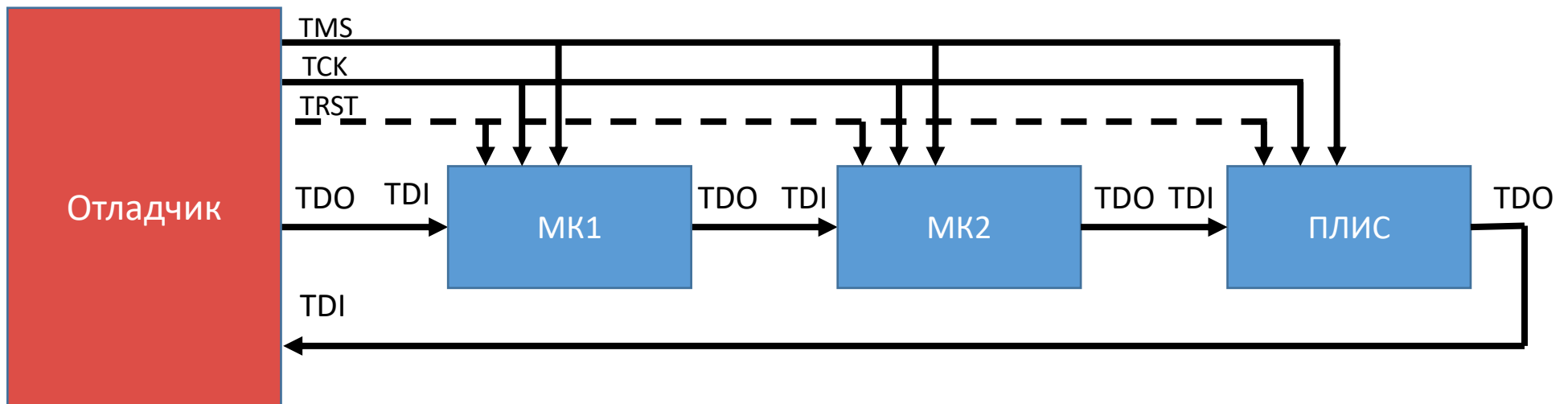
TDI – Test Data Input

TDO – Test Data Output

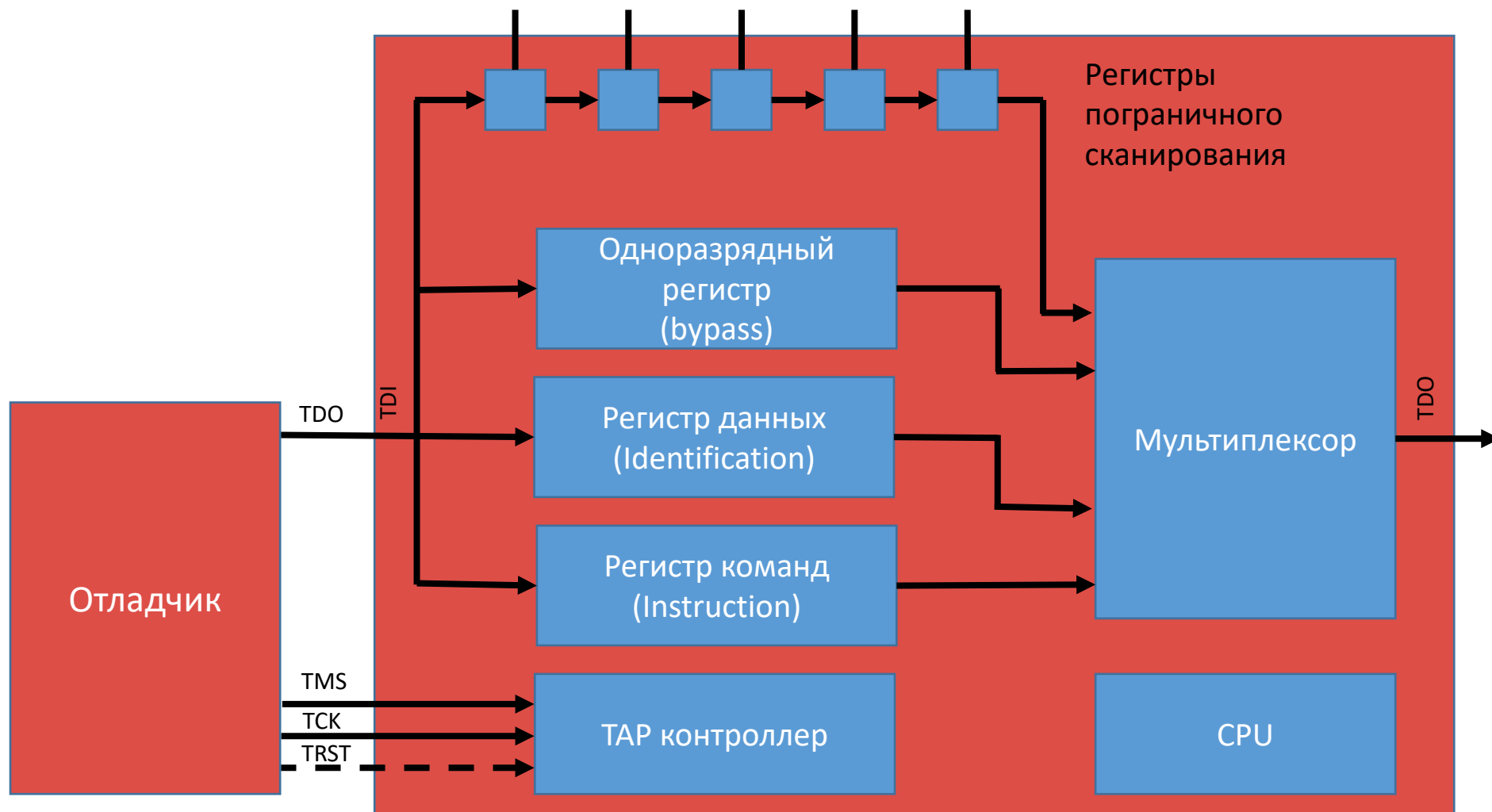
TMS – Test Mode Select

TCK – Test Clock

TRST – Test Reset



Внутрисхемная отладка: Интерфейс JTAG

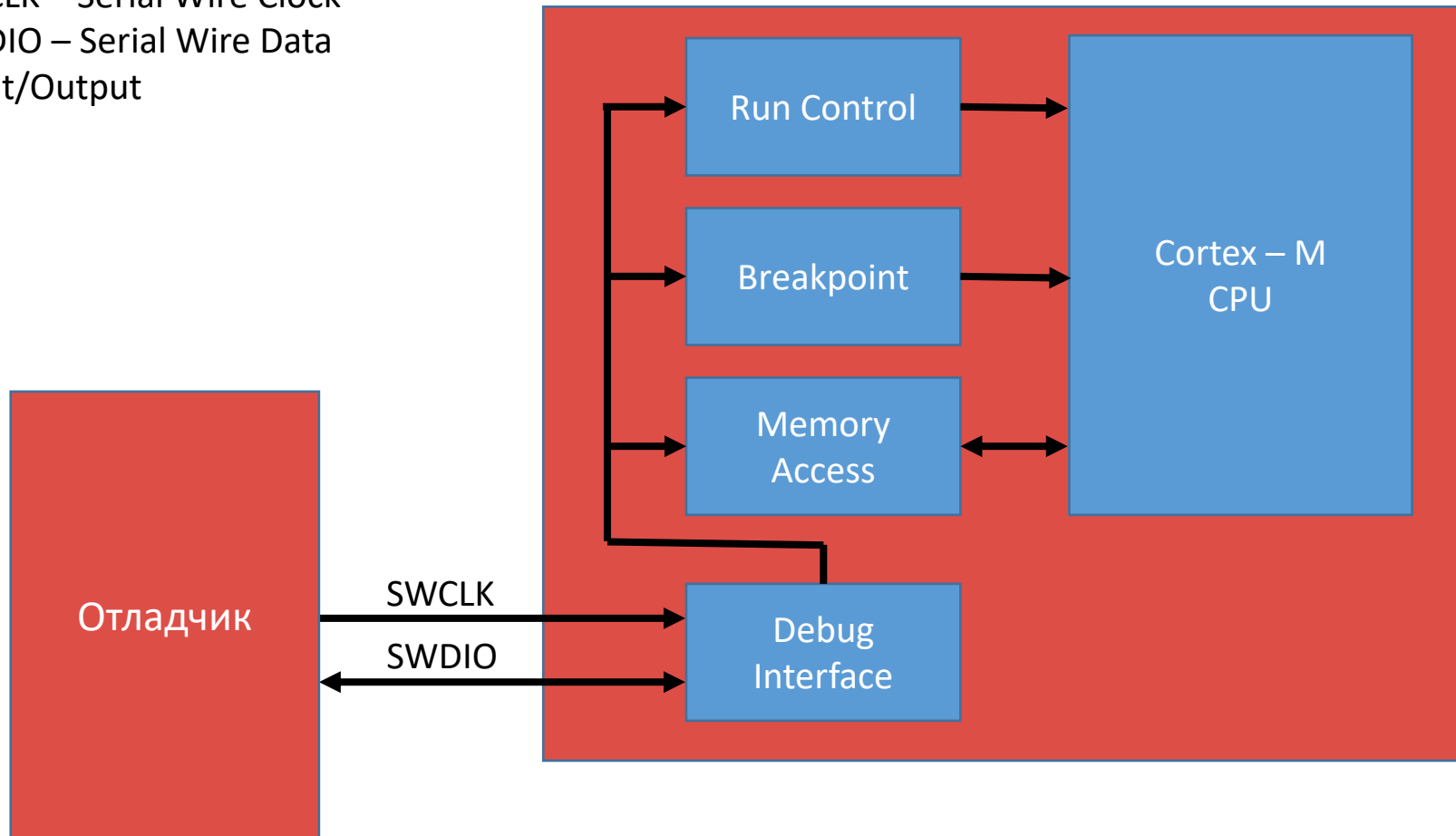


Внутрисхемная отладка: Интерфейс SWD

SWD – Serial Wire Debug ARM Cortex-M

SWCLK – Serial Wire Clock

SWDIO – Serial Wire Data
Input/Output

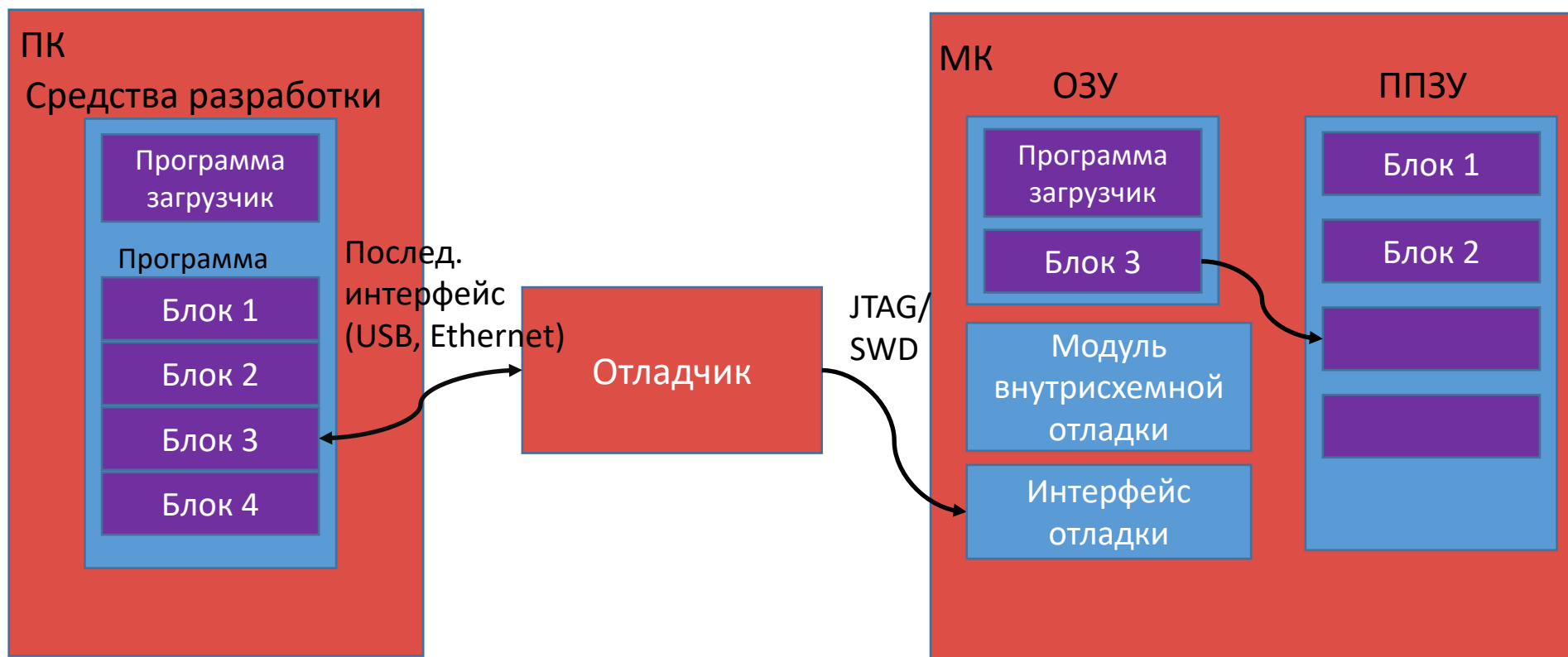


Загрузка программы (прошивка, flashing)

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 загрузкой из внешней памяти);
4. Обновление через беспроводной интерфейс (OTA – Over the Air, FOTA – Firmware Over the Air).

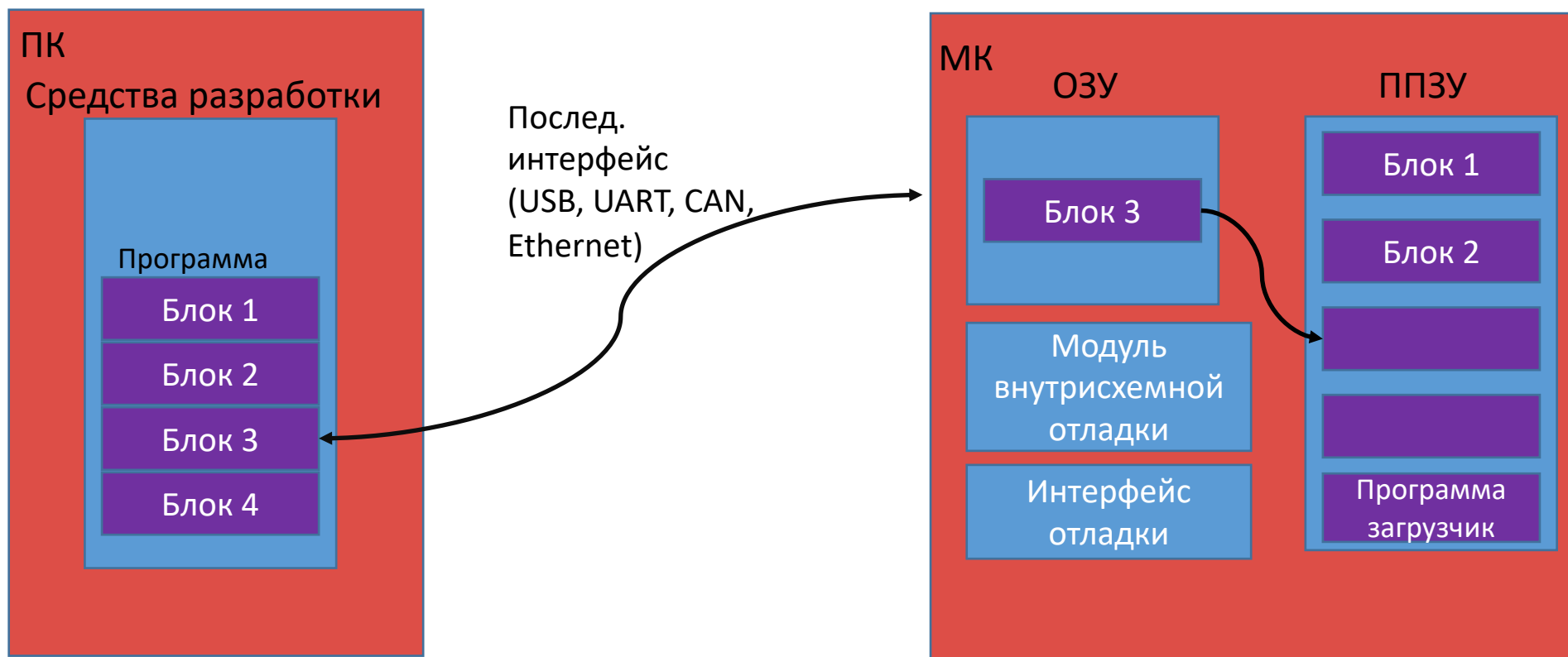
Первые три способа можно использовать как для первичной загрузки, так и для обновления. Четвертый способ – только для обновления.

Загрузка программы в микроконтроллер через внутрисхемный отладчик



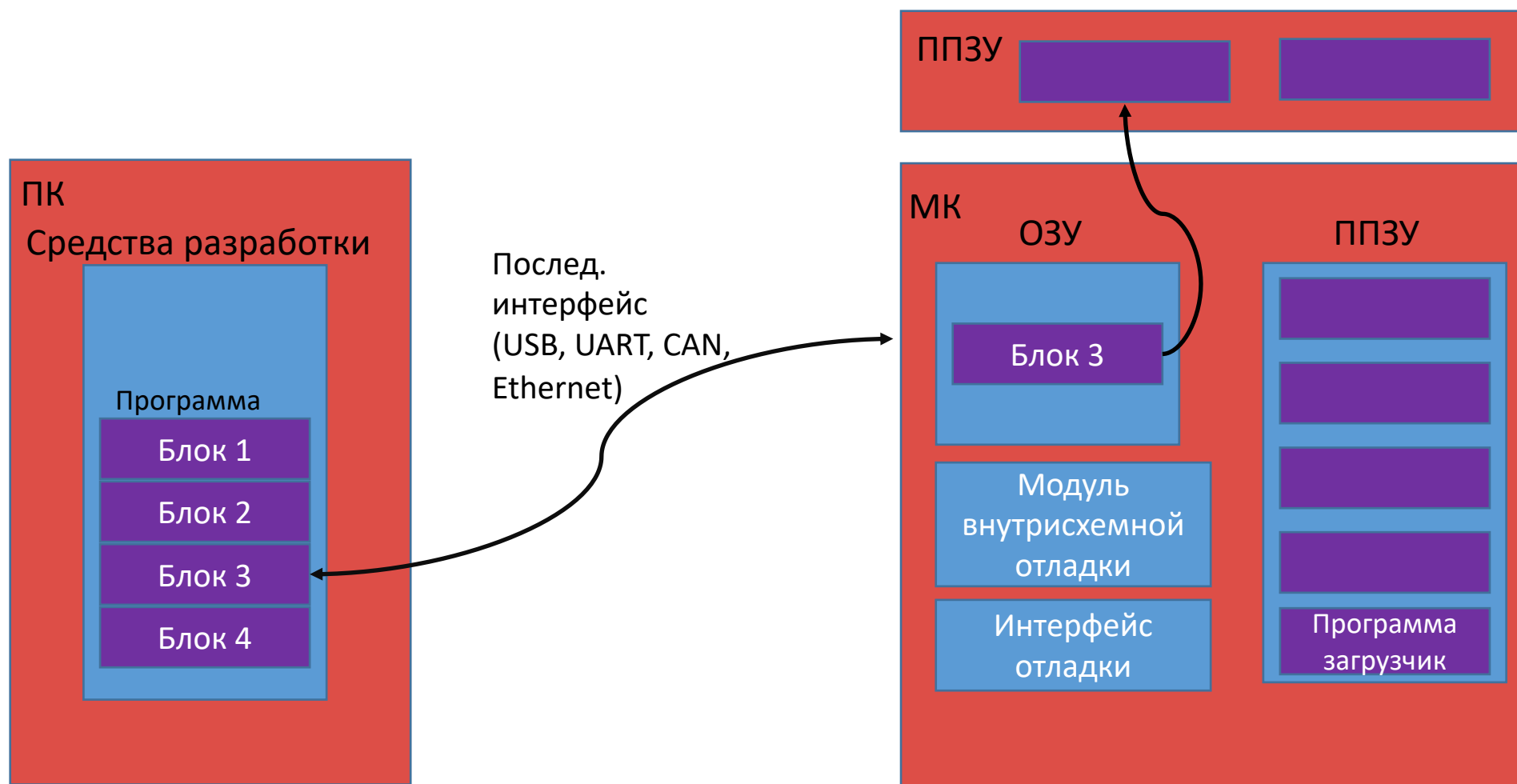
Отладчик загружает в ОЗУ МК специальную программу загрузчик, которая осуществляется программирование ППЗУ МК. Вся программа может не поместиться в ОЗУ МК, поэтому программирование идет блоками.

Загрузка программы в микроконтроллер через загрузчик



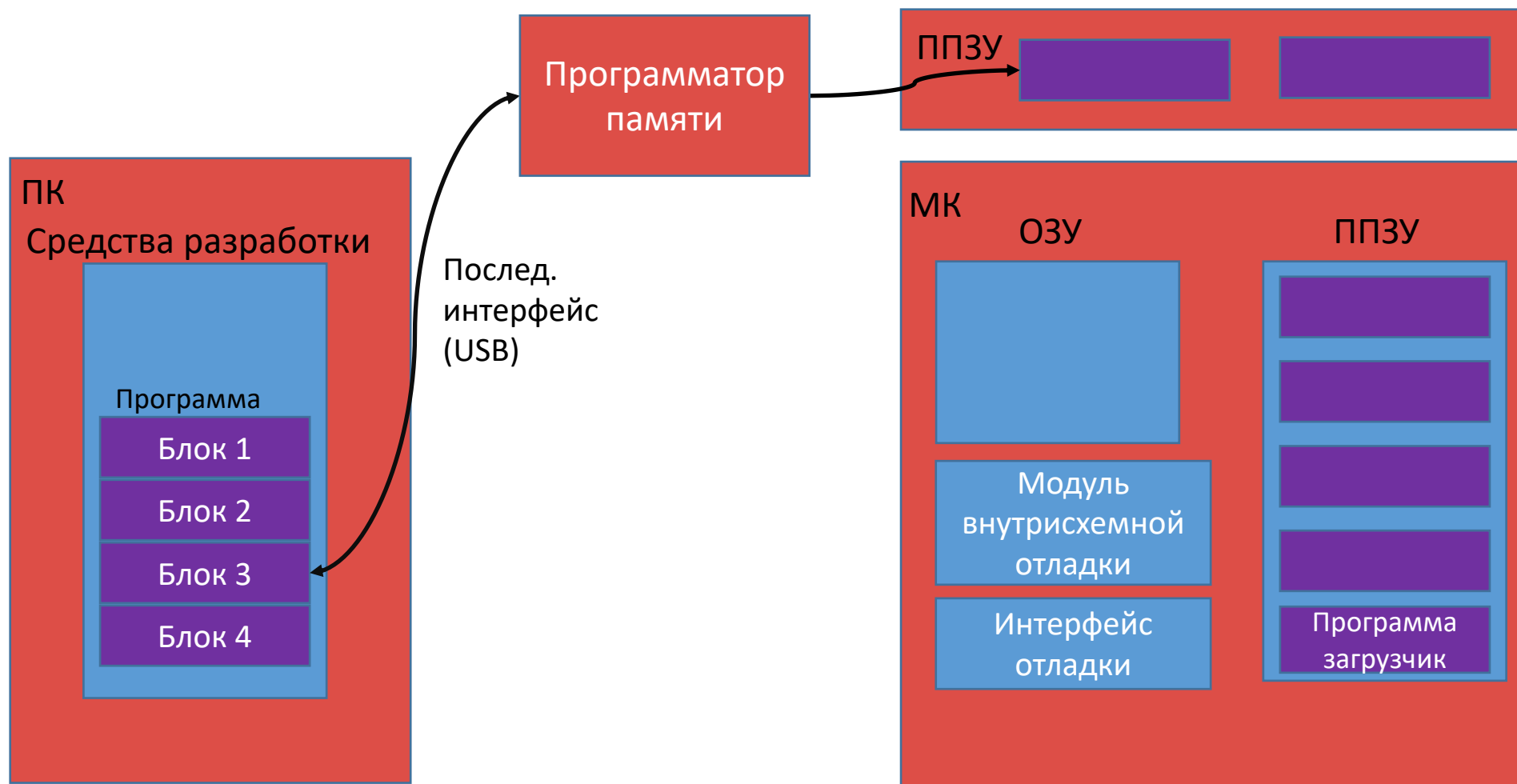
В МК предварительно (производителем или разработчиком) загружена программа загрузчик (bootloader), которая с помощью последовательного интерфейса загружает пользовательскую программу в ОЗУ и программирует ППЗУ.

Загрузка программы во внешнюю память



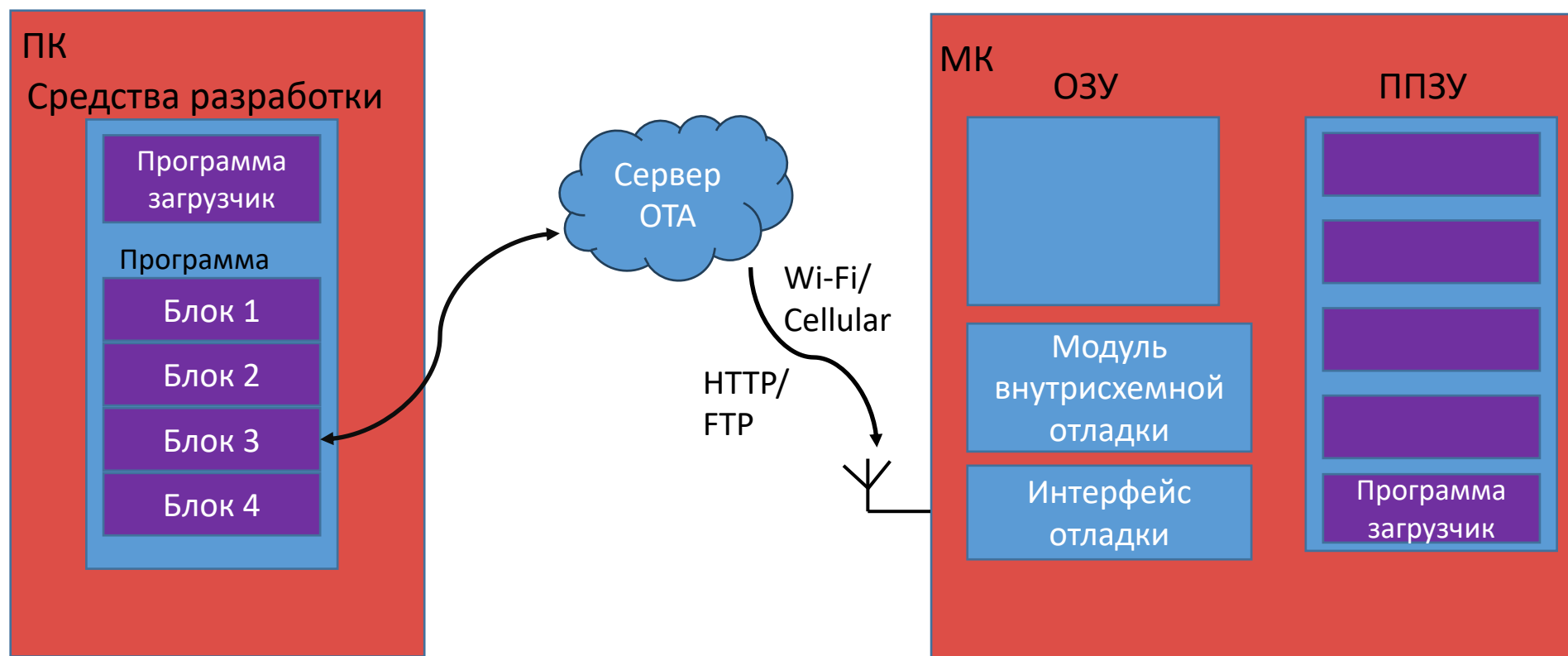
Если к МК или процессору подключена внешняя память по параллельному интерфейсу или SPI/QSPI и предполагается исполнение программы из нее, то загрузку в нее осуществляет программа загрузчик.

Загрузка программы во внешнюю память



Если к МК или процессору подключена внешняя память по параллельному интерфейсу или по SPI/QSPI/MMC, то загрузку программы можно провести до монтажа микросхемы памяти в плату с помощью специальных программаторов памяти.

Обновление программы OTA



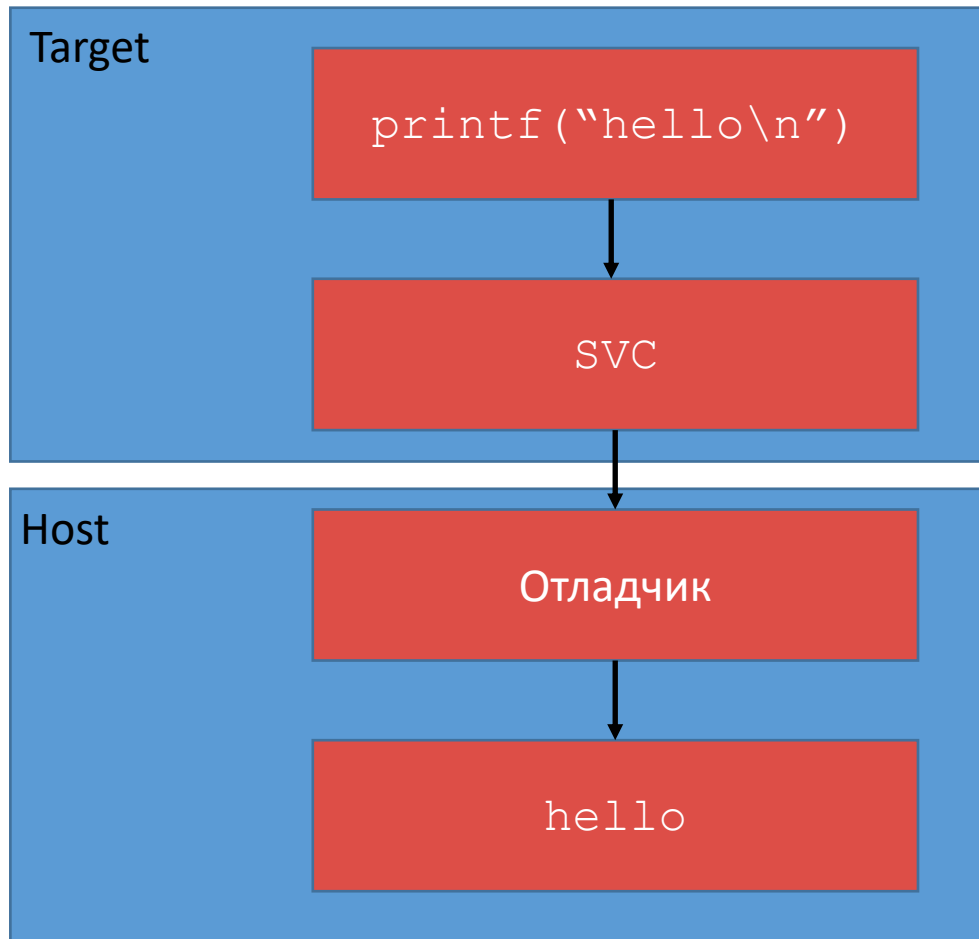
Разработчик готовит программу и загружает на сервер OTA (Over the Air). Устройство периодически проверяет сервер или сам сервер рассылает уведомление о новой программе. Устройство скачивает новую программу, проверяет целостность и обновляется.

Полухостинг (semi hosting)

Semi hosting — это прием, который позволяет коду, работающему в встраиваемой системе (target), взаимодействовать и использовать средства ввода-вывода на главном компьютере (host), где запущен отладчик. Это может быть полезно во время разработки, когда встраиваемая система может не иметь доступных средств ввода-вывода.

1. Программа во встраиваемой системе оснащена вызовами специального набора операций полухостинга (предоставляются интерфейсом отладки);
2. Когда программа во встраиваемой системе достигает вызова полухостинга, он выполняет инструкцию-ловушку, которая останавливает нормальное выполнение и сигнализирует об этом отладчику;
3. Отладчик, распознает ловушку как вызов полухостинга. Он считывает параметры операции полухостинга из памяти цели или регистров ЦП.
4. Выполнение хоста: затем отладчик выполняет запрошенную операцию на хост-компьютере. Это может быть чтение файла, запись в консоль и т.д.
5. После завершения операции отладчик возвращает управление программе во встраиваемой системе, которое продолжает выполнение, как если бы оно само выполнило операцию ввода-вывода.

Полухостинг (semi hosting)



Программа в МК вызывает функцию ввода/вывода

Стандартная библиотека C обрабатывает этот вызов, и вместо печати в доступные потоки вызывает отладчик

Отладчик считывает аргументы функции через чтение регистров и памяти

В терминале хоста отображается строка

Полухостинг можно применять для любого ввода/вывода, например можно открыть файл на хосте и прочитать его в память МК или записать файл из памяти МК в файловую систему хоста.

Трассировка

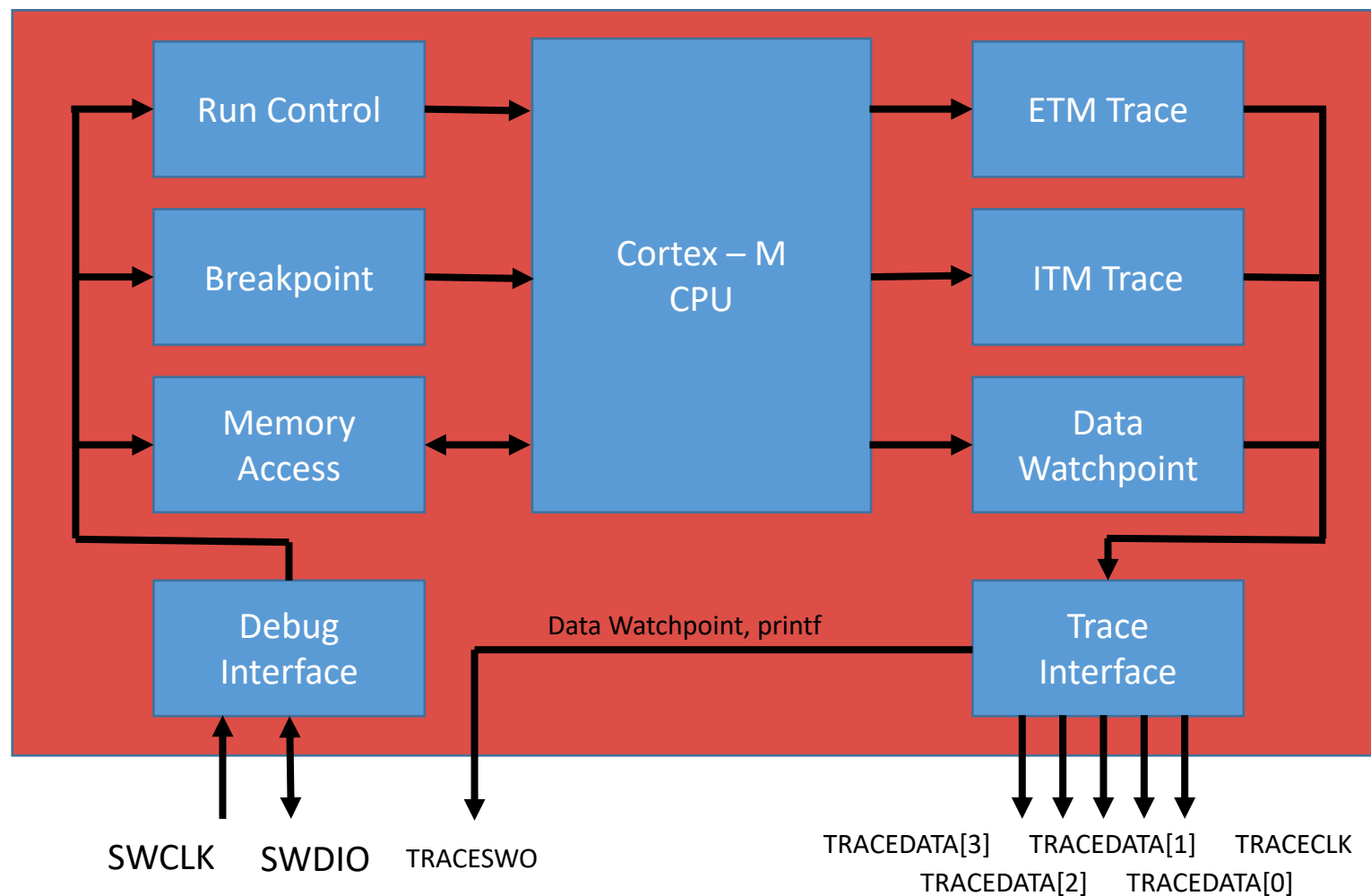
Трассировка – дает дополнительные возможности при отладке:

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.

Аппаратная трассировка: SWD, TRACEPORT

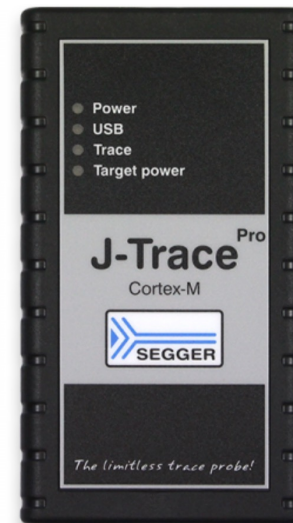


В МК реализован модуль и интерфейс трассировки. Для подключения к ПК требуется специальное устройство отладчик/трассировщик. Можно проводить трассировку в реальном времени не влияя на ход выполнения программы.

Пример отладчика и адаптера трассировки

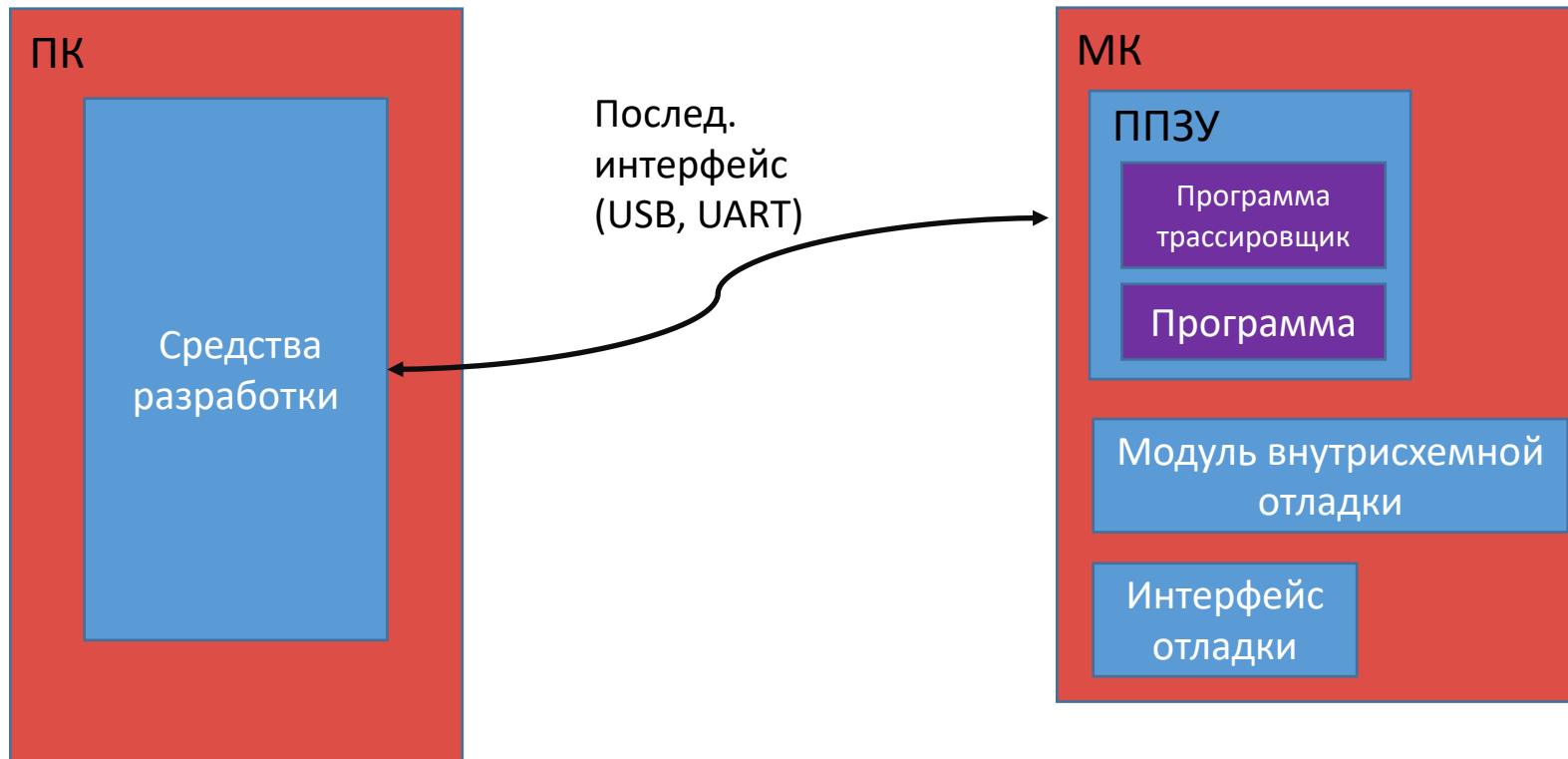


Отладчик
(debug adapter,
debug probe)



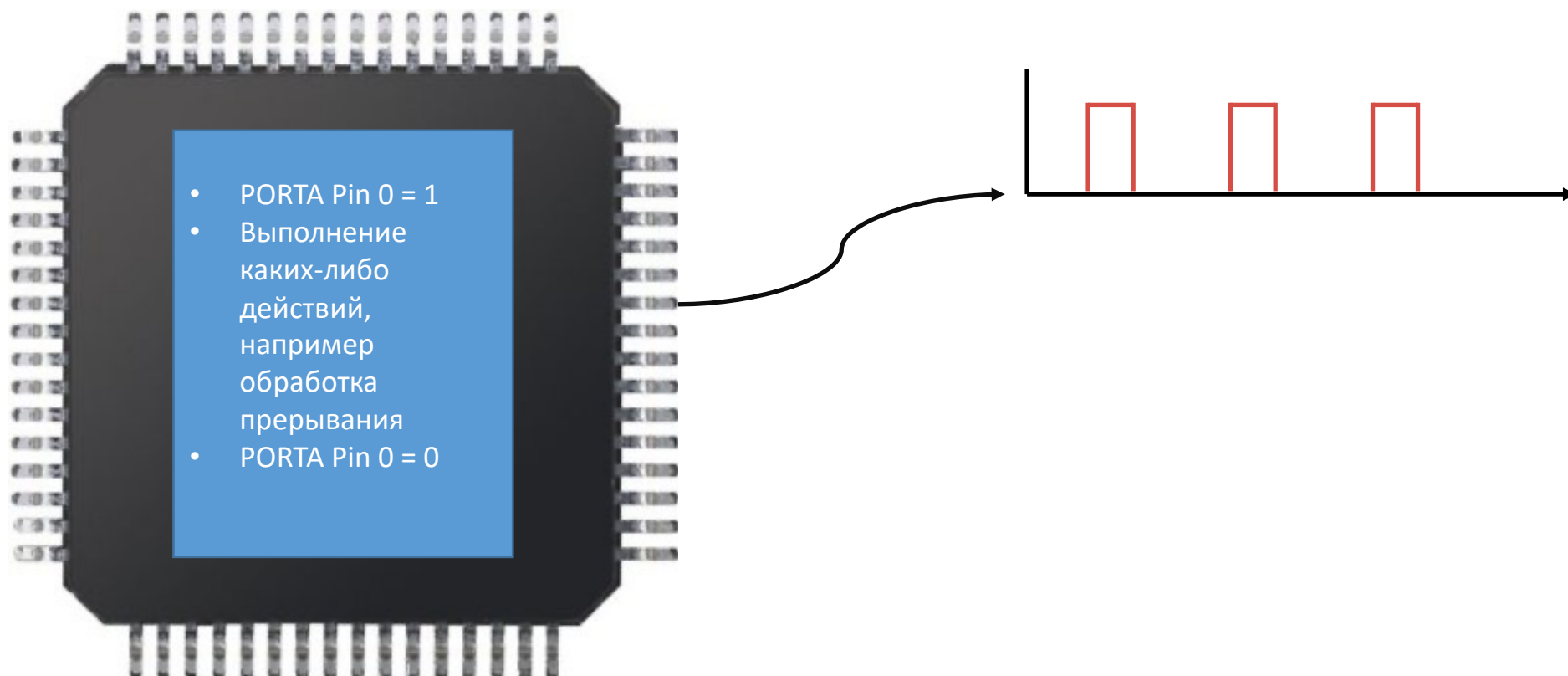
Адаптер трассировки
(trace adapter,
trace probe)

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.

1. Отладка/трассировка:

- В реальном масштабе времени: светодиоды или 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, I2C, 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.

Дополнительные слайды

Команды системы 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
```

Отправить 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 репозиториях:

```
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 удаленном репозитории origin:

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

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

```
git pull origin
```

Команды системы 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 ручного 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() "
```