Российский Альянс RISC-V
Конкурс учебно-методических и исследовательских работ

Исследовательская работа с учебно-методическим практикумом

Bare-metal разработка для микроконтроллера К1921ВГ1Т, включая оценку применимости языка Rust (no_std) и подготовку открытой документации

Автор: Черных В.А., ННГУ им. Н.И. Лобачевского, группа 3824Б1ПР2
Материалы: https://github.com/chernykhval/k1921vg1t-riscv-baremetal

© 2026 Черных В.А. Материалы распространяются на условиях лицензии Creative Commons «С указанием авторства» версии 4.0 Международная (CC BY 4.0):
https://creativecommons.org/licenses/by/4.0/legalcode.ru

2026

Содержание

Перечень сокращений и терминов 4
Аннотация 6
1. Актуальность и новизна 7
    1.1. Актуальность 7
    1.2. Состояние вопроса 7
    1.3. Новизна 8
2. Аппаратная и программная среда 9
    2.1. Микроконтроллер 9
    2.2. Отладочная плата и адаптер 9
    2.3. Программные средства 9
    2.4. Организация репозитория и проверка сборки 10
3. Rust-PAC из исправленного SVD 12
    3.1. Что такое PAC и зачем он нужен 12
    3.2. Исходные данные 12
    3.3. Исправление SVD 12
    3.4. Генерация и воспроизводимость 13
    3.5. Состав PAC 14
    3.6. Ограничения 14
4. Работа с периферией 16
    4.1. Включение блока 16
    4.2. Порты ввода-вывода 17
    4.3. Прерывание от таймера 17
    4.4. UART и частота внутреннего генератора 17
    4.5. PLL 19
    4.6. АЦП и датчик температуры 19
    4.7. ЦАП 20
    4.8. Блок шифрования 20
5. Оценка применимости Rust (no_std) 21
    5.1. Инструменты и поддержка микроконтроллера 21
    5.2. Размер прошивки 21
    5.3. Где нужен unsafe 22
    5.4. Что проверяет компилятор 24
    5.5. Время сборки 24
    5.6. Вывод 25
6. Найденные дефекты документации и SDK 26
    6.1. Дефекты материалов производителя 26
    6.2. Собственные ошибки 28
    6.3. Исправления производителя в более поздних версиях 29

CC BY 4.0

7. Учебно-методический практикум 30
7.1. Подготовка рабочего места 30
7.2. Работа 1. Программа без библиотек 30
7.3. Работа 2. Среда выполнения riscv-rt 31
7.4. Работа 3. Регистры по именам: PAC 31
7.5. Работа 4. Прерывание от таймера 32
7.6. Работа 5. UART и измерение частоты 32
7.7. Работа 6. АЦП и датчик температуры 33
7.8. Работа 7. ЦАП 33
7.9. Работа 8. Аппаратное шифрование 33
7.10. Работа 9. Переключение на PLL 34
7.11. Итоговое задание 34
8. Заключение 35
8.1. Основные результаты 35
8.2. Значимость 35
8.3. Ограничения 35
8.4. Возможная дальнейшая работа 36
8.5. Лицензия 36
Список источников 37
Приложение А. Места исправлений в SVD-описании 38
Приложение Б. Подключение JTAG-адаптера 39

CC BY 4.0

Перечень сокращений и терминов

Сокращение, термин	Пояснение
8N1	Формат кадра UART: 8 бит данных, без бита чётности, 1 стоповый бит.
AES	Advanced Encryption Standard — американский стандарт блочного шифрования (FIPS PUB 197).
bare-metal	Программирование «на голом железе»: программа работает на микроконтроллере без операционной системы.
cargo	Система сборки и менеджер пакетов языка Rust.
CMSIS-SVD	Стандартный формат SVD-описаний, предложенный компанией ARM.
CRYPTO	Блок аппаратного шифрования микроконтроллера: AES-128, AES-256, «Кузнечик», «Магма».
CSR	Control and Status Registers — служебные регистры ядра RISC-V (управление прерываниями, сведения о ядре и т. п.).
ECB	Electronic Codebook — режим простой замены: каждый блок данных шифруется независимо от других.
ELF	Формат исполняемого файла, который получается после сборки и прошивается в микроконтроллер.
embedded-hal	Набор общих интерфейсов Rust для периферии микроконтроллеров; на нём основаны драйверы внешних устройств, не зависящие от конкретного микроконтроллера.
FIFO	Буфер «первым пришёл — первым ушёл»; в работе — буфер передатчика UART на 32 байта.
Flash-память	Энергонезависимая память микроконтроллера, в которой хранится программа.
GCM	Galois/Counter Mode — режим шифрования со счётчиком и проверкой целостности данных.
GitHub Actions	Сервис GitHub, который автоматически собирает проект при каждом изменении репозитория (непрерывная интеграция, CI).
GPIO	General-Purpose Input/Output — порты ввода-вывода общего назначения.
HAL	Hardware Abstraction Layer — библиотека более высокого уровня, чем PAC, с удобными интерфейсами для работы с периферией.
hart	Аппаратный поток выполнения в терминах RISC-V; в К1921ВГ1Т каждое ядро — один hart.
HSE	Генератор с внешним кварцевым резонатором; на отладочной плате — кварц 27 МГц.
HSI	Внутренний RC-генератор микроконтроллера; после сброса ядро работает от него.
JTAG	Интерфейс отладки и прошивки микроконтроллера. Сигналы: TCK (тактирование), TMS (выбор режима), TDI и TDO (данные в микросхему и из неё). На разъём JTAG платы выведен также сигнал nRST — сброс микроконтроллера.
misa	Регистр CSR ядра RISC-V, по битам которого видно, какие расширения набора команд поддерживает ядро.
no_std	Режим Rust без стандартной библиотеки: доступна только базовая библиотека `core`, не требующая операционной системы.
NVIC	Nested Vectored Interrupt Controller — контроллер прерываний ядер ARM Cortex-M; в К1921ВГ1Т его нет.
OpenOCD	Открытая программа для отладки и прошивки микроконтроллеров через JTAG; в работе используется её версия от НИИЭТ.
PAC	Peripheral Access Crate — крейт доступа к периферии: регистры и поля микроконтроллера в виде типов Rust.
PLIC	Platform-Level Interrupt Controller — контроллер прерываний уровня платформы RISC-V: собирает запросы прерываний от периферии и передаёт их ядру.
PLL	Phase-Locked Loop — синтезатор частоты (фазовая автоподстройка частоты): из частоты кварца получает более высокую частоту для ядра и периферии.
RCU	Блок управления тактовыми сигналами и сбросом периферийных блоков.
RISC-V	Открытая архитектура набора команд процессора.
riscv-rt	Крейт среды выполнения для RISC-V: стартовый код, скрипт компоновки, обработчики исключений и прерываний.
rustup	Программа установки и обновления компилятора Rust.

Сокращение, термин	Пояснение
RV32IMAC	32-битный RISC-V с расширениями I (базовые целочисленные команды), M (умножение и деление), A (атомарные операции), C (сжатые 16-битные команды). Ядро К1921ВГ1Т поддерживает ещё F и D (плавающая точка одинарной и двойной точности) и P (упакованные SIMD-команды).
SDK	Software Development Kit — набор средств разработки производителя: библиотеки, примеры, конфигурации.
SIU	Системный блок микроконтроллера; в работе используется для запуска таймеров и управления датчиком температуры.
SMA	Резьбовой коаксиальный разъём; на плате к нему выведены выходы ЦАП.
SVD	System View Description — XML-файл с описанием регистров микроконтроллера: адреса блоков, регистры, поля, доступ.
svd2rust	Инструмент, генерирующий PAC из SVD-описания.
Tier 2	Второй уровень поддержки платформы компилятором Rust: сборка гарантируется для каждого выпуска, готовая библиотека `core` ставится через rustup.
TMR0	Таймер 0 микроконтроллера.
UART	Последовательный приёмопередатчик; в работе через него программы выводят текст на компьютер.
unsafe	Блок кода Rust, в котором часть проверок компилятора отключена и за корректность отвечает программист.
WinUSB, Zadig	WinUSB — стандартный USB-драйвер Windows, через который OpenOCD работает с адаптером; Zadig — программа для его установки.
АЦП	Аналого-цифровой преобразователь (ADC): переводит напряжение в число.
Идентификатор JTAG	Число, которое микросхема сообщает отладчику при подключении; по нему OpenOCD проверяет, что подключена нужная микросхема.
Крейт	Пакет (библиотека или программа) в экосистеме Rust.
Критическая секция	Участок кода, который выполняется без прерываний, чтобы к общим данным не обратились одновременно из двух мест.
«Кузнечик», «Магма»	Блочные шифры из ГОСТ Р 34.12-2015.
ОЗУ	Оперативная память.
Паника	Аварийная остановка программы на Rust при обнаруженной ошибке, например при выходе за границы массива; в no_std обрабатывается функцией, которую задаёт программа.
РП	Руководство пользователя микроконтроллера К1921ВГ1Т от АО «НИИЭТ».
Секвенсор АЦП	Управляющий блок АЦП, который запускает преобразования заданных каналов и складывает результаты в свой буфер.
Секции прошивки	Части прошивки: `.text` — код программы, `.rodata` — константы, `.data` — переменные с начальным значением, `.bss` — переменные, обнуляемые при запуске.
Скрипт компоновки	Файл для компоновщика, описывающий, по каким адресам памяти размещать код и данные.
Стартовый код	Код, который выполняется сразу после сброса до функции `main`: настраивает стек, подготавливает память.
Такты ожидания Flash-памяти	Дополнительные такты, которые ядро пропускает при чтении из Flash-памяти, чтобы она успела выдать данные на высокой частоте.
Функция крейта (feature)	Необязательная часть крейта, которая включается в файле `Cargo.toml` проекта, использующего крейт.
ЦАП	Цифро-аналоговый преобразователь (DAC): переводит число в напряжение.

CC BY 4.0

Аннотация

Работа посвящена bare-metal разработке для микроконтроллера К1921ВГ1Т (АО «НИИЭТ», архитектура RISC-V) и оценке того, насколько для неё применим язык Rust без стандартной библиотеки (no_std). Средства разработки производителя рассчитаны на C; для Rust поддержки этого микроконтроллера в открытом доступе не было.

Цель работы — выяснить, можно ли разрабатывать программы для К1921ВГ1Т на Rust, создать для этого необходимые средства и подготовить открытую документацию, по которой работу можно повторить.

Задачи:

1. получить из SVD-описания производителя крейт доступа к периферии (PAC) для Rust, исправив описание там, где оно этому мешает;
2. написать и проверить на плате примеры работы с основными блоками микроконтроллера;
3. количественно оценить применимость Rust в сравнении с C;
4. описать найденные дефекты документации и SDK;
5. подготовить репозиторий с инструкцией и практикум.

Результаты. В SVD-описании производителя обнаружены восемь классов дефектов, мешающих получить PAC (всего 48 мест исправлений); после их исправления PAC генерируется воспроизводимо. Девять примеров на Rust — от программы без единой библиотеки до прерываний, аналоговых блоков, аппаратного шифрования по ГОСТ Р 34.12-2015 и переключения на PLL 150 МГц — работают на плате. Размер прошивок на Rust того же порядка, что на C, а большая часть блоков `unsafe` вызвана неполнотой SVD-описания, а не языком: Rust (no_std) применим для bare-metal разработки под К1921ВГ1Т. Обнаружены также семь расхождений SVD-описания с руководством пользователя и заголовком SDK и дефекты самого руководства и SDK. Все найденные дефекты описаны; перечень не претендует на полноту. Подготовлен практикум из девяти работ с заданиями.

Все материалы опубликованы в репозитории https://github.com/chernykhval/k1921vg1t-riscv-baremetal и распространяются на условиях лицензии CC BY 4.0 [1].

Ключевые слова: К1921ВГ1Т, RISC-V, Rust, no_std, bare-metal, PAC, SVD, svd2rust.

1. Актуальность и новизна

1.1. Актуальность

К1921ВГ1Т — микроконтроллер АО «НИИЭТ» на архитектуре RISC-V. Производитель поставляет для него средства разработки на C: SDK `k1921vg1t_sdk` [2] и новый SDK `riscv_cmake_sdk` на основе CMake [3]. Библиотеки и примеры обоих SDK написаны на C.

Язык Rust рассматривают как альтернативу C во встраиваемых системах: программы на нём не требуют операционной системы и сборщика мусора, а компилятор проверяет работу с памятью и многие другие ошибки до запуска программы. Компилятор Rust официально поддерживает микроконтроллеры RISC-V без операционной системы: цель `riscv32imac-unknown-none-elf` входит во второй уровень поддержки (Tier 2) — её сборка гарантируется для каждого выпуска компилятора, а готовая библиотека `core` для неё устанавливается через rustup [4]. Однако чтобы писать на Rust программы для конкретного микроконтроллера, нужен крейт доступа к его периферии (PAC), а его генерация зависит от качества SVD-описания, которое поставляет производитель (раздел 3).

Для разработчика, который хочет применить Rust на К1921ВГ1Т, в открытом доступе нет ни PAC, ни примеров, ни оценки того, насколько это практично (см. следующий подраздел). Восполнить этот пробел — задача работы.

SVD-описание используют не только генераторы кода на Rust: по нему отладчики показывают регистры периферии по именам [5]. Поэтому найденные и исправленные дефекты описания (раздел 6) полезны и тем, кто пишет на C.

1.2. Состояние вопроса

Поиск существующих решений выполнен 10 октября 2026 г.

- **Реестр крейтов Rust crates.io** [6]. По запросам «k1921vg1t», «vg1t», «k1921» и «niiet» крейтов для К1921ВГ1Т нет. Для микроконтроллеров НИИЭТ есть только два PAC, оба для микроконтроллеров серии К1921ВК на ядре ARM; для RISC-V микроконтроллеров НИИЭТ крейтов нет.
- **GitHub.** Репозиториев на Rust, относящихся к К1921ВГ1Т или к другим RISC-V микроконтроллерам НИИЭТ, не найдено.
- **SDK производителя.** Ни в `k1921vg1t_sdk` (проверены коммиты `9fa40f85be5b` и `44510f7ad5e8`), ни в `riscv_cmake_sdk` (коммит `ae5a8e69dfe0`) нет кода на Rust. SVD-описание в обоих SDK генератор svd2rust не принимает без исправлений (подраздел 6.3).

Таким образом, на дату проверки открытых средств разработки на Rust для К1921ВГ1Т не обнаружено. Вывод относится только к К1921ВГ1Т и к перечисленным источникам.

1.3. Новизна

В работе впервые для К1921ВГ1Т (по результатам поиска выше) в открытом доступе:

1. создан PAC на Rust, который воспроизводимо генерируется из исправленного SVD-описания (раздел 3);
2. выявлены и описаны дефекты SVD-описания, документации и SDK производителя с проверкой по руководству пользователя и, где возможно, на плате (раздел 6);
3. написаны девять примеров на Rust (no_std) — от мигания светодиодом без внешних библиотек до прерываний, аналоговых блоков, аппаратного шифрования «Кузнечиком», проверенного по тестовому вектору ГОСТ Р 34.12-2015, и переключения на PLL; все проверены на плате (раздел 4);
4. количественно оценена применимость Rust: размер прошивок по сравнению с C, использование `unsafe`, ошибки, которые находит компилятор, время сборки (раздел 5).

Кроме того, подготовлена открытая документация: репозиторий с инструкцией от подключения платы к JTAG-адаптеру FT2232HL Board до мигания светодиода, с описанием всех используемых инструментов и их версий, и практикум по примерам (раздел 7).

Все материалы опубликованы на условиях лицензии CC BY 4.0 [1].

2. Аппаратная и программная среда

2.1. Микроконтроллер

К1921ВГ1Т — микроконтроллер АО «НИИЭТ» архитектуры RISC-V. По руководству пользователя [7] (далее — РП) он содержит два ядра SCR5 с набором команд RV32IMAFDCP и частотой до 204 МГц, 4 МБ основной и 512 КБ дополнительной Flash-памяти, 1 МБ ОЗУ, внутренний RC-генератор HSI (о его частоте — в подразделе 6.1.3), генератор HSE для внешнего кварца от 2 до 27 МГц и системный PLL. Из периферии в работе использованы порты ввода-вывода GPIO, таймер TMR0, контроллер прерываний PLIC, приёмопередатчик UART0, АЦП со встроенным датчиком температуры, ЦАП, блок шифрования CRYPTO и PLL. Регистр `misa`, прочитанный с микроконтроллера через JTAG, равен `0x4014912D`, что соответствует RV32 с расширениями A, C, D, F, I, M, P, S и U.

2.2. Отладочная плата и адаптер

Работа выполнена на отладочной плате НИИЭТ КФДЛ.441461.050 rev 1.1 (в SDK — `NIIET-DEV-K1921VG1T`). На плате установлены кварц 27 МГц для генератора HSE, светодиоды LED1–LED4 на выводах C12–C15, кнопка на выводе B3, мост USB–UART CH340B, подключённый к UART0 микроконтроллера, разъём SMA выхода ЦАП и разъём JTAG XP7. Встроенного программатора на плате нет.

Для прошивки и отладки использован JTAG-адаптер FT2232HL Board V3.6 на микросхеме FTDI FT2232HL. JTAG на этом адаптере разведён на канал B; в Windows для этого канала установлен драйвер WinUSB. Таблица соединений адаптера с платой приведена в приложении Б. По документу НИИЭТ «Быстрый старт с микроконтроллерами К1921ВГ015, К1921ВГ1Т…» [8] частота TCK для К1921ВГ1Т должна быть не выше 200–250 кГц; в работе использовалась частота 200 кГц.

2.3. Программные средства

Таблица 1 — Программные средства и их версии

Средство	Версия
Rust (rustc, cargo), цель `riscv32imac-unknown-none-elf`	1.97.1
svd2rust, form, rustfmt — генерация PAC	0.37.1, 0.13.0, 1.9.0
riscv-rt — среда выполнения Rust-примеров	0.18.0
OpenOCD НИИЭТ (форк с драйвером Flash-памяти `k1921vg1t`)	`-00247-g90632ca40-dirty` (2025-08-01; так версию выводит сама программа)
SDK НИИЭТ `k1921vg1t_sdk` — для примеров на C	коммит `9fa40f85be5b`
Компилятор C: xPack riscv-none-elf-gcc	13.4.0-1
GNU Make	3.81
Task — запуск команд сборки и прошивки	3.54.0
Операционная система	Windows 11

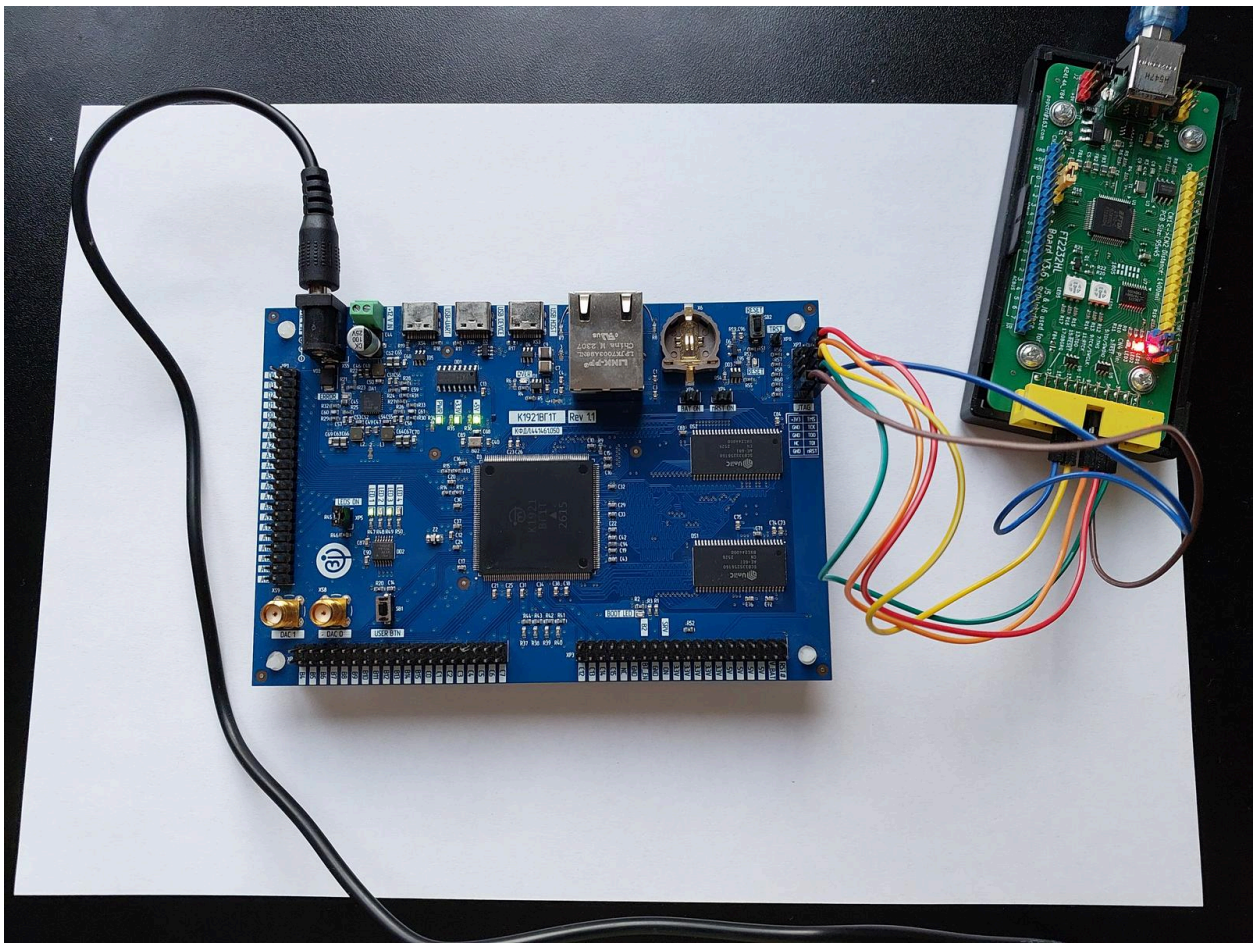


Рисунок 1 — Стенд: отладочная плата К1921ВГ1Т (слева) и JTAG-адаптер FT2232HL Board (справа); плата питается от источника 12 В через разъём XS5

Обычный OpenOCD для этого микроконтроллера не подходит: конфигурация использует драйвер Flash-памяти `k1921vg1t`, который есть в форке НИИЭТ [9]. Версия Rust закреплена в файле `rust-toolchain.toml`, поэтому rustup устанавливает нужную версию компилятора и цель автоматически.

Встроенная цель компилятора, под которую собираются Rust-примеры, — `riscv32imac-unknown-none-elf`: компилятор не использует команды с плавающей точкой, хотя ядро их поддерживает. В скриптах компоновки примеров заданы 1024 КБ Flash-памяти с адреса `0x00000000` и 512 КБ ОЗУ с адреса `0x20000000` — меньше, чем есть в микроконтроллере, но с запасом достаточно для примеров. Примеры используют одно ядро из двух.

2.4. Организация репозитория и проверка сборки

Все материалы работы опубликованы в репозитории (см. титульный лист): PAC, примеры на Rust и C, конфигурации OpenOCD, описание дефектов SVD и исходный текст этой записки. Команды сборки и прошивки Rust-примеров собраны в файле `Taskfile.yml` для утилиты Task. При каждом изменении репозитория сервис GitHub Actions на Linux и на Windows собирает PAC и все Rust-примеры и проверяет, что PAC воспроизводится из SVD-описания побайтно (подраздел 3.4).

CC BY 4.0

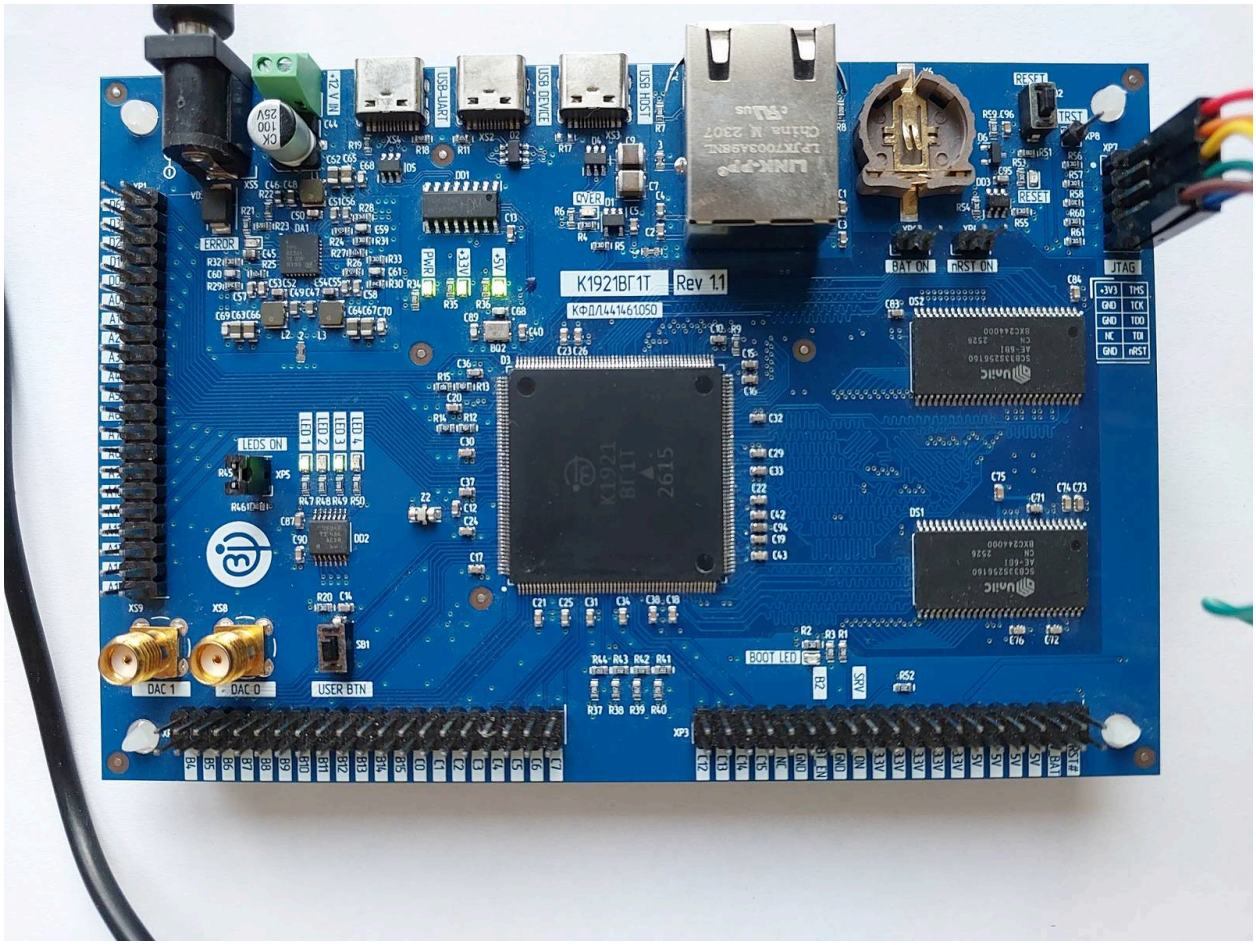


Рисунок 2 — Отладочная плата КФДЛ.441461.050 rev 1.1: вверху слева — разъём питания XS5 и разъём USB-UART XS4, вверху справа — кнопка RESET (SB2) и рядом с ней разъём JTAG XP7; слева — светодиоды LED1–LED4 с перемычкой XP5 «LEDS ON»; внизу слева — кнопка USER BTN и разъёмы SMA «DAC 0» (XS8) и «DAC 1» (XS9) — выходы ЦАП

CC BY 4.0

3. Rust-PAC из исправленного SVD

3.1. Что такое PAC и зачем он нужен

Чтобы программа на Rust работала с периферией микроконтроллера без единой строки на C, ей нужно описание регистров: адреса блоков, смещения регистров, назначение и доступ каждого поля. В экосистеме embedded Rust это описание оформляется как отдельный крейт доступа к периферии (PAC, Peripheral Access Crate). Его не пишут вручную: инструмент svd2rust [10] генерирует PAC из SVD-описания микроконтроллера — XML-файла в формате CMSIS-SVD [5], который производитель обычно поставляет вместе с SDK.

PAC проверяет обращения к регистрам при компиляции: у каждого регистра есть свой тип, поля доступны только через методы этого типа, поле только для чтения нельзя записать. Если в SVD для поля перечислены допустимые значения, PAC даёт для каждого из них именованный метод: например, длина слова UART задаётся вызовом `w.wlen()._8bit()`, а не числом. Чем эти проверки отличаются от проверок при работе с заголовочным файлом C, показано в подразделе 5.4. Для К1921ВГ1Т готового PAC в открытом доступе не было (подробнее — в подразделе 1.2), поэтому его пришлось сгенерировать в этой работе.

3.2. Исходные данные

SVD-описание К1921ВГ1Т входит в SDK производителя [2] (`tools/svd/K1921VG1T.svd`). В работе использована версия SDK на коммите `9fa40f85be5b` (14.07.2026); в самом файле указана версия описания 2.8 (тег `<version>`). Генерация выполнялась svd2rust 0.37.1; полученный код разбивался на модули инструментом form 0.13.0 и форматировался rustfmt 1.9.0.

Исходный файл svd2rust не принимает: первая же попытка генерации завершается ошибкой разбора. Исправление SVD стало основной частью работы над PAC.

3.3. Исправление SVD

Все изменения внесены в копию файла; оригинал в SDK не изменялся. Исправленный файл `K1921VG1T.patched.svd` и полный перечень правок в виде diff к оригиналу лежат в репозитории рядом с PAC. Подробный разбор каждой правки с номерами строк исходного файла, основаниями и ссылками на страницы руководства пользователя приведён в документе `projects/k1921vg1t-pac/SVD_FIXES.md`.

Дефекты, без исправления которых PAC не генерируется или не компилируется, сведены в таблицу 2; все места исправлений с именами регистров перечислены в приложении A.

Таблица 2 — Дефекты SVD, препятствующие генерации PAC (48 мест)

№	Дефект	Мест	Что происходит без исправления
1	Недопустимое значение порядка байт `Endian.LITTLE`	1	svd2rust: `Unknown endianness`
2	Пустой обязательный тег `nvicPrioBits`	1	svd2rust: `Expected content in <nvicPrioBits> tag`
3	Регистры без имени	30	svd2rust: `Expected a <name> tag`
4	Регистры нулевой ширины	8	svd2rust: `Reset value 0x10102 doesn't fit in 0 bits`
5	Символы `<>` в имени регистра	2	svd2rust аварийно завершается
6	Регистр «только запись» с читаемыми полями	4	svd2rust: `Field ... is not writtable or readable` (написание svd2rust)
7	Шаг массива `TMR0.CAPCOM` меньше размера элемента	1	svd2rust: размер элемента больше шага массива
8	Кластер `ADC.ACTL` и его регистр названы одинаково	1	rustc: рекурсивный тип бесконечного размера (E0072)

Необходимость каждого класса правок проверена отдельно: в исправленном файле откатывалась правка только одного класса, после чего запускались svd2rust и компилятор. Из восьми регистров нулевой ширины генерацию останавливает только один; остальные семь svd2rust принимает без ошибок, но код для них получается неверным, поэтому исправлены все восемь. Класс 8 стандарт CMSIS-SVD прямо не запрещает; правка нужна из-за того, как svd2rust строит имена типов.

Помимо дефектов, мешающих генерации, сверка исправленного файла с руководством пользователя и с заголовком `K1921VG1T.h` из того же SDK выявила ещё семь расхождений. Часть из них исправлена, часть только описана; они перечислены в таблице 9 и разобраны в подразделе 6.1.1.

В SVD-описании более позднего SDK производителя часть дефектов исправлена (подраздел 6.3), но классы 6–8 сохраняются, и svd2rust этот файл также не принимает. Поэтому исправленное описание нужно и при новом SDK.

3.4. Генерация и воспроизводимость

PAC получается из исправленного SVD тремя командами:

```
svd2rust --target riscv -i K1921VG1T.patched.svd
form -i lib.rs -o src/ && rm lib.rs
rustfmt --edition 2021 --config newline_style=Unix src/lib.rs build.rs
```

Сгенерированный код в репозитории не правился вручную. Результат воспроизводим побайтно: при тех же версиях инструментов повторная генерация даёт файлы, совпадающие с репозиторием. Без параметра `newline_style=Unix` rustfmt на Windows записывает файлы с концами строк CRLF, и побайтное совпа-

дение между операционными системами теряется; параметр устраняет эту зависимость. Версия компилятора и rustfmt закреплена в файле `rust-toolchain.toml`. Воспроизводимость проверяется автоматически: при каждом изменении репозитория сервис GitHub Actions на Linux и на Windows заново генерирует PAC из SVD и сравнивает результат с файлами репозитория.

3.5. Состав PAC

Таблица 3 — Объём исправленного SVD-описания и сгенерированного PAC

Показатель	Значение
Периферийных блоков (из них описаны как производные от других)	106 (72)
Описаний регистров в непроизводных блоках	578
Описаний полей	3104
Источников прерываний	251
Файлов сгенерированного кода	656
Строк сгенерированного кода	82 123

Использование PAC показано в листинге 1: включение тактирования порта GPIOC и мигание светодиодом из примера `rust_blink_pac`. Регистры и поля называются по SVD-описанию, а неверное имя поля или попытка записать поле только для чтения приводят к ошибке компиляции.

3.6. Ограничения

- На плате PAC проверен только в той части, которую используют примеры репозитория (блоки RCU, FLASHM, GPIO, SIU, UART0, TMR0, ADC, DAC0, CRYPTO); все примеры запущены на плате с текущей версией PAC (раздел 4). Полная сверка всех регистров с руководством пользователя не проводилась.
- Контроллер прерываний PLIC в SVD не описан, поэтому в PAC его нет; пример `rust_timer_irq` обращается к регистрам PLIC по адресам.
- Необязательная функция (feature) `rt` крейта (таблица векторов прерываний периферии) собирается, но примерами не используется и на плате не проверялась. Для неё svd2rust 0.37.1 подключает `riscv-rt` 0.13, а примеры используют более новую версию 0.18 (таблица 1); совместимость этих версий не проверялась.
- Имена `VALUE` для 30 безымянных регистров и `CTRL` для регистра управления АЦП выбраны в работе; в руководстве пользователя их нет.

CC BY 4.0

```rust
#[entry]
fn main() -> ! {
    // steal(): no singleton check (the program is single-threaded).
    let p = unsafe { pac::Peripherals::steal() };

    // Enable the GPIOC clock and release its reset (read-modify-write of one bit).
    p.rcu.cgcfgahb().modify(|_, w| w.gpiocen().set_bit());
    p.rcu.rstdisahb().modify(|_, w| w.gpiocen().set_bit());

    // C12..C15 as outputs.
    p.gpioc.outenset().write(|w| {
        w.pin12().set_bit();
        w.pin13().set_bit();
        w.pin14().set_bit();
        w.pin15().set_bit()
    });

    loop {
        // DATAOUT has a 16-bit `val` field; bit 12 is LED1.
        p.gpioc.dataout().write(|w| unsafe { w.val().bits(0x1000) });
        for _ in 0..500_000 {
            unsafe { core::arch::asm!("nop") };
        }
        p.gpioc.dataout().write(|w| unsafe { w.val().bits(0x0000) });
        for _ in 0..500_000 {
            unsafe { core::arch::asm!("nop") };
        }
    }
}
```

Листинг 1 — Мигание светодиодом через PAC (файл `projects/rust_blink_pac/src/main.rs`)

4. Работа с периферией

В этом разделе описано, как в Rust-примерах настраивается каждый блок микроконтроллера и что было получено на плате. Все результаты ниже получены на одной плате 9 и 10 октября 2026 г. с прошивками, собранными из текущей версии репозитория; исключение — первые замеры частоты, описанные в подразделе 4.4. Примеры с текстовым выводом передают его через UART0 и мост CH340B на компьютер со скоростью 9600 бит/с; журналы терминала записывались скриптом.

Таблица 4 — Rust-примеры и результаты проверки на плате

Пример	Блоки	Результат на плате
`rust_blink`, `rust_blink_rt`, `rust_blink_pac`	GPIO	LED1 мигает
`rust_timer_irq`	TMR0, PLIC, GPIO	LED1 переключается из обработчика прерывания
`rust_uart`	UART0, TMR0	частота HSI ≈ 3,34 МГц
`rust_pll`	PLL0, FLASHM, UART0, TMR0	системная частота 150,0 МГц
`rust_adc`	АЦП, датчик температуры	20,6 °C и 21,2 °C
`rust_dac`	ЦАП DAC0	синусоида от 0,003 В до 2,938 В
`rust_crypto`	CRYPTO	шифртекст совпал с тестовым вектором ГОСТ Р 34.12-2015

4.1. Включение блока

После сброса микроконтроллера периферийные блоки, использованные в примерах, не тактируются и удерживаются в сбросе: значения сброса регистров `CGCFGAHB`, `CGCFGAPB1`, `CGCFGAPB2`, `RSTDISAHB`, `RSTDISAPB1` и `RSTDISAPB2` равны нулю (РП, с. 521–528). Перед обращением к регистрам блока нужно в блоке управления тактированием RCU подать на блок тактовый сигнал (регистры `CGCFGAHB`, `CGCFGAPB1`, `CGCFGAPB2` — по шине, к которой подключён блок) и снять с него сброс (регистры `RSTDISAHB`, `RSTDISAPB1`, `RSTDISAPB2`, тот же бит). Например, для порта GPIOC:

```
p.rcu.cgcfgahb().modify(|_, w| w.gpiocen().set_bit());
p.rcu.rstdisahb().modify(|_, w| w.gpiocen().set_bit());
```

Таймерам этого мало: таймер начинает счёт только после установки его бита в регистре `TMREN` системного блока SIU. Блоки UART и АЦП имеют, кроме того, собственные регистры выбора источника тактового сигнала в RCU (`UARTCFG`, `ADCCFG`).

CC BY 4.0

4.2. Порты ввода-вывода

Светодиоды LED1–LED4 подключены к выводам C12–C15 и горят при единице на выводе (проверка полярности описана в подразделе 6.1.2). Чтобы управлять светодиодом, вывод переводят в режим выхода (регистр `OUTENSET`) и меняют его состояние: записью в регистр данных `DATAOUT` целиком или регистрами `DATAOUTSET`, `DATAOUTCLR` и `DATAOUTTGL` — установка, сброс и переключение отдельных выводов без чтения-изменения-записи. Примеры мигания для простоты пишут в `DATAOUT` целиком: `rust_blink` — без единой внешней библиотеки, со своим стартовым кодом, `rust_blink_rt` — на среде выполнения `riscv-rt`, `rust_blink_pac` — через PAC.

Если программа не настраивает порт C, все четыре светодиода слабо светятся. Так было при запуске примеров `rust_uart`, `rust_adc` и `rust_dac`. На результаты примеров это не влияет; причина свечения в работе не исследовалась.

4.3. Прерывание от таймера

Пример `rust_timer_irq` переключает LED1 из обработчика прерывания таймера TMR0. Таймер считает такты системной частоты до значения регистра периода `TMR0.PERIOD`, после чего формирует запрос прерывания. Запрос проходит через контроллер прерываний платформы PLIC и попадает в ядро как внешнее прерывание машинного режима (бит `MEIE` регистра `mie` и бит `MIE` регистра `mstatus`).

Контроллера PLIC в PAC нет (подраздел 3.6), поэтому пример обращается к нему по адресам. Адреса взяты из заголовочного файла `plic.h` SDK НИИЭТ. TMR0 — источник прерывания с номером 16. Кроме регистров, описанных в стандартной спецификации PLIC (приоритет, разрешение, порог, захват), в PLIC этого микроконтроллера есть регистр режима источника (`SRC_MODE`, адрес `0xFE1F0000` + 4 · номер): в нём задаётся, срабатывает ли источник по уровню или по фронту. Пока режим не задан, источник выключен. Обработчик выполняет протокол PLIC «захват — завершение»: читает номер источника, переключает светодиод, сбрасывает флаг прерывания в таймере (иначе прерывание сразу повторится) и записывает номер обратно.

При периоде 1 000 000 тактов и частоте HSI около 3,34 МГц (см. следующий подраздел) светодиод переключается раз в 0,3 с; вспышка — это два переключения, 0,6 с, то есть светодиод должен вспыхивать около 17 раз за 10 с. При ручном подсчёте получено около 18 вспышек за 10 с — с точностью такого подсчёта это совпадает.

4.4. UART и частота внутреннего генератора

После сброса ядро тактируется от внутреннего RC-генератора HSI. Его частота неточна, поэтому UART0 в примерах тактируется от кварцевого генератора HSE

на 27 МГц: делитель для 9600 бит/с равен 27 000 000 / (16 · 9600) = 175,78125, то есть целая часть 175 и дробная 50/64 — скорость получается точно 9600 бит/с.

Раз UART тактируется от кварца, им можно измерить частоту HSI. Таймер TMR0 считает такты ядра, то есть HSI, а программа передаёт через UART N = 2000 байт. Каждый байт в формате 8N1 занимает 10 бит, поэтому передача длится ровно N · 10 / 9600 с, и частота HSI равна числу тактов таймера за это время, умноженному на 9600 / (N · 10). Такой же замер выполняет пример `rust_pll` для частоты после переключения на PLL.

У передатчика UART есть буфер FIFO на 32 байта (РП, с. 350). Если в момент запуска замера в буфере ещё лежат байты предыдущего сообщения, таймер считает и время их передачи, и результат получается завышенным. В первой версии примеров, запущенной на плате 9 октября, буфер перед замером не опустошался, и в журнале оказались завышенные значения. Их можно пересчитать: перед первым замером в `rust_pll` в буфере оставалась вся строка `"\r\nBefore PLL: "` (14 байт с переводом строки), перед вторым — строка `"After PLL:  "` (12 байт), а в `rust_uart` буфер был полон — 32 байта и байт в сдвиговом регистре. Значит, таймер считал время передачи не 2000, а 2000 + k байт (k — число байт, оставшихся в буфере), и настоящая частота меньше записанной в (2000 + k) / 2000 раз. С учётом этого частота PLL должна быть 150 899 602 · 2000 / 2012 = 149 999 604 Гц, то есть 150 МГц, а частота HSI — около 3,337 МГц (столбец «С поправкой» в таблице 5). Затем в оба примера было добавлено ожидание пустого буфера перед замером, и 10 октября замеры повторены (по два запуска каждого примера).

Таблица 5 — Частоты, измеренные без ожидания пустого буфера (9 октября), те же значения с поправкой на содержимое буфера (значение · 2000 / (2000 + число байт)) и замеры с ожиданием (10 октября)

Замер	Без ожидания, Гц	Байт в буфере	С поправкой, Гц	С ожиданием, Гц
HSI, `rust_uart`	3 391 883	33	3 336 825	3 338 247 3 338 296
HSI, `rust_pll` до PLL	3 359 635	14	3 336 281	3 338 045 3 337 142
PLL0, `rust_pll`	150 899 602	12	149 999 604	150 000 459 150 000 449

Пересчитанные значения совпали с новыми замерами в пределах 0,06 %, а частота PLL во всех случаях отличается от расчётных 150 МГц примерно на 3 миллионные доли — в пределах точности кварца. Это подтверждает и сам способ замера, и объяснение ошибки.

Частота HSI на плате — около 3,34 МГц; РП приводит для неё два разных значения, 4 МГц и 3,375 МГц (подраздел 6.1.3). Рассчитывать делители от 4 МГц нельзя: UART со скоростью, рассчитанной для 4 МГц, на этой плате работал бы

на 16,6 % медленнее, чем нужно, а приёмник допускает рассогласование лишь в несколько процентов.

4.5. PLL

Пример `rust_pll` переводит системную частоту на PLL0 с настройками SDK НИИЭТ для кварца 27 МГц. По формуле SDK

$$F = F_{\mathrm{HSE}} \cdot \frac{\mathrm{NDIV}}{(\mathrm{RDIV}+1)\cdot(\mathrm{DIV1A}+1)\cdot(\mathrm{DIV1B}+1)} = 27\ \text{МГц} \cdot \frac{100}{3\cdot 2\cdot 3} = 150\ \text{МГц}.$$

Порядок действий важен: если, например, переключить системную частоту раньше, чем PLL захватит частоту, или не увеличить число тактов ожидания Flash-памяти, программа не будет работать правильно. Пример выполняет шаги в таком порядке:

1. включить HSE и подождать запуска кварца;
2. записать делители и множитель в регистры `PLL[0].DIV` и `PLL[0].CFG`;
3. подождать захвата частоты;
4. увеличить число дополнительных тактов ожидания при чтении Flash-памяти (поле `LAT` регистра `FLASHM.CTRL`): по таблице 9.1 РП (с. 76) для частоты до 140 МГц нужно 3 такта, до 175 МГц — 4; пример задаёт 4 (SDK задаёт 2, см. подраздел 6.1.2);
5. переключить системную частоту на PLL0 (записать 2 в поле `SRC` регистра `RCU.SYSCLKCFG`) и дождаться, пока регистр состояния `RCU.CLKSTAT` подтвердит переключение.

Замер частоты после переключения приведён в таблице 5: 150 МГц с точностью кварца.

При настройке PLL проявилась особенность PAC, сгенерированного svd2rust. Метод `write()` начинает запись со значения сброса регистра и меняет только явно указанные поля. Значение сброса берётся из SVD-описания; для регистра `RCU.PLL[0].CFG` оно равно `0x9010F001`, и бит 0 (`PD`, отключение PLL) в нём установлен. Первая версия примера `rust_pll` использовала `write()` и не трогала поле `PD`, поэтому PLL оставался выключенным, и программа бесконечно ждала захвата частоты. Исправление — метод `write_with_zero()`, который начинает запись с нуля, как присваивание регистру значения в C. Этот метод помечен как `unsafe`.

4.6. АЦП и датчик температуры

Встроенный датчик температуры подключён к каналу 63 АЦП (РП, с. 300). Пример `rust_adc` включает аналоговую часть блока ADC3 с разрешением 12 бит, включает датчик (регистр `SIU.TSENSCTRL`) и настраивает секвенсор 0 на одно программно запускаемое преобразование канала 63. Результат забирается из буфера секвенсора, когда счётчик заполнения буфера (`SFLOAD`) становится ненулевым.

У датчика высокое выходное сопротивление. По РП (с. 302) для достоверного результата АЦП нужно тактировать частотой не выше 180 кГц либо задержать

начало преобразования канала регистром CHDELAY. Пример тактирует АЦП от кварца 27 МГц и использует задержку.

Код АЦП переводится в напряжение в предположении, что опорное напряжение равно 3,3 В, а напряжение — в температуру по формуле линейной регрессии из РП для режима датчика SCALE = 1: $U = 0{,}793 - 0{,}00174 \cdot T$ (таблица 21.1, с. 301). На плате получены значения 20,6 и 21,2 °C с интервалом около 15 с. Эталонного термометра не было, поэтому можно сказать только, что значения правдоподобны для комнаты.

4.7. ЦАП

Пример rust_dac выводит на выход ЦАП DAC0 (разъём SMA «DAC 0» на плате) синусоиду из 64 точек с кодами от 1 до 4095. ЦАП работает от внутреннего источника опорного напряжения с настройкой 2,5 В (поле ADJUST = 3 регистра BGCTRL, РП, с. 756) и с включённым выходным буфером; каждое преобразование запускается программно битом SWTRIG. Шаг сделан медленным, чтобы напряжение можно было отследить мультиметром: полупериод длится около 77 с.

Мультиметр между центральным контактом и корпусом разъёма показал минимум 0,003 В и максимум 2,938 В. Максимум заметно выше выбранных 2,5 В. Причина в работе не выяснялась: замер выполнен мультиметром на одной плате, а таблица электрических параметров ЦАП в РП (таблица 20.1) не позволяет однозначно определить ожидаемое значение.

4.8. Блок шифрования

Блок CRYPTO аппаратно выполняет AES-128, AES-256, «Кузнечик» и «Магму». Пример rust_crypto шифрует блок алгоритмом «Кузнечик» в режиме ECB тестовым ключом из ГОСТ Р 34.12-2015 [11] и сравнивает результат с эталонным шифртекстом из стандарта, затем расшифровывает его обратно. Журнал с платы:

```
Plaintext:  11223344 55667700 FFEEDDCC BBAA9988
Ciphertext: 7F679D90 BEBC2430 5A468D42 B9D4EDCD
Expected:   7F679D90 BEBC2430 5A468D42 B9D4EDCD
MATCH: ciphertext equals the GOST test vector
Decrypted:  11223344 55667700 FFEEDDCC BBAA9988
OK: decrypted == plaintext
```

Ключ и данные загружаются в регистры блока 32-битными словами в том порядке, в котором они записаны в стандарте, без перестановки байт. Проверка только обратным расшифрованием для этого недостаточна: она проходит и при неверном порядке байт (см. таблицу 10), поэтому сравнение с эталоном стандарта обязательно.

CC BY 4.0

5. Оценка применимости Rust (no_std)

Применимость Rust для bare-metal разработки под К1921ВГ1Т оценивалась на девяти примерах репозитория по пяти вопросам: есть ли инструменты и поддержка микроконтроллера; каков размер прошивок по сравнению с C; где приходится отказываться от проверок компилятора (`unsafe`); какие ошибки компилятор находит сам; сколько времени занимает сборка. Замеры выполнены на ноутбуке с процессором Intel Core i7-1065G7 и 8 ГБ ОЗУ под Windows 11, версии инструментов приведены в таблице 1.

5.1. Инструменты и поддержка микроконтроллера

Компилятор Rust с целью `riscv32imac-unknown-none-elf` устанавливается штатно через rustup, отдельный компилятор для RISC-V не нужен. Готовые библиотеки экосистемы подошли без изменений: среда выполнения `riscv-rt` [12] (стартовый код, таблица исключений, обработчики прерываний ядра) и крейт `riscv` (регистры CSR). Собранный cargo файл ELF прошивается тем же OpenOCD НИИЭТ, что и прошивки на C.

Поддержки самого микроконтроллера в экосистеме Rust не было: ни PAC, ни HAL для К1921ВГ1Т не найдено (подраздел 1.2). PAC пришлось сгенерировать, а SVD-описание производителя генератор svd2rust не принял без исправлений (подраздел 3.3). Контроллера прерываний PLIC в PAC нет (подраздел 3.6).

Набор команд ядра используется не полностью. Во встроенных целях Rust 1.97.1 для 32-битного RISC-V без операционной системы есть варианты без плавающей точки и `riscv32imafc-unknown-none-elf` с одинарной точностью; цели с расширением D (двойная точность) нет. Примеры собраны под `riscv32imac` и не используют блок плавающей точки, что для них не важно. Использовать двойную точность можно только со своим описанием цели.

5.2. Размер прошивки

Пример `rust_blink` со своим стартовым кодом занимает 70 байт кода. Переход на `riscv-rt` добавляет около 420 байт: стартовый код с обработкой исключений и таблицу обработчиков исключений. Переход на PAC добавляет 20 байт. По дизассемблированному коду цикл мигания в `rust_blink_rt` и `rust_blink_pac` состоит из одинаковых команд: обращение к регистру по имени компилируется в ту же запись по адресу. Разница — два чтения регистров RCU (пример с PAC меняет в них только нужный бит методом `modify()`, а пример без PAC записывает их целиком) и запись байта-флага, которым PAC отмечает, что периферия уже выдана программе.

Для сравнения с C взят пример `binary_counter`, который, как и `rust_blink_pac`, работает с регистрами GPIO напрямую. При сборке по Makefile из репозитория (оптимизация `-Og`, все исходные файлы SDK) он занимает 894 байта

Таблица 6 — Rust-примеры: строки кода в `main.rs` и `build.rs` без пустых строк и комментариев, размеры секций (утилита `riscv-none-elf-size` с ключом `-A`), число зависимостей в дереве cargo и блоков `unsafe`; сборка с `opt-level = "s"`

Пример	Строк кода	`.text`, байт	`.rodata`, байт	Крейтов	Блоков `unsafe`
`rust_blink`	49	70	0	0	1
`rust_blink_rt`	41	488	112	11	1
`rust_blink_pac`	38	508	112	17	5
`rust_timer_irq`	68	600	112	17	5
`rust_uart`	92	3 540	464	17	6
`rust_adc`	125	2 720	404	17	12
`rust_dac`	52	556	240	17	4
`rust_crypto`	125	1 372	328	17	9
`rust_pll`	133	3 720	428	17	14

кода и 2064 байта данных; данные — это таблицы обработчиков прерываний PLIC из SDK, которые попадают в прошивку, потому что Makefile компилирует все файлы SDK. В варианте `Makefile_fit` с явным списком файлов SDK — 600 байт кода и 0 байт данных. Программы делают разное, а C собирается под набор команд с плавающей точкой (`rv32imafdc`), поэтому сравнение показывает только порядок величин: простая программа на Rust и на C занимает сотни байт, размеры сопоставимы.

Размер заметно растёт от двух скрытых затрат, которые видны в таблице символов прошивки:

- В примерах, которые выводят десятичные числа (`rust_uart`, `rust_adc`, `rust_pll`), компилятор не может доказать, что индекс массива цифр не выходит за границы, и вставляет проверку. Ветка ошибки вызывает панику, а она тянет за собой код форматирования `core::fmt` — около 1,5 КБ, хотя обработчик паники сообщение не использует. В `rust_crypto`, где выводятся только шестнадцатеричные числа, такого кода нет.
- В `rust_uart` и `rust_pll` частота считается в 64-битной арифметике. В наборе команд RV32 нет деления 64-битных чисел, и компилятор подключает программное деление — около 1 КБ.

Обе затраты обычны для Rust на 32-битных микроконтроллерах; их можно убрать, переписав соответствующие функции, но в примерах код намеренно оставлен простым.

5.3. Где нужен unsafe

Блок `unsafe` отключает часть проверок компилятора: программист берёт на себя ответственность за корректность кода внутри блока. В девяти примерах 57 таких блоков. По содержимому они делятся на шесть групп (один блок может попасть в несколько групп):

Таблица 7 — Блоки `unsafe` в Rust-примерах

Что внутри блока	Блоков	Можно ли обойтись без `unsafe`
Запись числа в поле регистра методом `bits()`	35	да, если описать в SVD допустимые значения поля
Задержка циклом с командой `nop` (`asm!`)	13	да: безопасные `riscv::asm::nop()` и `riscv::asm::delay()` (нужна зависимость от крейта `riscv`)
Получение периферии `Peripherals::steal()`	8	да: метод `take()` при включённой функции (feature) `critical-section` PAC
Обращение по адресу через указатель	4	да: через PAC; в `rust_blink` и `rust_blink_rt` указатели оставлены намеренно (это шаги до PAC), а для PLIC его сначала нужно описать в SVD
`write_with_zero()`	2	да: `write()` с явно заданными значениями всех полей регистра
Разрешение прерываний через крейт `riscv`	1	нет: эти функции `unsafe` по построению

Больше половины блоков приходятся на запись чисел в поля регистров. svd2rust помечает метод `bits()` небезопасным всегда: генератор не знает, допустимо ли произвольное значение поля. Безопасный метод `set()` появляется, только если допустимые значения описаны в SVD: ограничением записи `writeConstraint` (тогда `set()` проверяет диапазон во время выполнения) или перечислением всех значений поля (тогда `set()` ничего не проверяет и просто отбрасывает лишние старшие биты числа). В SVD-описании К1921ВГ1Т нет ни одного `writeConstraint`. Например, поле делителя скорости UART `IBRD.DIVINT` не имеет перечисления, поэтому его можно записать только методом `bits()` в блоке `unsafe`: `w.divint().bits(175)`. В портах GPIO у всех регистров установки (`DATAOUTSET`, `OUTENSET` и других) есть отдельное поле для каждого вывода, а у регистров сброса (`DATAOUTCLR`, `OUTENCLR` и других), переключения `DATAOUTTGL` и данных `DATAOUT` — одно общее поле `VAL`; так же их описывает и РП (с. 650–651). Поэтому светодиод через PAC включается без `unsafe` (`w.pin12().set_bit()` в `DATAOUTSET`), а выключается только через `unsafe { w.val().bits(1 << 12) }` в `DATAOUTCLR`. Дополнение SVD ограничениями записи и единообразное описание полей сделали бы большую часть этих блоков ненужной.

В итоге неизбежен только блок с разрешением прерываний, а обращения к PLIC по адресам останутся, пока PLIC не описан в SVD. Остальные блоки связаны с неполнотой SVD-описания или с упрощениями в примерах.

CC BY 4.0

5.4. Что проверяет компилятор

Чтобы проверить, какие ошибки PAC ловит при компиляции, в копию примера `rust_uart` были внесены четыре типичные ошибки. Все четыре найдены компилятором:

```
p.uart0.fr().write(|w| w);                          // FR только для чтения
p.uart0.lcrh().modify(|_, w| w.wlen()._9bit());     // у поля WLEN нет значения 9 бит
p.uart0.cr().modify(|_, w| w.txen().set_bit());     // поле называется TXE, а не TXEN
p.uart0.lcrh().modify(|_, w| w.wlen().bits(3));     // bits() вне блока unsafe
```

О первых трёх ошибках компилятор сообщает как об отсутствии метода (`error[E0599]`): у регистра только для чтения нет метода `write()`, у поля — метода с именем несуществующего значения, у регистра — метода с неверным именем поля. Четвёртая ошибка — вызов небезопасной функции вне `unsafe` (`error[E0133]`).

Для честного сравнения те же ошибки были записаны на C с заголовочным файлом SDK `K1921VG1T.h` и скомпилированы GCC с ключами `-Wall -Wextra`. Если писать через структуры битовых полей и перечисления заголовка, C находит первые три ошибки тоже: регистр `FR` объявлен как `volatile const`, а неверные имена поля и значения не компилируются. Разница проявляется, когда в поле записывается число:

```
UART0->LCRH_bit.WLEN = 9;               // предупреждение: значение обрезано до 1
UART0->LCRH = 9 << UART_LCRH_WLEN_Pos;  // без предупреждений
```

Первая строка компилируется с предупреждением, вторая — вообще без предупреждений, хотя она задаёт неверную длину слова и устанавливает бит за пределами поля. В PAC для поля с перечислением естественно писать именованное значение (`w.wlen()._8bit()`), и ошибиться в нём нельзя. Но число записать тоже можно, и в этом PAC не лучше C: у полей, где перечислены все значения, как у `WLEN`, есть безопасный метод `set()`, и `w.wlen().set(9)` компилируется без ошибок и предупреждений, а в поле записывается 1. В поле без перечисления число записывается только методом `bits()` в блоке `unsafe`, который виден при чтении кода.

Компилятор не проверяет смысл записи. В PAC `write()` начинает со значения сброса регистра, а не с нуля, как присваивание в C; из-за этого в примере `rust_pll` PLL не запускался (подраздел 4.5). Такие различия нужно знать, переходя с C на Rust.

5.5. Время сборки

Чистая сборка — сборка с нуля, вместе со всеми зависимостями, как при первой сборке после клонирования репозитория; повторная — после правки программы, когда зависимости уже скомпилированы.

Таблица 8 — Время сборки (чистая сборка Rust — минимум из трёх запусков, C — один запуск)

Сборка	Время, с
`rust_blink`, чистая сборка, без зависимостей	1,5
`rust_blink_rt`, чистая сборка, 11 крейтов	18
Примеры с PAC, чистая сборка, 17 крейтов	27–29
`rust_blink_pac`, повторная сборка после правки `main.rs`	около 1
C-примеры, чистая сборка с файлами SDK	1,7–6,3

Чистая сборка примера с PAC занимает около 28 с. Компиляция самого PAC — 82 тысячи строк сгенерированного кода — занимает из них 9,9 с, остальное приходится на другие зависимости, которые компилируются параллельно. Повторная сборка после правки программы занимает около секунды, потому что зависимости уже скомпилированы. Для работы с примерами такое время сборки приемлемо.

5.6. Вывод

Rust (no_std) применим для bare-metal разработки под К1921ВГ1Т: стандартные инструменты устанавливаются без доработок, все девять примеров, от мигания светодиодом до аппаратного шифрования и переключения на PLL, работают на плате, размер прошивок того же порядка, что у C, а PAC проверяет при компиляции имена регистров, полей и значений и делает запись произвольного числа в поле без перечисления видимой в коде как блок `unsafe`. От записи неверного числа через `set()` в поле с перечислением он не защищает. Ограничения связаны не с языком, а с состоянием поддержки микроконтроллера: PAC пришлось создавать и исправлять для этого SVD-описание, PLIC в нём не описан, HAL нет, а неполнота описания полей заставляет писать `unsafe` там, где он мог бы быть не нужен.

6. Найденные дефекты документации и SDK

В ходе работы обнаружены расхождения между материалами производителя и поведением микроконтроллера, а также между самими материалами: SVD-описанием, заголовочным файлом SDK, примерами и руководством пользователя (РП). Ниже они разделены на две группы: дефекты материалов производителя (подраздел 6.1) и собственные ошибки, допущенные в работе и исправленные (подраздел 6.2). Все сведения о SDK относятся к `k1921vg1t_sdk` на коммите `9fa40f85be5b`, если не указано иное.

6.1. Дефекты материалов производителя

6.1.1. SVD-описание

Дефекты SVD-описания — самая многочисленная группа. Восемь классов дефектов, мешающих получить PAC (всего 48 мест исправлений), описаны в подразделе 3.3. Ещё семь расхождений выявлены сверкой исправленного описания с РП и заголовком `K1921VG1T.h` (таблица 9; в ней номера 9–15 продолжают нумерацию классов 1–8 из таблицы 2). Все места исправлений перечислены в приложении А.

Таблица 9 — Расхождения SVD-описания с РП и заголовком SDK

№	Расхождение	Подтверждение	Решение
9	Размер блока адресов WDT и TMR0 меньше, чем занимают их регистры	РП, с. 807 и 673	исправлено
10	В блоке DAC0 описан регистр внешней микросхемы ЦАП MCP4921	РП, с. 759; `K1921VG1T.h`	описано
11	Нет пяти источников прерываний: FLASH, L2_ERROR, USB0, USB1, PWMx_TZ	РП, табл. 13.1, с. 187–193 `K1921VG1T.h`	4 добавлены (кроме L2_ERROR)
12	Поля статуса HASH.SR помечены «только запись»	РП, с. 787	описано
13	Таймеры TMR4–TMR15 описаны с четырьмя блоками CAPCOM вместо одного	РП, с. 220	описано
14	Значения сброса HASH.SR и MSC.U_ITEM расходятся с РП; для HASH.SR чтение на плате подтвердило значение из SVD (`0x00000001`)	РП, с. 787 и 699	описано
15	В поле выбора алгоритма блока CRYPTO нет значения «Магма»	РП, с. 324 и 337; `K1921VG1T.h`	исправлено

Кроме того, шапка SVD-описания совпадает с шапкой учебного примера `ARM_Example.svd` из пакета CMSIS компании ARM: в ней остались серия `K1921VKx` (серия НИИЭТ на ядре ARM), ядро `CM4` и отказ от ответственности от имени ARM.

6.1.2. Программные компоненты SDK

Настройка PLL. В функции настройки тактирования из файла `system_k1921vg1t.c` блок для кварца 27 МГц снабжён комментарием «FOUT = 204 000 000 Hz», то есть частота на выходе PLL (FOUT) в нём указана равной 204 МГц. Но заданные в этом блоке множитель и делители по формуле из того же файла дают $27\ \text{МГц} \cdot 100/(3 \cdot 2 \cdot 3) = 150$ МГц, где 27 МГц — частота кварца, 100 — множитель, 3, 2 и 3 — делители; измеренная в этой работе частота после переключения на PLL0 — тоже 150 МГц (подраздел 4.5). Комментарий расходится с кодом: тот, кто ему поверит, будет рассчитывать делители периферии от 204 МГц вместо 150 МГц.

В той же функции число дополнительных тактов ожидания при чтении Flash-памяти задано равным 2 (поле `LAT` регистра `FLASHM.CTRL`). По таблице 9.1 РП (с. 76) для 150 МГц (диапазон 140–175 МГц) их нужно 4, а для указанных в комментарии 204 МГц — 5, поэтому 2 тактов недостаточно ни для фактической, ни для заявленной частоты.

Конфигурация OpenOCD. При подключении OpenOCD читает из микросхемы идентификатор JTAG и сверяет его с указанным в конфигурации, чтобы убедиться, что подключена именно та микросхема, для которой написана конфигурация. При несовпадении OpenOCD сообщает, что цепочка JTAG не такая, как ожидалось. Микроконтроллер возвращает идентификатор `0xDEB15001`, а в конфигурации для одного ядра (файл `k1921vg1t_singlecore.cfg`) указан `0xDEB05001`; частота TCK задана 2000 кГц, тогда как документ производителя «Быстрый старт…» требует не более 200–250 кГц для К1921ВГ1Т. Двухъядерная конфигурация `k1921vg1t.cfg` в условиях работы не заработала: OpenOCD сообщал «Hart 0 unavailable». Для работы сделана исправленная одноядерная конфигурация.

Пакет поддержки платы (BSP). Функция `BSP_LED_On` для платы `NIIET-DEV-K1921VG1T` записывает единицы в регистр `DATAOUTCLR`, то есть выводит на линию ноль, исходя из того, что светодиод горит при нуле. На плате светодиоды LED1–LED4 горят при единице: это установлено двумя независимыми опытами с прямой записью в регистр `DATAOUT` (значение `0x5000` зажигает LED1 и LED3, значение `0xE000` — LED2, LED3 и LED4) и подтверждено примером `binary_counter`: после пяти нажатий кнопки (двоичное 0101) горят LED1 и LED3 (рисунок 3). Поэтому `BSP_LED_On` на этой плате гасит светодиод. В примере `run_leds` ошибка не проявляется: он использует только переключение состояния (`BSP_LED_Toggle`). Функция инициализации кнопки `BSP_Btn_Init` в этом BSP пуста и помечена `// todo`.

Заголовочный файл. В заголовочном файле `K1921VG1T.h` у каждого значения поля есть комментарий-описание, которое среда разработки показывает

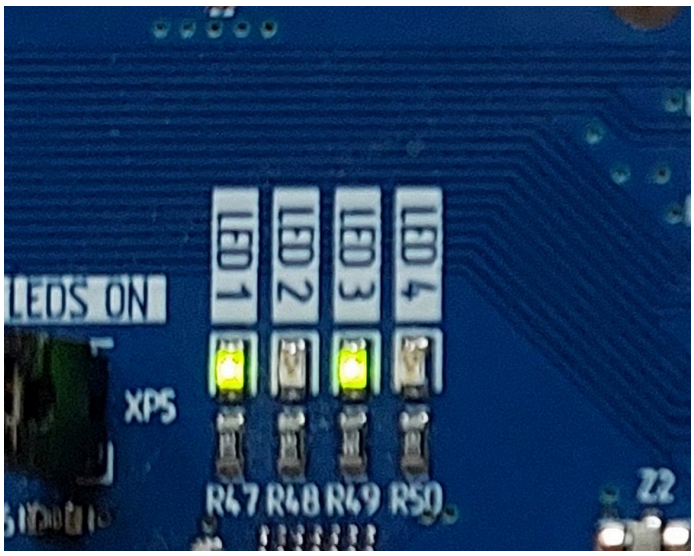


Рисунок 3 — Пример `binary_counter` после пяти нажатий кнопки: горят LED1 и LED3

как подсказку. У значения «Магма» (2) поля выбора алгоритма блока CRYPTO (`CRYPTO_CONTROL_ALGORITHM_Magma`) стоит описание «GCM payload phase». Оно совпадает с описанием значения 2 соседнего поля `GCM_PHASE`, которое задаёт этап режима шифрования GCM, и к выбору алгоритма отношения не имеет: по подсказке нельзя понять, что значение 2 выбирает «Магму».

6.1.3. Руководство пользователя

В таблице полей регистра `HASH.SR` (с. 787) биты 1 и 0 названы `DCIE` и `DINIE`. Это имена битов соседнего регистра разрешения прерываний `IMR` (с. 786). По описанию в той же таблице биты `SR` — флаги состояния (вычисление завершено, входной буфер готов), а в SVD-описании и заголовке SDK они называются `DCIS` и `DINIS`. Поиск бита по имени в РП приводит к описанию другого регистра.

Сразу после включения микроконтроллер работает от встроенного RC-генератора HSI; от его частоты зависят задержки и скорость периферии, пока программа не переключится на кварц или PLL. Для этой частоты РП приводит два разных значения: 4 МГц (например, в описании системы тактирования, с. 56) и номинал диапазона подстройки 3,375 МГц с нестабильностью ±4 % в таблице характеристик генератора (таблица 42.7, с. 513). Измерение на плате дало около 3,34 МГц (подраздел 4.4): это соответствует таблице, но на 16,6 % ниже значения 4 МГц.

6.2. Собственные ошибки

Ниже приведены ошибки, допущенные в работе и исправленные: их симптомы и причины полезны тем, кто будет работать с этим микроконтроллером.

CC BY 4.0

Таблица 10 — Собственные ошибки и их исправление

Ошибка	Как проявилась	Исправление
В Makefile C-примеров частота кварца `HSECLK_VAL` задана 16 МГц вместо 27 МГц	неверная скорость UART при тактировании от кварца	значение 27 МГц
Не определён макрос `RETARGET` (перенаправление вывода `printf` в UART)	вывод `printf` не попадал в UART	макрос добавлен в Makefile
При загрузке ключа и данных в блок CRYPTO переставлялся порядок байт	шифрование и обратное расшифрование сходились, но шифртекст не совпадал с тестовым вектором ГОСТ Р 34.12-2015	данные загружаются без перестановки; шифртекст совпал с эталоном
Для регистра `PLL[0].CFG` использовался метод `write()`	PLL не запускался	метод `write_with_zero()` (подраздел 4.5)
Перед замером частоты через UART не дожидались опустошения буфера передатчика	измеренные частоты завышены на 0,6–1,7 %	ожидание пустого буфера перед замером (подраздел 4.4)

6.3. Исправления производителя в более поздних версиях

Часть описанных дефектов исправлена производителем в последующих версиях SDK.

В `k1921vg1t_sdk` (проверено на коммите `44510f7ad5e8`) блок настройки PLL для кварца 27 МГц подписан «150 000 000 Hz» и задано 5 тактов ожидания Flash-памяти, а в одноядерной конфигурации OpenOCD указан идентификатор `0xDEB15001`. SVD-описание в этом SDK после коммита `9fa40f85be5b` не менялось.

В новом SDK производителя `riscv_cmake_sdk` (проверено на коммите `ae5a8e69dfe0`) в SVD-описании нет дефектов классов 1–5, размер блока WDT исправлен, регистр микросхемы MCP4921 удалён, добавлены прерывания USB0, USB1 и PWMx_TZ и значение «Магма». Сохраняются классы 6–8, размер блока TMR0, отсутствие прерываний FLASH и L2_ERROR, а также расхождения 12–14; svd2rust этот файл не принимает — генерация останавливается на ошибке класса 6. Светодиоды платы в BSP этого SDK по-прежнему включаются записью в `DATAOUTCLR`.

7. Учебно-методический практикум

Практикум построен на примерах репозитория и ведёт от программы без единой библиотеки к работе с периферией через PAC. Он рассчитан на читателей, которые знают C, знакомы с основами Rust (владение, типы, модули) и в общих чертах представляют устройство микроконтроллера. Каждая работа опирается на предыдущую. Ожидаемые результаты на плате приведены в таблице 4, подробности о каждом блоке — в разделе 4.

Задания и контрольные вопросы носят примерный характер: их можно менять и дополнять по своему усмотрению. Решения заданий в репозиторий не включены.

7.1. Подготовка рабочего места

Нужны отладочная плата К1921ВГ1Т, JTAG-адаптер FT2232HL Board, кабель USB-C для вывода через UART и компьютер с инструментами из таблицы 1 (для практикума на Rust достаточно Rust и OpenOCD НИИЭТ; Task удобен, но не обязателен). Порядок подготовки:

1. Клонировать репозиторий и установить инструменты по разделам «Программы» и «Быстрый старт» файла `README.md`.
2. Соединить адаптер с разъёмом XP7 платы по таблице из приложения Б, проверить, что установлена перемычка XP5 «LEDS ON» (без неё светодиоды LED1–LED4 отключены от выводов C12–C15), подать питание 12 В на разъём XS5.
3. Проверить связь с микроконтроллером командой `task scan` из корня репозитория. В выводе OpenOCD должны быть идентификатор `0xdeb15001` и строка `hart 0: XLEN=32, misa=0x4014912d`.
4. Собрать и прошить пример командой `task run P=<имя примера>`, затем нажать кнопку RESET (SB2): управления сбросом в конфигурации OpenOCD нет.
5. Для Rust-примеров с выводом текста открыть терминал на COM-порту моста CH340B: 9600 бит/с, 8 бит данных, без контроля чётности, 1 стоповый бит.

Если OpenOCD не находит адаптер, рекомендации — в разделе «Особенности» файла `README.md`.

В каждой работе нужно прочитать код примера, собрать и запустить его, сверить результат с ожидаемым, выполнить задания и ответить на контрольные вопросы. Изменённый код нужно собирать, прошивать и проверять на плате.

7.2. Работа 1. Программа без библиотек

Пример: `rust_blink`. **Цель:** понять, что нужно, чтобы программа на Rust вообще запустилась на микроконтроллере.

Изучаются атрибуты `#![no_std]` и `#![no_main]`, стартовый код на ассемблере в `global_asm!`, собственный скрипт компоновки `link.x` и файл `.cargo/`

`config.toml`, который задаёт цель сборки и передаёт компоновщику этот скрипт. К регистрам программа обращается по адресам функцией `write_volatile`. Ожидаемый результат: мигает LED1.

Задания:

1. Мигать светодиодом LED2 (вывод C13) вместо LED1.
2. Сделать «бегущий огонь» по светодиодам LED1–LED4.
3. Изменить число итераций цикла задержки, измерить период мигания секундомером и оценить, сколько тактов занимает одна итерация (частота ядра после сброса — около 3,34 МГц).

Контрольные вопросы:

1. Почему секция `.init` должна оказаться по адресу `0x00000000`?
2. Почему для записи в регистр используется `write_volatile`, а не обычное присваивание через указатель?
3. В `link.x` нет секции `.data`, а стартовый код не копирует данные из Flash-памяти в ОЗУ. Что пришлось бы добавить, если в программе появится глобальная переменная с начальным значением?

7.3. Работа 2. Среда выполнения riscv-rt

Пример: `rust_blink_rt`. **Цель:** перейти от своего стартового кода к стандартной среде выполнения.

Программа делает то же, что в работе 1, но стартовый код, скрипт компоновки `link.x` и обработчики исключений даёт крейт `riscv-rt`. От проекта требуется только описание памяти `memory.x`, которое скрипт сборки `build.rs` кладёт туда, где его найдёт компоновщик. Точка входа отмечается атрибутом `#[entry]`.

Задания:

1. Сравнить размеры секций прошивок `rust_blink` и `rust_blink_rt` утилитой `riscv-none-elf-size` с ключом `-A` (входит в xPack riscv-none-elf-gcc) и объяснить разницу (подраздел 5.2).
2. Найти в исходном коде `riscv-rt` (файл `src/asm.rs`) место, где копируется секция `.data` и обнуляется секция `.bss`.

Контрольные вопросы:

1. Что изменится, если удалить `build.rs`?
2. Почему в `memory.x` указано меньше Flash-памяти и ОЗУ, чем есть в микроконтроллере, и чем это ограничивает программы?

7.4. Работа 3. Регистры по именам: PAC

Пример: `rust_blink_pac`. **Цель:** научиться работать с периферией через PAC.

Изучаются получение периферии `Peripherals::steal()`, методы `read()`, `write()` и `modify()` и то, как в PAC выглядят регистры и поля (раздел 3).

Задания:

1. Включать светодиод записью в `DATAOUTSET`, а выключать записью в `DATAOUTCLR`. Объяснить, почему одна из этих записей требует `unsafe`, а другая нет.
2. Заменить цикл задержки безопасной функцией `riscv::asm::delay()`. Для этого в файл `Cargo.toml` нужно добавить зависимость от крейта `riscv` версии 0.16. Сколько блоков `unsafe` осталось в программе?
3. Внести в программу ошибки из примера в подразделе 5.4 и разобрать сообщения компилятора.

Контрольные вопросы:

1. Чем отличаются `write()` и `modify()`? Почему для регистров RCU используется `modify()`?
2. Чем `steal()` отличается от `take()` и почему `steal()` помечен как `unsafe`?

7.5. Работа 4. Прерывание от таймера

Пример: `rust_timer_irq`. **Цель:** настроить прерывание от периферийного блока.

Изучается путь запроса: таймер TMR0 → контроллер PLIC → внешнее прерывание машинного режима ядра, разрешение прерываний в регистрах `mie` и `mstatus`, обработчик с атрибутом `#[riscv_rt::core_interrupt]` и протокол PLIC «захват — завершение».

Задания:

1. Рассчитать значение периода (регистр `TMR0.PERIOD`), при котором LED1 переключается раз в секунду, по измеренной частоте HSI (подраздел 4.4), и проверить секундомером.
2. Переключать LED2 на каждом втором прерывании. Счётчик прерываний хранить в статической переменной типа `AtomicU32`.

Контрольные вопросы:

1. Что произойдёт, если в обработчике не сбросить флаг прерывания таймера?
2. Почему к регистрам PLIC пример обращается по адресам, а не через PAC?

7.6. Работа 5. UART и измерение частоты

Пример: `rust_uart`. **Цель:** настроить UART и использовать его как эталон времени.

Изучаются выбор источника тактирования UART, расчёт делителя скорости и способ измерения частоты ядра по времени передачи через UART (подраздел 4.4).

Задания:

1. По формуле 29.2 РП (с. 352) рассчитать значения `IBRD` и `FBRD` для 115 200 бит/с при частоте 27 МГц, найти ошибку установки скорости, перенастроить пример и проверить вывод в терминале на этой скорости.
2. Убрать из функции замера ожидание опустошения буфера передатчика и сравнить результат с исходным. Объяснить разницу.

Контрольные вопросы:

1. Почему UART тактируется от кварца HSE, а не от HSI?
2. Почему регистр `LCRH` нужно записывать после `IBRD` и `FBRD`?

7.7. Работа 6. АЦП и датчик температуры

Пример: `rust_adc`. **Цель:** выполнить аналоговое измерение.

Изучаются включение аналоговой части АЦП, секвенсор, буфер результатов и пересчёт кода в температуру по формуле из РП.

Задания:

1. Выводить вместе с температурой код АЦП.
2. Усреднять 16 измерений и сравнить разброс показаний до и после усреднения.

Контрольные вопросы:

1. Зачем для канала датчика температуры задаётся задержка `CHDELAY`?
2. Какие допущения сделаны при пересчёте кода АЦП в температуру?

7.8. Работа 7. ЦАП

Пример: `rust_dac`. **Цель:** сформировать аналоговый сигнал.

Изучаются включение ЦАП, внутренний источник опорного напряжения и программный запуск преобразования.

Задания:

1. Вместо синусоиды выводить треугольный сигнал.
2. Измерить мультиметром напряжение при кодах 0, 1024, 2048 и 4095 и построить зависимость напряжения от кода.

Контрольный вопрос: какое напряжение ожидается при коде 4095 и чем может объясняться расхождение с измеренным (подраздел 4.7)?

7.9. Работа 8. Аппаратное шифрование

Пример: `rust_crypto`. **Цель:** использовать блок CRYPTO и проверить результат по стандарту.

Задания:

1. Изменить один бит открытого текста и сравнить, сколько битов шифртекста изменилось.

2. Исследовательское задание: зашифровать блок алгоритмом AES-128 (значение поля `ALGORITHM` — 0), проверить результат по тестовому вектору из стандарта FIPS-197 [13] и определить, в каком порядке нужно загружать в блок ключ и данные.

Контрольный вопрос: почему совпадения расшифрованного текста с исходным недостаточно для проверки правильности шифрования?

7.10. Работа 9. Переключение на PLL

Пример: `rust_pll`. **Цель:** перевести микроконтроллер на высокую частоту.

Изучаются порядок настройки PLL, такты ожидания Flash-памяти и разница между методами `write()` и `write_with_zero()` (подраздел 4.5).

Задания:

1. По формуле из подраздела 4.5 рассчитать частоту для другого значения `NDIV` и по таблице 9.1 РП определить нужное число тактов ожидания Flash-памяти. Изменённую настройку прошивать только после сверки параметров PLL с ограничениями РП: при неверных настройках PLL может не захватить частоту, и программа зависнет в ожидании.
2. Сравнить время выполнения одного и того же цикла до и после переключения на PLL.

Контрольный вопрос: почему с методом `write()` PLL не запускается?

7.11. Итоговое задание

Написать на Rust через PAC двоичный счётчик нажатий кнопки USER BTN (вывод B3, при нажатии на выводе ноль), который показывает число от 0 до 15 на светодиодах LED1–LED4. Это аналог С-примера `binary_counter`. Сравнить две реализации по размеру прошивки, числу строк кода и числу блоков `unsafe`.

8. Заключение

8.1. Основные результаты

1. В SVD-описании К1921ВГ1Т из SDK производителя исправлены восемь классов дефектов, мешающих получить PAC (всего 48 мест), и из него сгенерирован PAC на Rust. Генерация воспроизводима: по закреплённым версиям инструментов PAC получается из исправленного файла побайтно, что проверяется автоматически на Linux и Windows (подраздел 3.4).
2. Написаны девять примеров на Rust (no_std), от программы без единой библиотеки до прерываний, UART, АЦП, ЦАП, аппаратного шифрования и переключения на PLL. Все они проверены на плате; шифрование «Кузнечиком» совпало с тестовым вектором ГОСТ Р 34.12-2015, частота PLL — 150 МГц с точностью кварца (раздел 4).
3. Количественно оценена применимость Rust: размер прошивок того же порядка, что у C; блоки `unsafe` в основном вызваны неполнотой SVD-описания, а не языком; PAC позволяет задавать значения полей по именам и выделяет запись произвольного числа в поле без перечисления блоком `unsafe` (раздел 5). Вывод: Rust (no_std) применим для bare-metal разработки под К1921ВГ1Т.
4. Выявлены и описаны дефекты SVD-описания, руководства пользователя и SDK с проверкой по первоисточникам и, где возможно, на плате; перечень не исчерпывающий (раздел 6).
5. Подготовлена открытая документация: репозиторий с инструкцией от подключения платы до мигания светодиода, описанием инструментов и их версий, и практикум из девяти работ с заданиями (раздел 7).

8.2. Значимость

Разработчик, который хочет писать для К1921ВГ1Т на Rust, получает готовый PAC, рабочие примеры для основных блоков и описание подводных камней: от порядка байт при загрузке ключа в блок CRYPTO до тактов ожидания Flash-памяти при переходе на PLL. Перечень дефектов полезен и тем, кто пишет на C, и производителю: часть описанных дефектов производитель исправил в более поздних версиях SDK (подраздел 6.3), остальные описаны с указанием страниц руководства и способа проверки. Практикум позволяет шаг за шагом самостоятельно освоить работу с микроконтроллером на Rust; все материалы открыты.

8.3. Ограничения

- Все проверки выполнены на одной плате с одним экземпляром микроконтроллера. Напряжения измерялись мультиметром, температура — без эталонного термометра.
- Используется одно ядро из двух; двухъядерная конфигурация OpenOCD из SDK в условиях работы не заработала.

- На плате PAC проверен только в тех блоках, которые используют примеры; полная сверка всех регистров с руководством пользователя не проводилась. Расхождения 10, 12–14 и прерывание L2_ERROR из расхождения 11 описаны, но в SVD-описании не исправлены.
- Контроллер прерываний PLIC в PAC не входит, библиотеки уровня HAL нет.
- Прошивка выполнялась только в Windows. Сборка PAC и примеров на Rust проверяется на Linux и Windows; примеры на C собирались только в Windows и только с SDK `k1921vg1t_sdk` на коммите `9fa40f85be5b`.

8.4. Возможная дальнейшая работа

- Описать в SVD-описании контроллер PLIC, допустимые значения числовых полей (`writeConstraint`) и единообразные поля регистров GPIO; это уберёт большую часть блоков `unsafe`.
- Включить в PAC реализацию критической секции, чтобы получать периферию безопасным методом `take()`.
- Написать HAL с реализацией интерфейсов `embedded-hal`, чтобы для К1921ВГ1Т можно было использовать готовые драйверы устройств из экосистемы Rust.
- Освоить второе ядро и работу с плавающей точкой.
- Передать перечень дефектов производителю.

8.5. Лицензия

Все результаты работы — исходный код, исправленное SVD-описание, PAC, документация и эта записка — распространяются на условиях лицензии Creative Commons «С указанием авторства» версии 4.0 Международная (CC BY 4.0) [1]; лицензии сторонних файлов указаны в файле `NOTICE.md` репозитория.

Список источников

1. Creative Commons. «С указанием авторства» версии 4.0 Международная (CC BY 4.0) — юридический текст [Электронный ресурс]. URL: https://creativecommons.org/licenses/by/4.0/legalcode.ru (дата обращения: 10.10.2026).
2. АО «НИИЭТ». niiet/k1921vg1t_sdk: Набор средств разработки [Электронный ресурс]. URL: https://gitflic.ru/project/niiet/k1921vg1t_sdk (дата обращения: 10.10.2026).
3. АО «НИИЭТ». niiet/riscv_cmake_sdk [Электронный ресурс]. URL: https://gitflic.ru/project/niiet/riscv_cmake_sdk (дата обращения: 10.10.2026).
4. Platform Support — The rustc book [Электронный ресурс]. URL: https://doc.rust-lang.org/rustc/platform-support.html (дата обращения: 10.10.2026).
5. Open-CMSIS-SVD: System View Description [Электронный ресурс]. URL: https://open-cmsis-pack.github.io/svd-spec/main/index.html (дата обращения: 10.10.2026).
6. crates.io: Rust Package Registry [Электронный ресурс]. URL: https://crates.io/ (дата обращения: 10.10.2026).
7. АО «НИИЭТ». Микросхемы интегральные К1921ВГ1Т, К1921ВГ10Я. Руководство пользователя от 10.06.2026 [Электронный ресурс]. URL: https://gitflic.ru/project/niiet/niiet (дата обращения: 10.10.2026).
8. АО «НИИЭТ». Быстрый старт с микроконтроллерами К1921ВГ015, К1921ВГ1Т, К1921ВГ3Т, К1921ВГ5Т, К1921ВГ7Т (версия от 17.04.2026) [Электронный ресурс]. URL: https://gitflic.ru/project/niiet/niiet (дата обращения: 10.10.2026).
9. АО «НИИЭТ». niiet/openocd [Электронный ресурс]. URL: https://gitflic.ru/project/niiet/openocd (дата обращения: 10.10.2026).
10. rust-embedded/svd2rust: Generate Rust register maps (structs) from SVD files [Электронный ресурс]. URL: https://github.com/rust-embedded/svd2rust (дата обращения: 10.10.2026).
11. ГОСТ Р 34.12-2015. Информационная технология. Криптографическая защита информации. Блочные шифры. Стандартинформ, 2015.
12. riscv-rt: Minimal runtime / startup for RISC-V CPU's [Электронный ресурс]. URL: https://crates.io/crates/riscv-rt (дата обращения: 10.10.2026).
13. NIST. Advanced Encryption Standard (AES). FIPS PUB 197 [Электронный ресурс]. URL: https://doi.org/10.6028/NIST.FIPS.197-upd1 (дата обращения: 10.10.2026).

Приложение А. Места исправлений в SVD-описании

В приложении перечислены все 48 мест SVD-описания К1921ВГ1Т (SDK `k1921vg1t_sdk`, коммит `9fa40f85be5b`), исправленных, чтобы из него можно было сгенерировать PAC. Номера классов соответствуют таблице 2; там же указано, что происходит без исправления. Запись `[%s]` означает массив регистров или кластеров. Построчный перечень правок приведён в файле `projects/k1921vg1t-pac/K1921VG1T.svd.diff`, обоснование каждой — в `projects/k1921vg1t-pac/SVD_FIXES.md`.

Таблица 11 — Места исправлений в SVD-описании (48 мест)

Класс	Где	Было → стало	Основание
1	блок `<cpu>`	`Endian.LITTLE` → `little`	список допустимых значений в спецификации CMSIS-SVD
2	блок `<cpu>`	пустой `<nvicPrioBits />` → `4`	тег обязателен по схеме CMSIS-SVD; значение условное, в К1921ВГ1Т нет контроллера NVIC
3	единственный регистр в кластерах-массивах (30 мест): `GPIOA.MASKLB`, `GPIOA.MASKHB`, `CANFD0.MESSAGE`, `CAN.LIST`, `CAN.MSPND`, `CAN.MSID`, `ADC.CHCTL`, `ADC.CHDELAY`, `HASH.HR`, `CRYPTO.IV`, `CRYPTO.TEXT_IN`, `CRYPTO.KEY`, `CRYPTO.TEXT_OUT`, `FLASHM.DATA`, `FLASHM.ERRMASK`, `SIU.TMRMUX`, `RCU.UARTCFG`, `RCU.SPICFG`, `RCU.I2SCFG`, `RCU.USBCFG`, `RTC.GPR`, `TRNG.KEY`, `TRNG.FIFO`, `EMC.WINCFG`, `FLASHD.DATA`, `FLASHD.ERRMASK`, `LIN0.DTX`, `LIN0.DRX`, `I2C0.EADDR`, `MSC.U_ITEM` (у всех — `[%s]`)	пустое имя → `VALUE`	имя регистра обязательно; имя `VALUE` выбрано в работе
4	`CAN.Node[%s].Reserved`, `DMA.CH[%s].Reserved`, `DMA.Reserved`, `SIU.Reserved`, `SRAM.Reserved`, `SDRAM.Reserved`, `DAC0.Reserved`, `TMR0.CAPCOM[%s].Reserved`	ширина 0 → 32 бита	ширина регистров по умолчанию в этом файле; у `DAC0.Reserved` есть поля до бита 16
5	`ADC.ACTL[%s].<>TACC0`, `ADC.ACTL[%s].<>Reserved`	префикс `<>` удалён	допустимые символы имени по схеме CMSIS-SVD; соседний регистр называется `TACC1`
6	`ADC.SEQ[%s].SCVAL`, `HASH.SR`, `SIU.NODESTEST`, `MSC.U_ITEM[%s]`	`write-only` → `read-write`	в регистре есть и читаемые, и записываемые поля; РП, с. 744, 787, 581, 699
7	`TMR0.CAPCOM[%s]`	шаг 12 → 16 байт	элемент — четыре 32-битных регистра; РП, с. 673
8	регистр `ACTL` в кластере `ADC.ACTL[%s]`	`ACTL` → `CTRL`	совпадение имён кластера и регистра; схема имён svd2rust

Кроме этих правок, в исправленном файле устранены расхождения 9, 11 (частично) и 15 из таблицы 9; они разобраны в подразделе 6.1.1.

Приложение Б. Подключение JTAG-адаптера

Адаптер FT2232HL Board V3.6 подключается к разъёму XP7 «JTAG» отладочной платы проводами напрямую, без перекрещивания линий TDI и TDO. Номера контактов разъёма CN4 адаптера взяты по его принципиальной схеме, номера контактов XP7 — по маркировке платы.

Таблица 12 — Соединение адаптера FT2232HL Board с разъёмом XP7

Сигнал	Адаптер, CN4	Плата, XP7
TMS	7	2
TCK	9	4
TDO	13	6
TDI	5	8
nRST	15	10
GND	GND	3, 5 или 9

Контакт 1 разъёма XP7 (+3,3 В) к адаптеру не подключается. Провод nRST можно не подключать: в конфигурации OpenOCD управление сбросом отключено (`reset_config none`), и плата сбрасывается кнопкой RESET (SB2). Линия TRST на разъёме XP7 не выведена. В Windows канал B адаптера (Interface 1), на который разведён JTAG, работает с драйвером WinUSB, установленным программой Zadig; канал A (Interface 0) остаётся с драйвером FTDI.

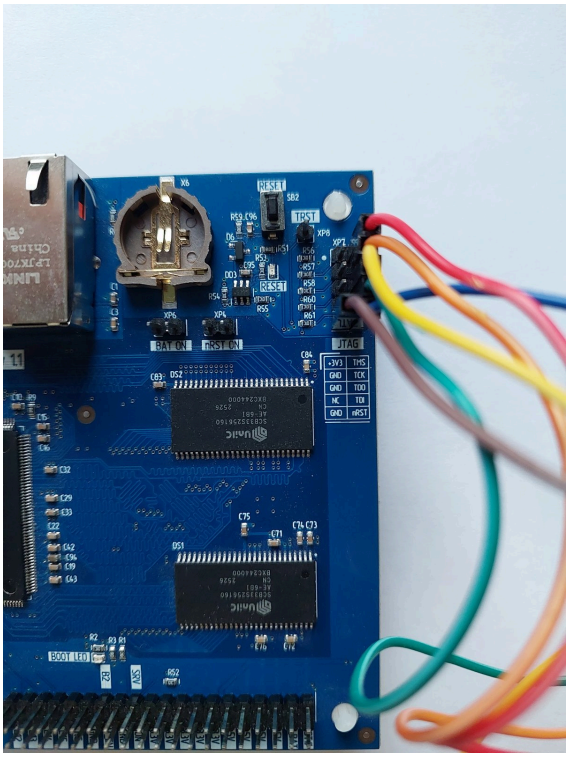


Рисунок 4 — Подключение адаптера к разъёму XP7 «JTAG»; справа под разъёмом — маркировка его контактов на плате