**И.И. Романова**

# К1921ВГ015: От мигающего светодиода до промышленного узла управления

*Практикум по встраиваемым системам на базе RISC-V для студентов и инженеров*

**Москва, 2026**

**УДК: 004.4**
**ББК: 32.973.26-018.2**

**Автор:**
Романова И. И. – старший преподаватель Московского института электроники и математики им. А. Н. Тихонова Национального исследовательского университета «Высшая школа экономики».

И.И. Романова,
К1921ВГ015: От мигающего светодиода до промышленного узла управления. М.: ДМК Пресс, 2026. – 263 с.: ил.

Перед вами практикум по встраиваемым системам на базе отечественного RISC-V-микроконтроллера К1921ВГ015. Пособие последовательно проводит читателя от самых простых задач управления вводом-выводом до построения промышленного узла управления, охватывая полный цикл практической разработки.
Материал разделен на четыре части. Первая знакомит с GPIO, таймерами, внешними прерываниями и контроллером NVIC. Вторая посвящена аналоговой периферии: широтно-импульсной модуляции, аналого-цифровому преобразованию и генератору истинно случайных чисел. Третья часть рассматривает коммуникации и хранение данных — UART, DMA, энергонезависимую память и часы реального времени. Четвертая часть посвящена профессиональным вопросам: управлению энергопотреблением, режимам сна, сторожевому таймеру и обработке ошибок.
Каждая глава сопровождается практической работой, закрепляющей полученные навыки на реальном микроконтроллере. Всего в пособии двенадцать практических работ, охватывающих ключевые подсистемы микроконтроллера.
Издание предназначено для студентов технических специальностей, инженеров и разработчиков, желающих освоить проектирование встраиваемых систем на отечественной аппаратной платформе.

**© И.И. Романова, 2026**
**Распространяется по лицензии CC BY 4.0**

СОДЕРЖАНИЕ

**Введение**

**Часть I. Фундамент: Управление вводом-выводом и временем**

Глава 1. Цифровой мир МК: GPIO и основы таймеров

Практическая работа 1

Глава 2. Реакция на события: Внешние прерывания и NVIC

Практическая работа 2

**Часть II. Преобразование данных: Аналоговая периферия и генераторы**

Глава 3. Генерация аналоговых сигналов: ШИМ и силовая электроника

Практическая работа 3

Глава 4. Измерение физического мира: АЦП и датчики

Практическая работа 4

Глава 5. Истинная случайность: TRNG

Практическая работа 5

**Часть III. Коммуникации и хранение: Диалог с внешним миром**

Глава 6. Текстовый интерфейс: UART и DMA

Практическая работа 6

Практическая работа 7

Глава 7. Сохранение состояния: Non-volatile память

Практическая работа 8

Глава 8. Аппаратные часы: RTC

Практическая работа 9

Практическая работа 10

**Часть IV. Энергетика и надежность: Профессиональная разработка**

Глава 9. Экономика энергии: PMU и режимы сна

Практическая работа 11

Глава 10. Отказоустойчивость: Watchdog и обработка ошибок

Практическая работа 12

Заключение

Список использованных источников

Приложение

# Введение

За последние несколько лет российская микроэлектроника сделала огромный шаг вперед, перейдя от штучных разработок к серийному выпуску современных чипов. Главным трендом стал массовый переход на открытую процессорную архитектуру RISC-V, что позволяет создавать полностью независимые решения.

Разработанный Воронежским НИИЭТ микроконтроллер К1921ВГ015 представляет собой энергоэффективное решение, которое отлично подходит для автономных устройств. Базируясь на 32-разрядном ядре RISC-V, контроллер работает на частоте 50 МГц (в некоторых режимах до 80 МГц) и, в отличие от «Амура», обладает встроенным модулем вычислений с плавающей запятой (FPU). Внутри кристалла размещен 1 Мбайт встроенной Flash-памяти и 320 Кбайт оперативной памяти (256 КБ основное ОЗУ + 64 КБ батарейный домен). Отладочные платы с этим чипом позволяют инженерам работать с точной аналоговой периферией, куда входят быстрые SAR-АЦП и точные 16-разрядные сигма-дельта АЦП. Благодаря режимам сна и батарейному питанию таймеров реального времени, этот чип идеален для портативной медицинской техники, беспроводной телеметрии и смарт-счетчиков.

Микроконтроллер К1921ВГ015 – это 32-разрядная система на кристалле (СнК), разработанная АО «НИИ электронной техники» (г. Воронеж). На сегодняшний день данный чип является одним из ключевых компонентов в стратегии импортозамещения отечественной электроники, предлагая инженерам полноценную альтернативу широко распространенным зарубежным решениям (в частности, семействам STM32) с фокусом на задачи реального времени.

К1921ВГ015 представляет собой специализированный инструмент. Он уступает флагманским западным аналогам в максимальной тактовой частоте, но значительно превосходит их в прикладной готовности к задачам силовой электроники и суровым условиям российских производств. Изучение этого микроконтроллера дает студенту или инженеру компетенции, которые напрямую конвертируются в востребованные навыки разработки отечественного промышленного оборудования.

Настоящее учебно-методическое пособие построено как практикум, который последовательно проводит читателя от простейших задач управления вводом-выводом до построения законченного промышленного узла управления. Материал разделен на четыре части. Первая часть закладывает фундамент: работа с GPIO, таймерами, внешними прерываниями и контроллером NVIC. Вторая часть посвящена преобразованию данных и

аналоговой периферии: широтно-импульсной модуляции, аналого-цифровому преобразованию и генератору истинно случайных чисел. Третья часть рассматривает коммуникации и хранение состояния: UART, DMA, энергонезависимую память и часы реального времени. Четвертая часть посвящена профессиональным вопросам энергетики и надежности: управлению энергопотреблением, режимам сна, сторожевому таймеру и обработке ошибок.

Каждая глава сопровождается практической работой, закрепляющей полученные навыки на реальном микроконтроллере. Всего в пособии двенадцать практических работ, охватывающих ключевые подсистемы К1921ВГ015. Примеры построены так, чтобы читатель не просто копировал готовый код, а понимал, как устроена периферия и почему принимаются те или иные инженерные решения.

Пособие предназначено для студентов технических специальностей, инженеров и разработчиков, желающих освоить проектирование встраиваемых систем на отечественной аппаратной платформе. Для работы с практикумом достаточно базовых знаний языка С и представления о том, как устроен микроконтроллер. Все примеры проверены на реальном оборудовании и могут быть воспроизведены в учебной лаборатории или на рабочем месте инженера.

# Часть I. Фундамент: Управление вводом-выводом и временем

ОТЕЧЕСТВЕННЫЕ МИКРОКОНТРОЛЛЕРЫ



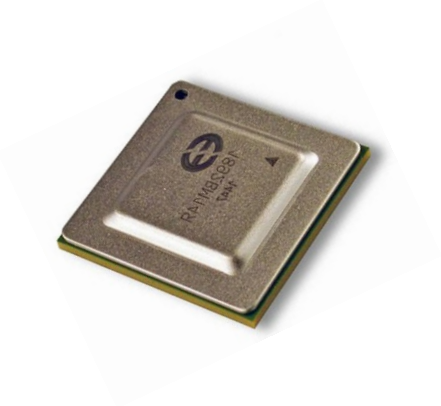
НИИЭТ — К1921ВГ015



Микрон — MIC32_V2



МЦСТ Эльбрус — К1879ВМ1



НПЦ «ЭЛВИС» — 1892ВМ



АО «ПКК Миландр» — 1986ВЕ



АО «НТЦ Модуль» — К1879ВМ8Я

# Глава 1. Цифровой мир МК: GPIO и основы таймеров

**Общие сведения о микроконтроллере К1921ВГ015**

К1921ВГ015 – это отечественный 32-битный микроконтроллер на архитектуре RISC-V. Он используется в учебных, промышленных и исследовательских проектах благодаря открытой архитектуре ядра и наличию базовых встроенных периферийных модулей.

В его состав входят:

1) порты GPIO;
2) 32-битный таймер TMR32;
3) контроллер прерываний PLIC;
4) системный таймер;
5) внутренняя SRAM и Flash;
6) отладочный интерфейс через FT2232;
7) UART, SPI, I²C, CAN (в рамках этой лабораторной работы не используются).

Отладочная плата построена вокруг этого микроконтроллера и содержит все необходимые элементы для разработки: питание, USB-интерфейс, кнопки, светодиоды, контакты GPIO, схемы сброса, интерфейс программирования.

Плата состоит из функциональных блоков:

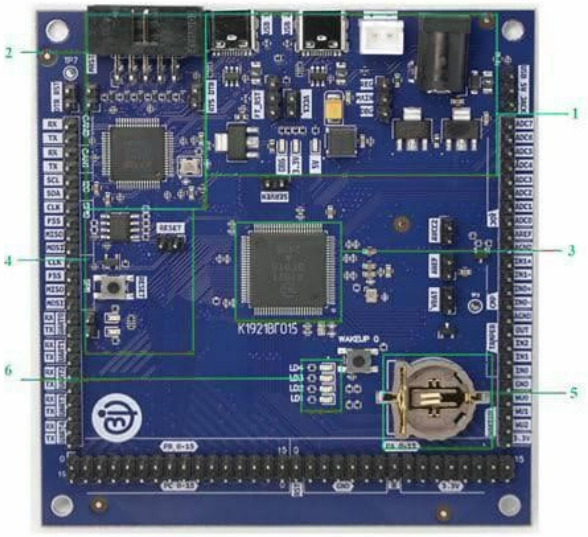



Рисунок 1.1. Отладочная плата К1921ВГ015

1 – блок источников питания;

2 – блок программатора;

3 – микросхема 1921ВГ015;

4 – сервисный блок;

5 – батарейный отсек CR1220;

6 – светодиодная индикация.

## 2. Архитектура RISC-V

К1921ВГ015 построен на 32-битной архитектуре **RISC-V**. Это современная открытая архитектура процессорных ядер, которая широко используется в учебных и прикладных проектах.

Ключевые особенности RISC-V (в контексте данной лабораторной работы):

- **Простая и регулярная система команд**, удобная для обучения и отладки;
- **Высокая предсказуемость выполнения**: многие операции выполняются за фиксированное число тактов, что упрощает понимание времени выполнения кода;
- **Большое число регистров общего назначения**, что облегчает работу компилятора и повышает эффективность кода;
- Открытая спецификация архитектуры: можно изучать устройство и принципы работы без “закрытых” ограничений.

В рамках работы особенности архитектуры RISC-V проявляются в том, что студент пишет программу на C, которая выполняется ядром микроконтроллера, а управление периферией (GPIO, таймеры) осуществляется через запись/чтение **регистров памяти** периферийных модулей.

## 3. Биты, маски и регистры: основы bare metal

Для студентов, которые раньше работали в основном с Arduino или HAL-подобными библиотеками, главный переход в bare metal связан не столько с новым синтаксисом языка C, сколько с новым способом мышления. Теперь программист управляет не «готовой функцией», а конкретными битами регистров периферии. Именно поэтому перед работой с GPIO и таймерами полезно отдельно разобрать, что такое бит, маска, регистр и почему одна запись может изменить поведение всего узла микроконтроллера.

В микроконтроллере большинство аппаратных блоков – порт ввода-вывода, таймер, UART, контроллер прерываний – представлены набором регистров по фиксированным адресам памяти. *Регистр* – это обычно 32-битное слово, в котором каждый бит или группа битов отвечает за конкретную функцию: разрешение выхода, выбор режима, флаг события, запуск счетчика и т.д. Когда программа выполняет запись в такой регистр, она фактически управляет аппаратурой.

Минимальная единица такой настройки – бит со значением 0 или 1. Если говорить упрощенно, то 0 обычно означает «сброшено», «выключено» или «запрещено», а 1 – «установлено», «включено» или «разрешено». Но точный смысл всегда определяется описанием конкретного регистра в документации.

Чтобы работать не со всеми 32 битами сразу, а только с нужным, используют маску – число, в котором единицы стоят только в интересующих нас позициях. Например, для вывода PA12 маска часто записывается так:

```
const uint32_t LED = (1u << 12);
```

Эта запись означает: взять число 1 и сдвинуть его влево на 12 позиций. В результате получится число, у которого установлен только бит 12. Символ **u** показывает, что константа беззнаковая, а оператор **<<** выполняет побитовый сдвиг.

В bare metal особенно важны четыре побитовые операции:

- операция | (OR, «или») – установить нужные биты;

- операция **&** (AND, «и») – проверить или оставить только нужные биты;
- операция ~ – инвертировать маску, чтобы затем сбрасывать выбранные биты;
- операция ^ (XOR, «исключающее или») – переключить бит 0→1 или 1→0.

```
reg |= mask; // установить бит
reg &= ~mask; // сбросить бит
if (reg & mask) // проверить состояние бита
reg ^= mask; // переключить бит
```

Во многих регистрах нельзя бездумно записывать произвольное число целиком, потому что вместе с нужным битом можно случайно изменить соседние. Поэтому базовый прием в bare metal – *read-modify-write* («прочитал – изменил только нужные биты – записал обратно»). Именно для упрощения этой задачи у многих периферийных блоков есть специальные регистры вида `SET`, `CLR` и `TGL`.

Для GPIO в К1921ВГ015 это особенно удобно. Вместо записи полного значения в `DATAOUT` можно использовать отдельные регистры, которые меняют только отмеченные маской биты:

```
GPIOA->OUTENSET = LED; // перевести PA12 в режим выхода
GPIOA->DATAOUTSET = LED; // записать на вывод логическую 1
GPIOA->DATAOUTCLR = LED; // записать на вывод логический 0
GPIOA->DATAOUTTGL = LED; // инвертировать текущее состояние
```

Здесь полезно читать выражение по частям: `GPIOA` – это описание блока регистров порта А; оператор -> обращается к конкретному полю структуры; `DATAOUTSET` – имя регистра; `LED` – маска нужного вывода. Такая запись означает не «вызвать функцию», а «записать значение по адресу аппаратного регистра». Обычно такие поля объявлены как *volatile*, чтобы компилятор не удалял и не кэшировал реальные обращения к периферии.

Для новичка важна еще одна идея: высокий уровень на выводе и фактическое «включение устройства» – не всегда одно и то же. Если светодиод подключен по инверсной схеме, то запись 0 может включать его, а запись 1 – выключать. Поэтому в bare metal нужно смотреть не только на код, но и на электрическую схему.

**Важно! При работе с несколькими выводами безопаснее использовать `DATAOUTSET`, `DATAOUTCLR` и `DATAOUTTGL`, а не перезаписывать `DATAOUT` целиком без необходимости.**

Типичные ошибки начинающего разработчика в bare metal:

- забыли перевести вывод в режим выхода перед управлением светодиодом;
- используют «магические числа» вместо понятных масок вида **(1u << n)**;
- изменяют весь регистр сразу, хотя нужно было затронуть только один бит;
- не учитывают инверсную логику подключения `LED`;
- забывают очищать флаг таймера или прерывания после обработки события.

Таким образом, bare metal действительно кажется сложнее Arduino-подхода на старте, потому что требует понимания связи между кодом, регистром и схемой. Но именно эта связь дает реальное понимание того, как микроконтроллер управляет периферией. После освоения битовых операций и масок работа с GPIO, таймерами и прерываниями становится гораздо более предсказуемой.

Ниже приведены дополнительные визуальные примеры, которые помогают увидеть связь между номером бита, 32-битной двоичной маской, записью в регистр и фактическим изменением состояния периферии.

**Наглядный пример: как из номера бита получается 32-битная маска**

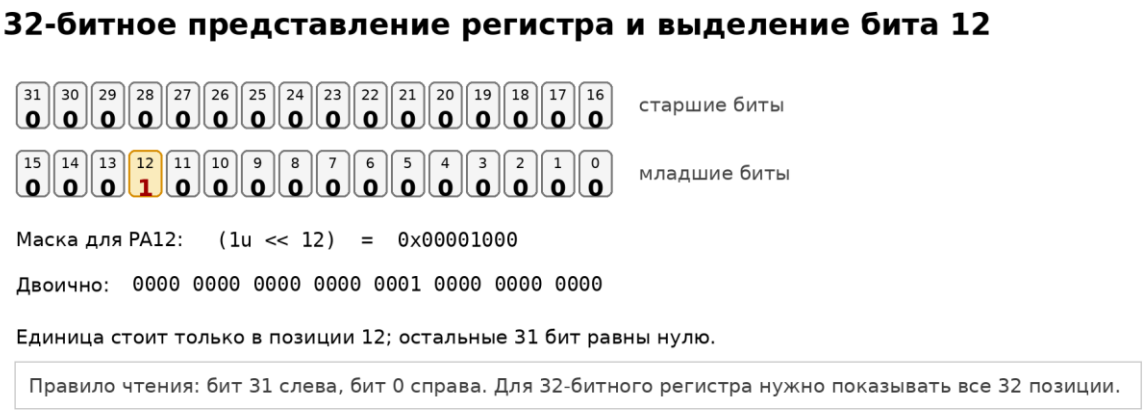



Рисунок 1.2. 32-битное представление регистра и маска для бита 12

**Наглядный пример: четыре базовые побитовые операции**




Рисунок 1.3. Базовые операции над битами и их смысл в bare metal-программе

**Наглядный пример: почему для GPIO лучше использовать SET / CLR / TGL**




Рисунок 1.4. Сравнение прямой записи в DATAOUT и безопасной записи через SET/CLR/TGL

**Наглядный пример: как маска меняет только выбранный бит**

Таблица 1.1

| Операция | Выражение | Было | Стало |
|---|---|---|---|
| Установить бит 12 | reg \|= mask | 0x0550 | 0x1550 |
| Сбросить бит 12 | reg &= ~mask | 0x1550 | 0x0550 |
| Переключить бит 12 | reg ^= mask | 0x0550 | 0x1550 |
| Проверить бит 12 | reg & mask | 0x0550 | 0x0000 (бит не установлен) |

Изменение выбранного бита с помощью маски

**Пошаговый пример: как управлять светодиодом на уровне регистров**

1. Определить номер вывода и сформировать маску, например: `LED = (1u << 12)`.

2. Разрешить вывод как цифровой выход: `GPIOA->OUTENSET = LED`.

3. Подать нужный уровень: `DATAOUTSET` или `DATAOUTCLR`.

4. Если нужно мигание, использовать `DATAOUTTGL` или таймер.

5. Проверить схему платы: иногда логический 0 включает светодиод, а логическая 1 выключает его.

4. Порты ввода-вывода (GPIO)

***GPIO (General Purpose Input/Output)*** – универсальные цифровые линии микроконтроллера, которые могут работать как **входы** (чтение уровня логического сигнала) или как **выходы** (установка логического уровня).

Примеры устройств на GPIO:

- светодиоды;

- кнопки;
- реле;
- цифровые датчики.

### 4.1 Основные регистры управления GPIO (в К1921ВГ015)

Для управления состоянием выводов используются следующие регистры:

- `OUTENSET` – перевод выбранных выводов в режим выхода (разрешение выхода);

Без включения выхода управлять светодиодом записью `DATAOUT` не получится.

- `DATAOUT` – запись полного значения на выходной порт (задает уровни сразу на группе выводов);
- `DATAOUTSET` – установить на выбранных выводах логический уровень **1** (устанавливает биты в 1, остальные не изменяет);
- `DATAOUTCLR` – установить на выбранных выводах логический уровень **0** (сбрасывает биты в 0, остальные не изменяет);
- `DATAOUTTGL` – инвертировать текущее состояние выбранных выводов (переключить 0→1 или 1→0).

### 4.2 Важное замечание про “включение” светодиода

Команды `DATAOUTSET` и `DATAOUTCLR` устанавливают уровень на выводе (1 или 0), но не всегда напрямую означают “включить/выключить светодиод”.

В зависимости от схемы подключения светодиода на плате:

- LED может включаться при уровне **1** (активный “1”);
- или при уровне **0** (активный “0”, инверсная логика).

Поэтому в лабораторной работе предусмотрена отдельная проверка (эксперимент), которая позволяет определить: какая операция (`SET` или `CLR`) фактически включает `LED` на вашей плате.

**Важно! В зависимости от того, куда подключен светодиод (к GND или к VCC), его логика может быть прямой или инверсной.**

## 5. Таймер TMR32

*Таймер* – это отдельный аппаратный модуль, который может выполнять действия периодически.

Как работает таймер:

1) Счетчик таймера начинает отсчет от 0;
2) Когда число достигает значения в регистре `CAPCOM`;
3) Таймер устанавливает флаг события – `RIS`;
4) Если включено прерывание, вызывается обработчик IRQ;
5) Флаг необходимо сбросить через регистр `IC`, чтобы таймер продолжил работу.

Основные регистры таймера:

1) `CTRL` – настройка режима;
2) `CAPCOM` – значение сравнения;

3) `RIS` – флаг прерывания;

4) `IC` – сброс флага прерывания;

5) `IM` – маска разрешения IRQ.

Сводка по основным регистрам таймера TMR32 приведена в таблице 1.2.

Таблица 1.2

| Регистр | Назначение | Типовое действие |
|---|---|---|
| CTRL | Настройка режима и запуск таймера | Задать режим работы и включить таймер |
| CAPCOM | Значение сравнения (период) | Записать число отсчетов до события |
| RIS | Флаг события таймера | Проверить факт срабатывания таймера |
| IC | Сброс флага события | Очистить флаг после обработки события |
| IM | Разрешение запроса прерывания | Включить работу таймера через IRQ |

Основные регистры таймера TMR32

Пример: если вы хотите, чтобы светодиод загорался каждую секунду, вы подбираете такое значение CAPCOM, которое соответствует 1 секунде при тактовой частоте процессора.

**6. Прерывания**

*Прерывания* позволяют микроконтроллеру реагировать на события, не тратя время на постоянные проверки.

Источники прерываний:

1) таймер TMR32;

2) внешние сигналы;

3) системные события;

Обработка прерываний выполняется через контроллер PLIC (Platform Level Interrupt Controller).

Основные функции:

1) `PLIC_SetIrqHandler()` – назначить обработчик;

2) `PLIC_SetPriority()` – приоритет;

3) `PLIC_IntEnable()` – разрешить источник прерывания;

4) `InterruptEnable()` – разрешить глобальные IRQ.

Примеры взаимодействия с системой прерываний приведены в таблице 1.3.

Таблица 1.3

| Элемент | Назначение | Пример взаимодействия |
|---|---|---|
| TMR32->IM | Разрешение запроса прерывания от таймера | TMR32->IM = 1; |
| PLIC_SetIrqHandler() | Назначение пользовательского обработчика | PLIC_SetIrqHandler(TMR32_IRQn, TMR32_IRQHandler); |
| PLIC_SetPriority() | Задание приоритета источника | PLIC_SetPriority(TMR32_IRQn, 1); |
| PLIC_IntEnable() | Разрешение источника в PLIC | PLIC_IntEnable(TMR32_IRQn); |

| Элемент | Назначение | Пример взаимодействия |
|---|---|---|
| InterruptEnable() | Глобальное разрешение прерываний | InterruptEnable(); |
| TMR32_IRQHandler() | Выполнение действия по событию таймера | В обработчике можно переключить светодиод. |
| TMR32->IC | Сброс флага после обработки | TMR32->IC = 1; |

Взаимодействие с системой прерываний PLIC

Как работает:

1) Источник (например, таймер) генерирует событие;
2) PLIC вызывает вашу функцию-обработчик;
3) В обработчике вы выполняете нужную логику (например, переключение `LED`);
4) Обязательно сбрасываете флаг `IC`.

# Практическая работа 1
# Работа с GPIO и таймерами

1. Цель и оборудование
2. Структура проекта и основные файлы
3. Подготовка оборудования и программной среды
3.1 Подключение платы и установка драйверов
3.2 Установка и настройка Visual Studio Code
3.3 Установка SDK и Toolchain
4. Создание, сборка и прошивка проекта
4.1 Создание нового проекта на основе примера
4.2 Замена файла main.c
4.3 Сборка проекта
4.4 Прошивка микроконтроллера
5. Практические задания
5.1 Общие правила для всех вариантов
5.2 Задание 1.1. Мигающий светодиод через программную задержку
5.3 Задание 1.2. Мигающий светодиод через таймер TMR32
5.4 Задание 1.3. Управление светодиодом с использованием SET и CLR
5.5 Задание 1.4. Изменение частоты мигания через период таймера
5.6 Задание 1.5. Использование таймера в режиме опроса флага
6. Контрольные вопросы
7. Тестовые вопросы

## ЦЕЛЬ И ОБОРУДОВАНИЕ

**Цель:** освоить базовые принципы работы с портами ввода-вывода (GPIO), таймерами и прерываниями микроконтроллера К1921ВГ015, а также научиться создавать и настраивать проект в среде VS Code с использованием SDK и инструментов НИИЭТ.

**Оборудование:** отладочная плата с микроконтроллером К1921ВГ015; персональный компьютер с доступом в интернет; USB-кабель для подключения платы к компьютеру.

## СТРУКТУРА ПРОЕКТА И ОСНОВНЫЕ ФАЙЛЫ

Основные файлы проекта можно увидеть на рисунке 1.5.

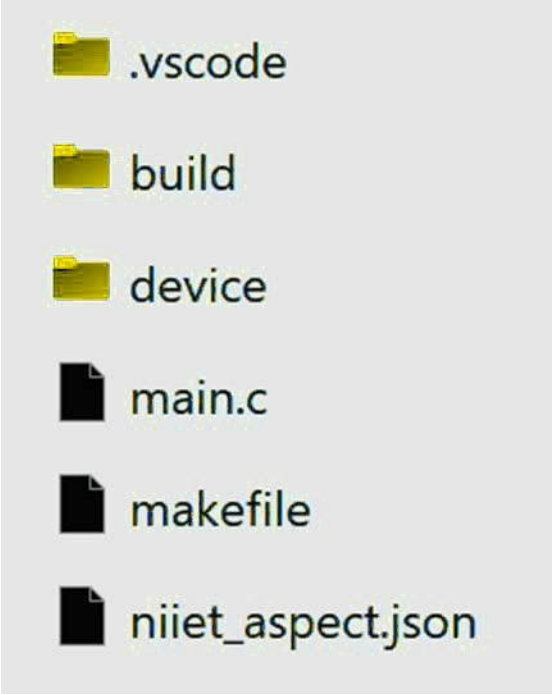



Рисунок 1.5. Структура файлов проекта

Таблица 1.4 Основные файлы проекта

| Категория | Файл | Назначение | Как работает |
|---|---|---|---|
| Основные файлы проекта | main.c | Ваш основной рабочий файл с кодом программы. | Пишете здесь всю логику для выполнения лабораторных работ. |
| Основные файлы проекта | makefile | Скрипт для автоматической сборки проекта. | При запуске команды make в терминале:<br>Компилирует все .c файлы в .o (объектные файлы)<br>Линкует их вместе в исполняемый файл .elf<br>Создает файлы .hex и .dump для прошивки и отладки |
| Основные файлы | config.json | Конфигурация проекта | Настройки пути к файлам, |

| Категория | Файл | Назначение | Как работает |
|---|---|---|---|
| проекта | | для среды разработки. | параметры компилятора. |
| Основные файлы проекта | niiet_aspect.json | Специфичная конфигурация для инструментов NIIET (производитель микроконтроллера). | ВАЖНО!!!<br>Чтобы прошить плату, в этом файле нужно: Изменить в этой строке “ram” на “flash”: "scriptFile": "./ldscripts/k1921vg015_ram.ld "scriptFile": "./ldscripts/k1921vg015_flash.ld"<br>ВАЖНО!!! |
| Системные файлы (НЕ ТРОГАЙТЕ без необходимости): | system_k1921vg015.c | Системная инициализация микроконтроллера. | Настраивает тактовую частоту (SystemCoreClock)<br>Инициализирует системные прерывания<br>Настраивает базовые периферийные модули |
| Системные файлы (НЕ ТРОГАЙТЕ без необходимости): | sys_init.c | Дополнительная системная инициализация. | Подготовка системы перед запуском main(). |
| Системные файлы (НЕ ТРОГАЙТЕ без необходимости): | startup_k1921vg015.S | Ассемблерный стартовый файл. | Задает таблицу векторов прерываний<br>Инициализирует стек<br>Копирует данные из Flash в RAM<br>Обнуляет секцию BSS (неинициализированные переменные)<br>Переходит к вызову main() |
| Системные файлы (НЕ ТРОГАЙТЕ без необходимости): | retarget.c | Перенаправление стандартного ввода/вывода.<br>Что позволяет: Использовать printf() для вывода в UART (последовательный порт).<br>Важно: Без этого файла printf не будет работать. | |
| Системные файлы (НЕ ТРОГАЙТЕ без необходимости): | printf.c | Реализация функции форматированного вывода. | Преобразует данные в строку для вывода. |
| Системные файлы (НЕ ТРОГАЙТЕ без необходимости): | plic.c | Программный контроллер прерываний (PLIC - Platform Level Interrupt Controller). | Управление прерываниями, приоритетами, обработчиками. |

Таблица 1.5 Директории проекта

| Директория | Назначение | Содержит |
|---|---|---|
| .vscode/ | Настройки среды разработки VSCode. | launch.json - настройки отладчика<br>tasks.json - задачи для сборки (вызов make)<br>settings.json - параметры редактора |
| build/ | Директория для сгенерированных файлов (создается автоматически). | .o файлы - результат компиляции исходных файлов<br>.elf - исполняемый файл в формате ELF<br>.hex - файл для прошивки микроконтроллера<br>.dump - файл для отладки с дизассемблированным кодом |
| device/Include/ | Заголовочные файлы (.h). | K1921VG015.h - основной файл с определениями всех регистров МК. |
| device/Ldscripts/ | Скрипты линковщика. | k1921vg015_flash.ld - для размещения программы во Flash-памяти<br>k1921vg015_ram.ld - для размещения программы в RAM<br>k1921vg015.common.lds - общие настройки линковки |
| device/Source/ | Исходные системные файлы. | |

Директории проекта

## ПОДГОТОВКА ОБОРУДОВАНИЯ И ПРОГРАММНОЙ СРЕДЫ

### Подключение платы и установка драйверов

1) Подключите плату К1921ВГ015 к компьютеру с помощью USB-кабеля. Порт USB Type C отмечен на рисунке 1.6;

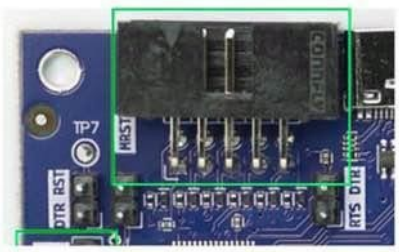



Рисунок 1.6. Подключение платы




Рисунок 1.7. Положение перемычек

2) Установите перемычку USB 5V JACK в положение питания от USB;

3) Установите перемычку DBG MAIN REG (рисунок 1.6) в положение питания микроконтроллера от программатора (рисунок 1.7)

4) Проверьте, что перемычка FT_RST (рисунок 1.7) установлена (сброс программатора снят);

5) Чтобы установить драйверы для JTAG-эмулятора необходима программа «Zadig». Программа «Zadig» находится в каталоге «sc-dt\tools\bin». Каталог относится к

комплекту инструментов НИИЭТ / SDK. Скачать другие версии программы «Zadig» можно по ссылке: https://zadig.akeo.ie;

*Пункты 6-8 нужны, если вы подключаете микроконтроллер к ПК с помощью внешнего JTAG-программатора (обычно работают на основе микросхемы FT2232H, например, отечественный ELJTAG). В противном случае, сразу смотрите пункт 9.*

6) Подключите JTAG-адаптер к USB-порту вашего ПК. Дождитесь, пока закончится процедура автоматической установки драйверов средствами Windows. Независимо от успешности ее результата по окончании запустите программу «Zadig» (рисунок 1.8);

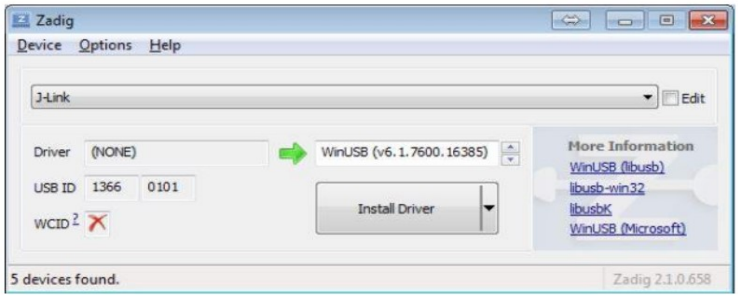



Рисунок 1.8. Окно программы Zadig

7) В меню «Options» поставьте галочку «List all devices». В выпадающем списке выберите используемое устройство (в нашем примере «J-Link») или неизвестное устройство (убедитесь в таком случае, что это именно jtag подключая/отключая USB кабель);

8) В ОС Windows JTAG-эмулятор «J-Link» определяется как два интерфейса: – "CDC (Interface 0)" можно использовать в качестве USB-UART преобразователя. – "BULK Interface (Interface 2)" используется в качестве JTAG отладчика. В ОС Windows необходимо сделать подмену драйвера на WinUSB с помощью программы zadig;

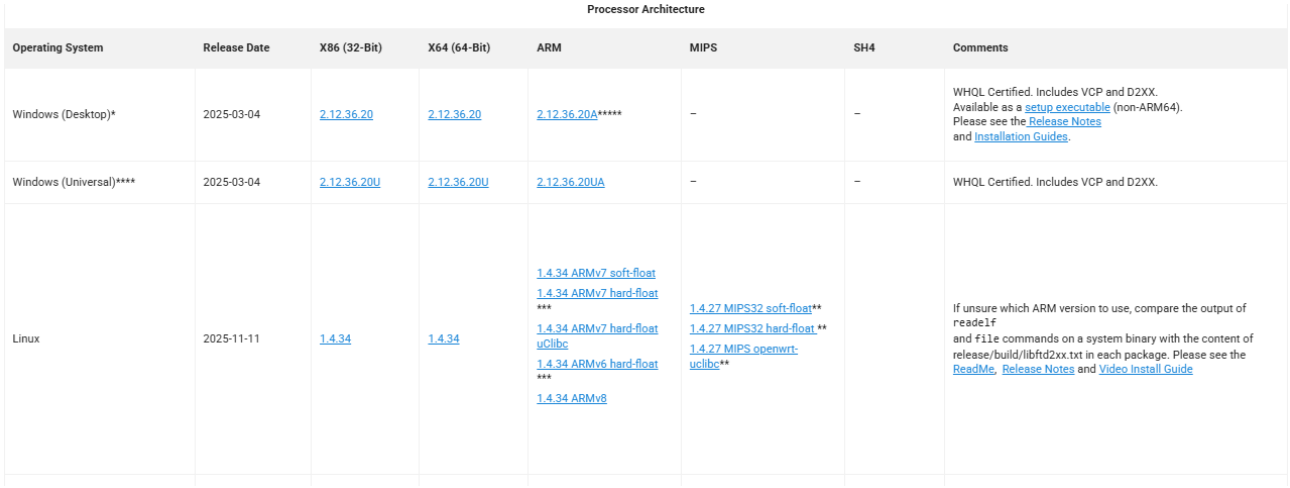
9) В ОС Windows ИС FT2232 (FTDI), установленная на плате К1921ВГ015 (КФДЛ.441461.029) определяется как два интерфейса: – "Dual RS232-HS (Interface 0)" на плате используется в качестве USB-UART преобразователя, подключенного к UART0 и должен использовать оригинальные драйвера "FTDI". – "Dual RS232-HS (Interface 1)" на плате используется в качестве JTAG отладчика. В ОС Windows необходимо сделать подмену драйвера на WinUSB с помощью программы zadig;

10) В строке справа от стрелки выберите драйвер «WinUSB (v.xxxx)». Нажмите на кнопку «Install Driver» (если кнопка называется по-другому, значит у вас установлены другие драйвера и нужно нажать «Replace Driver»). Драйвера установлены. Однако по опыту эксплуатации рекомендуется вынуть/вставить JTAG из USB компьютера – без этого драйвер может не заработать. В некоторых случаях требуется перезагрузка.

11) Если драйвер не установился или не работает, можно попробовать нажать на кнопку «Install Driver» несколько раз (если «Zadig» сообщает, что установка не удалась), вынуть/вставить JTAG из USB, перезагрузить компьютер, отключить антивирус, запустить программу с правами администратора, выключить службу "Центр обновления Windows" (если она включена), проверить состояние устройства в диспетчере устройств Windows. Чтобы вернуться к стандартным драйверам JTAG для работы с другой средой, в диспетчере устройств Windows найдите свое устройство (например J-Link), правой кнопкой → Обновить

драйверы... → Выполнить поиск драйверов на этом компьютере → Выбрать драйвер из списка уже установленных" и выбрать, например, "J-link driver";

*Пункт 11, опять же, актуален, если используется внешний программатор. Если же у вас не определяется драйвер внутреннего программатора на плате (это FTDI), то необходимо перейти на сайт https://ftdichip.com/drivers/d2xx-drivers/ и скачать драйвер под вашу ОС:*



Processor Architecture

| Operating System | Release Date | X86 (32-Bit) | X64 (64-Bit) | ARM | MIPS | SH4 | Comments |
|---|---|---|---|---|---|---|---|
| Windows (Desktop)* | 2025-03-04 | 2.12.36.20 | 2.12.36.20 | 2.12.36.20A***** | – | – | WHQL Certified. Includes VCP and D2XX.<br>Available as a setup executable (non-ARM64).<br>Please see the Release Notes<br>and Installation Guides. |
| Windows (Universal)**** | 2025-03-04 | 2.12.36.20U | 2.12.36.20U | 2.12.36.20UA | – | – | WHQL Certified. Includes VCP and D2XX. |
| Linux | 2025-11-11 | 1.4.34 | 1.4.34 | 1.4.34 ARMv7 soft-float<br>1.4.34 ARMv7 hard-float ***<br>1.4.34 ARMv7 hard-float uClibc<br>1.4.34 ARMv6 hard-float ***<br>1.4.34 ARMv8 | 1.4.27 MIPS32 soft-float**<br>1.4.27 MIPS32 hard-float **<br>1.4.27 MIPS openwrt-uclibc** | | If unsure which ARM version to use, compare the output of `readelf` and `file` commands on a system binary with the content of release/build/libftd2xx.txt in each package. Please see the ReadMe, Release Notes and Video Install Guide |

Рисунок 1.9. Проверка установленного драйвера в Zadig

12) После замены драйвера в главном окне программы zadig мы также можем убедиться, что драйвер заменился. Так, при выборе используемого интерфейса, напротив надписи "Driver" должен отобразиться драйвер "WinUSB".

**Установка и настройка Visual Studio Code**

1) Установите Visual Studio Code, затем откройте раздел Extensions (Ctrl+Shift+X).

2) Для установки расширения скачайте файл niiet-aspect-x.x.x.vsix из комплекта НИИЭТ. В окне Extensions откройте меню Views and More Actions и выберите команду Install from VSIX....

3) В открывшемся диалоге укажите скачанный файл расширения. После установки в левой панели появится значок NIIET Aspect.

4) Если расширение ms-vscode.cpptools не установилось автоматически, установите его отдельно – оно используется NIIET Aspect для поддержки C/C++.

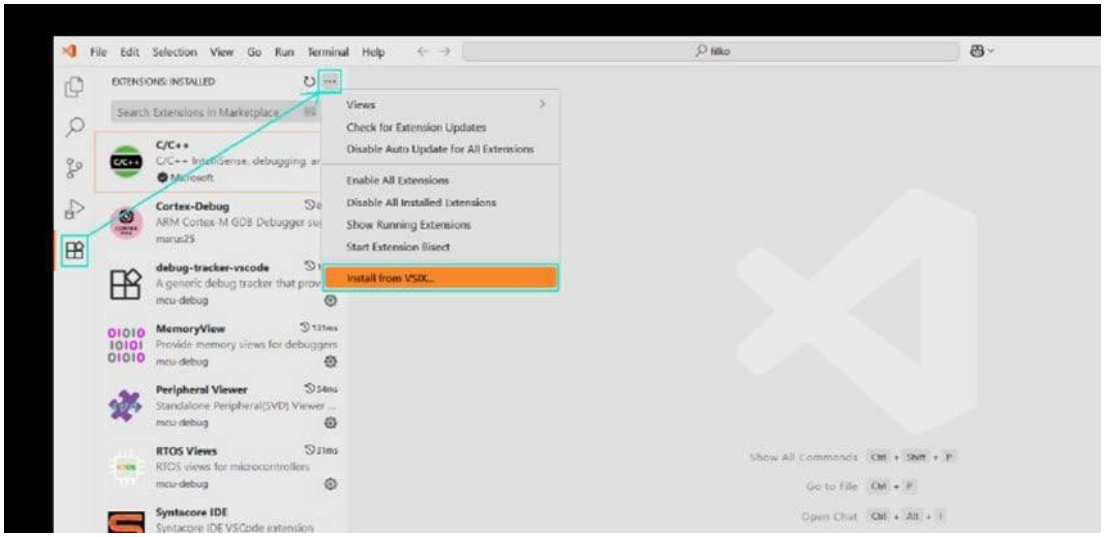



Рисунок 1.10. Установка расширения NIIET Aspect

После установки расширения в левой части окна VS Code появляется отдельная панель NIIET Aspect для работы с проектами и инструментами.

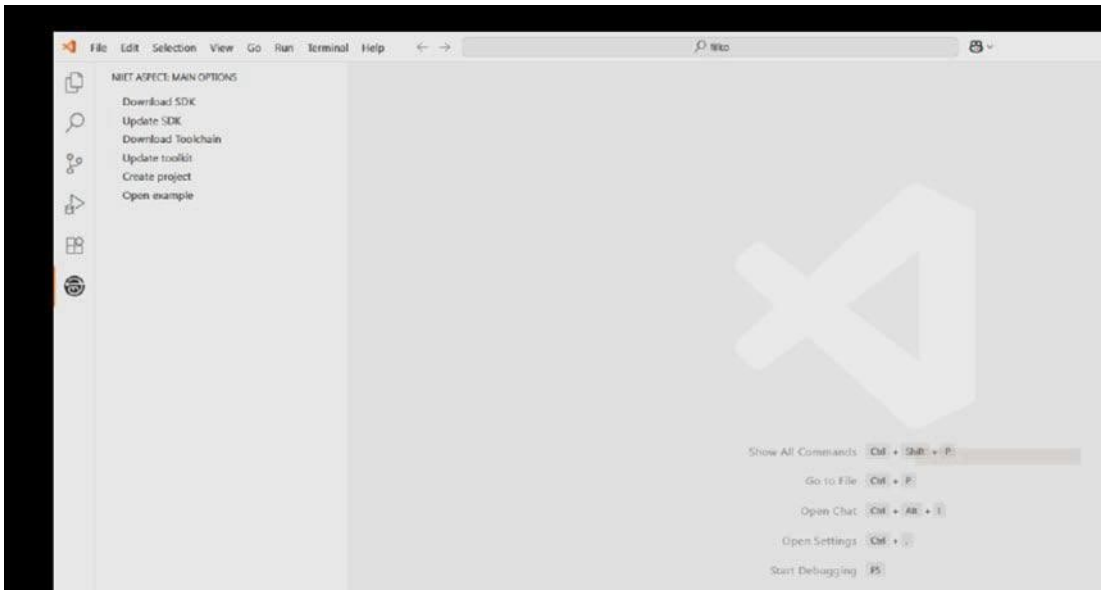



Рисунок 1.11. Меню расширения NIIET Aspect

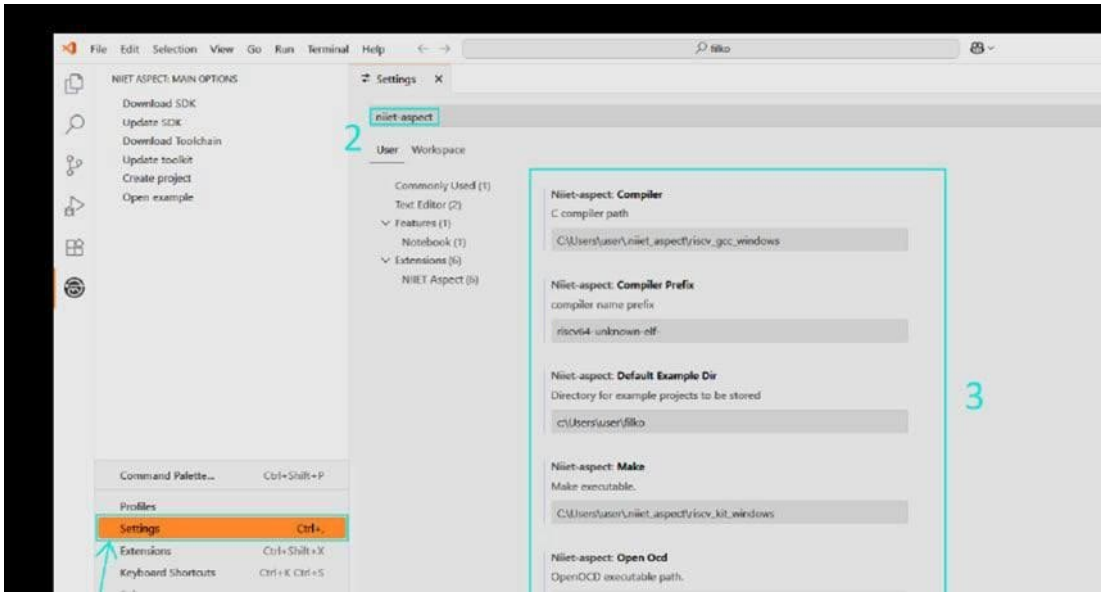



Рисунок 1.12. Настройки расширения NIIET Aspect

**Установка SDK и Toolchain**

1) После установки расширения откройте панель NIIET Aspect. Из этого меню можно загрузить SDK, компилятор и инструменты отладки.

2) Для установки компонентов через расширение используйте пункт Download toolchain и выберите, что требуется: SDK, компилятор или отладочные инструменты.

3) Если архивы устанавливаются вручную, откройте Settings (**Ctrl+**), введите в поиске niiet-aspect и задайте пути к SDK, компилятору, make и OpenOCD.

4) Без установленного SDK расширение не сможет создать новый проект и открыть пример.

## СОЗДАНИЕ, СБОРКА И ПРОШИВКА ПРОЕКТА

### Создание нового проекта на основе примера

**Это обязательный шаг – нельзя просто открыть папку из архива Lab1.**

1) Откройте VS Code;
2) В левой панели нажмите **NIIET Aspect → Open Example**;
3) Выберите пример: Run_leds (bare);
4) В параметре OpenOCD Debugger interface выберите: **onboard_ftdi** – это включает встроенный отладчик платы;
5) Нажмите **Create** и дождитесь, пока проект откроется.

Замена файла main.c

1) В VS Code откройте папку проекта;
2) Найдите файл: main.c;
3) Удалите его или переименуйте;
4) Откройте архив Lab1;
5) Скопируйте файл main.c из архива в папку проекта;
6) Сохраните файл.

Теперь ваш проект соответствует лабораторной работе.

### Сборка проекта

1) Сборку проекта выполните через кнопку **Compile project** на панели NIIET Aspect или через соответствующую задачу VS Code;

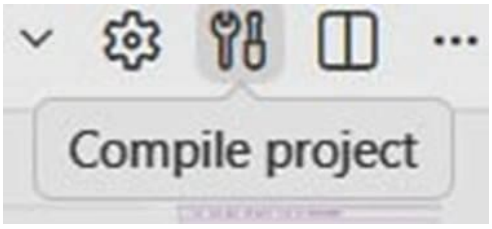



Рисунок 1.13. Компиляция проекта

2) После успешной компиляции в каталоге проекта создается исполняемый файл проекта с расширением .elf;
3) При необходимости также могут формироваться вспомогательные файлы сборки (.hex, .dump и др.);
4) Если сборка завершилась без ошибок, проект готов к прошивке и отладке;
5) Нажимаем на **Build Project**.

### Прошивка микроконтроллера

1) Убедитесь, что для проекта выбрана отладочная конфигурация Debug NIIET;

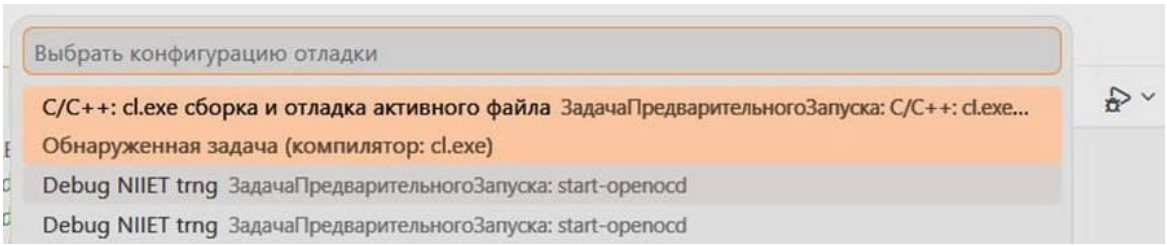


Рисунок 1.14. Режим отладки Debug

2) Для прошивки микроконтроллера запустите эту конфигурацию из списка отладки VS Code;

3) Дождитесь завершения загрузки программы в память микроконтроллера;

4) После запуска отладки можно использовать точки останова и пошаговое выполнение программы;

5) Нажать кнопку **Reset**.

**ПРАКТИЧЕСКИЕ ЗАДАНИЯ**

**Общие правила для всех вариантов:**

1. Файл main.c не переписывать. Дописывать код только:
   - внутри `while(1) { … }` (основное место);
   - и/или добавить свои функции выше `main()`.
2. В этой лабораторной работаем **только** с пользовательскими LED платы: **PA12, PA13, PA14, PA15**.
3. Для работы с выбранными LED необходимо заранее определить соответствующие маски (например, через константы или макросы выше main()):

```
// Один LED (для варианта 1)
const uint32_t LED = (1u << 12); // можно 13/14/15
// Четыре LED (для вариантов 2 и 3)
const uint32_t L0 = (1u<<12);
const uint32_t L1 = (1u<<13);
const uint32_t L2 = (1u<<14);
const uint32_t L3 = (1u<<15);
const uint32_t ALL = L0|L1|L2|L3;
```

4. Команды управления `LED` (их использовать везде):

```
GPIOA->DATAOUTTGL = LED; // переключить выбранный LED (toggle)
GPIOA->DATAOUTSET = LED; // записать "1" на вывод
GPIOA->DATAOUTCLR = LED; // записать "0" на вывод
```

5. Задержка (нужна для первых шагов): Добавить выше `main()`:

```
static void delay(volatile uint32_t n)
{
while (n--) { __asm volatile("nop"); }
}
```

6. Перед управлением светодиодом выполнить `GPIOA->OUTENSET = LED`; для одного светодиода или `GPIOA->OUTENSET = ALL`; для группы светодиодов.

**Задание 1.1. Мигающий светодиод через программную задержку**

**Цель:** Научиться управлять светодиодом с использованием программных задержек.

**Пояснение:**

В этом задании вам предстоит написать функцию задержки, которая будет создавать паузу за счет циклического уменьшения счетчика. Это простая задержка, которая блокирует выполнение программы, пока не будет достигнуто нулевое значение счетчика. Она используется для того, чтобы светодиод мигал с определенной частотой.

**Ход работы:**

1. Реализуйте функцию задержки `delay()`, которая создает паузу за счет циклического уменьшения счетчика;
2. В основном цикле программы организуйте мигание светодиода `LED0` (пин PA12) с использованием функции `delay()`;
3. Измените параметр задержки (величина задержки должна быть больше 2 мм) и наблюдайте изменение частоты мигания.

**Ожидаемый результат:**

- Светодиод `LED0` мигает с постоянной частотой;
- Изменение значения в функции `delay()` приводит к изменению частоты мигания.

**Задание 1.2. Мигающий светодиод через таймер TMR32**

**Цель:** Освоить использование таймера для организации временных задержек через прерывания.

**Пояснение:**

Задание направлено на использование аппаратного таймера микроконтроллера для создания более точных задержек без блокировки работы процессора. В отличие от программных задержек, таймер позволяет выполнять другие операции, пока он отсчитывает время.

**Ход работы:**

1. Реализуйте функцию инициализации таймера `TMR32_init();`
2. Настройте таймер на генерацию прерываний с заданным периодом;
3. Реализуйте обработчик прерывания `TMR32_IRQHandler()`, в котором будет переключаться состояние светодиода `LED0`;
4. В основной программе разрешите прерывания и запустите таймер.

**Ожидаемый результат:**

- Светодиод `LED0` мигает с заданной частотой;
- Основной цикл программы пустой, вся работа выполняется в прерывании;
- Изменение периода таймера приводит к изменению частоты мигания.

**Задание 1.3. Управление светодиодом с использованием SET и CLR**

**Цель:** Изучить различные способы управления состоянием выходов GPIO.

**Пояснение:**

В этом задании вы будете работать с командами для управления состоянием светодиодов через использование регистров GPIO.

**Ход работы:**

1. Организуйте включение светодиода `LED0` на 1 секунду с использованием `DATAOUTCLR`;
2. Выключите светодиод на 1 секунду с использованием `DATAOUTSET`.
3. Повторите цикл несколько раз;
4. Экспериментально определите, какая операция включает светодиод.

**Ожидаемый результат:**

- Светодиод включается и выключается на равные промежутки времени;
- Демонстрируется понимание разницы между `SET` и `CLR` операциями.

**Задание 1.4. Изменение частоты мигания через период таймера**

**Цель:** Научиться рассчитывать период таймера для получения заданной частоты мигания.

**Пояснение:**

В этом задании вы научитесь изменять частоту мигания светодиода, регулируя период таймера. Вы будете использовать таймер для генерации прерываний, чтобы управлять частотой мигания светодиода.

**Ход работы:**

1. Используя реализацию из задания 1.2, настройте таймер на три различных частоты:
   – Приблизительно 1 Гц (1 раз в секунду);
   – Приблизительно 2 Гц (2 раза в секунду);
   – Приблизительно 0.5 Гц (1 раз в 2 секунды).
2. Для каждой частоты рассчитайте значение для регистра `CAPCOM[0].VAL`.
3. Организуйте последовательное переключение между этими режимами.

**Ожидаемый результат:**

- Демонстрация трех различных частот мигания.
- Понимание связи между значением периода и частотой тактирования.

**Задание 1.5. Использование таймера в режиме опроса флага**

**Цель:** Освоить альтернативный способ работы с таймером без использования прерываний.

**Пояснение:**

Задание направлено на использование режима опроса флага вместо использования прерываний для управления таймером. Этот метод менее эффективен по сравнению с прерываниями, так как требует постоянной проверки состояния флага, но в некоторых случаях может быть полезен.

**Ход работы:**

1. Настройте таймер как в задании 1.2, но без разрешения прерываний;
2. В основном цикле программы организуйте опрос флага RIS таймера;
3. При обнаружении установленного флага переключайте состояние светодиода и сбрасывайте флаг.

**Ожидаемый результат:**

- Светодиод мигает с заданной частотой;
- Программа использует опрос флага вместо прерывания;
- Основной цикл не блокируется на время задержки.

**КОНТРОЛЬНЫЕ ВОПРОСЫ**

1. Что такое GPIO и для чего он используется в микроконтроллерах?
2. Что означает “настроить вывод как выход” и что произойдет, если этого не сделать перед управлением светодиодом?
3. В чем разница между командами “установить 1”, “установить 0” и “переключить состояние” для вывода (`SET/CLR/TGL`)?
4. Почему в лабораторной сначала выполняют проверку `SET/CLR` для светодиода? Что она позволяет понять?
5. Как устроена программная задержка `delay()` (в общих словах) и почему она “тормозит” работу программы?
6. Что такое аппаратный таймер и зачем он нужен, если можно сделать задержку циклом?
7. Что такое прерывание и как программа “попадает” в обработчик прерывания?
8. Зачем в обработчике прерывания таймера нужно очищать (сбрасывать) флаг события таймера? Что будет, если не очищать?
9. Чем отличается работа таймера через прерывание (IRQ) от работы через опрос флага (polling)? Когда удобнее одно, а когда другое?
10. Если светодиод мигает слишком быстро или слишком медленно, какие параметры вы будете менять в программе, чтобы отрегулировать частоту мигания?

**ТЕСТОВЫЕ ВОПРОСЫ**

Выберите один правильный ответ.

**1. Что означает сокращение GPIO?**

А) Универсальный порт ввода-вывода.
Б) Генератор программных операций.
В) Группа периферийных таймеров.

**2. Какой регистр используется для перевода вывода в режим выхода?**

А) OUTENSET.
Б) DATAOUTSET.
В) RIS.

**3. Какой регистр устанавливает логический уровень 1 на выбранных выводах?**

А) DATAOUTCLR.
Б) DATAOUTSET.
В) IC.

**4. Для чего в лабораторной работе выполняют проверку SET/CLR при управлении светодиодом?**

А) Чтобы определить, какая операция реально включает светодиод на данной плате.
Б) Чтобы увеличить тактовую частоту микроконтроллера.
В) Чтобы очистить память Flash перед прошивкой.

**5. В чем основной недостаток программной задержки delay()?**

А) Она требует обязательного использования UART.
Б) Она блокирует выполнение основной программы на время ожидания.
В) Она работает только вместе с прерываниями.

**6. Какой регистр таймера хранит значение сравнения периода?**

А) CAPCOM.
Б) CTRL.
В) IM.

**7. Что происходит, когда счетчик таймера достигает значения CAPCOM?**

А) Микроконтроллер автоматически перезагружается.
Б) Устанавливается флаг события RIS.
В) Все выводы GPIO переходят в нулевое состояние.

**8. Какой регистр используется для сброса флага прерывания таймера?**

А) IC.
Б) OUTENSET.
В) DATAOUTTGL.

**9. Какая функция назначает обработчик прерывания в контроллере PLIC?**

А) PLIC_SetIrqHandler().
Б) InterruptEnable().

В) make().

**10. Чем отличается работа через прерывание от работы через опрос флага?**

А) При опросе флага программа сама постоянно проверяет состояние таймера.

Б) При опросе флага прерывания работают быстрее, чем IRQ.

В) Между этими режимами нет различий.

**11. Какой файл является основным для написания логики лабораторной работы?**

А) main.c.

Б) printf.c.

В) startup_k1921vg015.S.

**12. Какая команда используется для сборки проекта во встроенном терминале VS Code?**

А) build.

Б) make.

В) run.

**13. Что нужно изменить в файле niiet_aspect.json перед прошивкой платы во Flash?**

А) Заменить scriptFile с k1921vg015_ram.ld на k1921vg015_flash.ld.

Б) Удалить файл config.json.

В) Переименовать main.c в startup.c.

**14. Где по условиям лабораторной работы разрешено дописывать основной код студента?**

А) Только в файле plic.c.

Б) Внутри while(1) и/или в собственных функциях выше main().

В) Только в makefile.

**15. Почему в обработчике прерывания таймера нужно очищать флаг события?**

А) Чтобы таймер мог корректно формировать следующие события и прерывания.

Б) Чтобы светодиод начал мигать быстрее.

В) Чтобы включился режим отладки через UART.

**16. Какой регистр таймера разрешает запрос прерывания от таймера?**

А) IM.

Б) IC.

В) RIS.

**17. Какая функция разрешает глобальные прерывания в программе?**

А) PLIC_SetPriority()

Б) InterruptEnable()

В) TMR32_IRQHandler()

**18. Какой регистр GPIO используется для инвертирования текущего состояния выбранного вывода?**

А) DATAOUTSET.

Б) DATAOUTCLR.

В) DATAOUTTGL.

**19. Какой файл проекта содержит определения регистров микроконтроллера К1921ВГ015?**

А) K1921VG015.h.

Б) makefile.

В) retarget.c.

**20. В каком режиме работы таймера основная программа постоянно проверяет состояние флага RIS?**

А) В режиме опроса флага.

Б) В режиме обработки через UART.

В) В режиме прошивки Flash.

# Глава 2. Реакция на события: Внешние прерывания и NVIC

Прерывания — это фундаментальный механизм, позволяющий микроконтроллеру мгновенно реагировать на внешние события (нажатие кнопки, приход байта по UART) или внутренние ситуации (завершение преобразования АЦП), не тратя ресурсы ядра на постоянный опрос регистров периферии (*polling*). В архитектуре RISC-V, используемой в К1921ВГ015, управление прерываниями разделено между ядром процессора и внешним контроллером.

**Общие сведения о микроконтроллере К1921ВГ015**

К1921ВГ015 – отечественный 32-битный микроконтроллер на архитектуре RISC-V. Он применяется в учебных, промышленных и исследовательских проектах благодаря открытой архитектуре ядра и наличию встроенных периферийных модулей. Подробные сведения о микроконтроллере приведены в лабораторной работе №1.

## 2.2 Кнопка WAKEUP0

WAKEUP0 – аппаратная кнопка на макетно-отладочной плате. Сигнал кнопки скоммутирован с выводом WAKEUP0 и линией PA11 микроконтроллера.

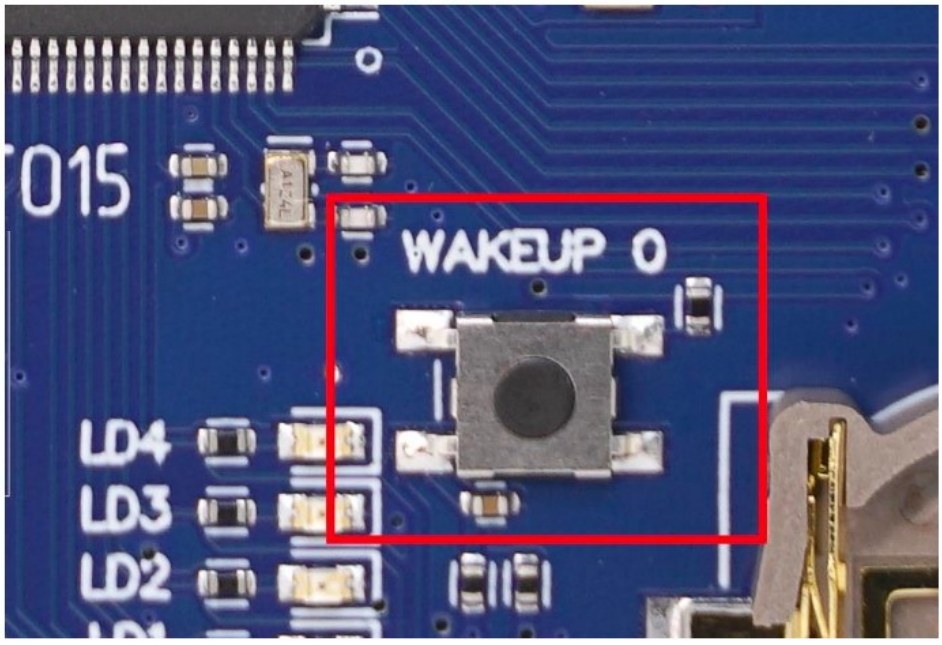



Рисунок 2.1. Кнопка WAKEUP0 на макетно-отладочной плате

Кнопка формирует управляющий сигнал при нажатии и может использоваться для пробуждения микроконтроллера или для обработки внешнего события программой.

В данной работе кнопка WAKEUP0 используется как вход управления программой; при нажатии изменяется скорость эффекта «бегущий огонь».

В микроконтроллере К1921ВГ015 предусмотрены входы пробуждения WAKE0, WAKE1 и WAKE2. В данной работе используется вход WAKE0, состояние или событие по которому фиксируется в регистрах блока RTC и управления энергопотреблением.

Типовая схема обработки включает настройку полярности входа, разрешение WAKEUP0, фиксацию события в поле `RTC_HISTORY.WAKE0` и последующий сброс этого события записью логической единицы.

**Примечание.** Если на конкретной плате кнопка не подключена к линии WAKE0 напрямую, подключение следует выполнить в соответствии с *pinout* платы.

### 2.3 Порты ввода-вывода (GPIO)

GPIO – универсальные цифровые линии микроконтроллера, которые могут работать как входы для чтения уровня сигнала и как выходы для установки логического уровня.

На плате реализована пользовательская световая индикация, связанная с линиями порта GPIOA. Для лабораторной работы используются светодиоды, расположенные в зоне PA12–PA15.

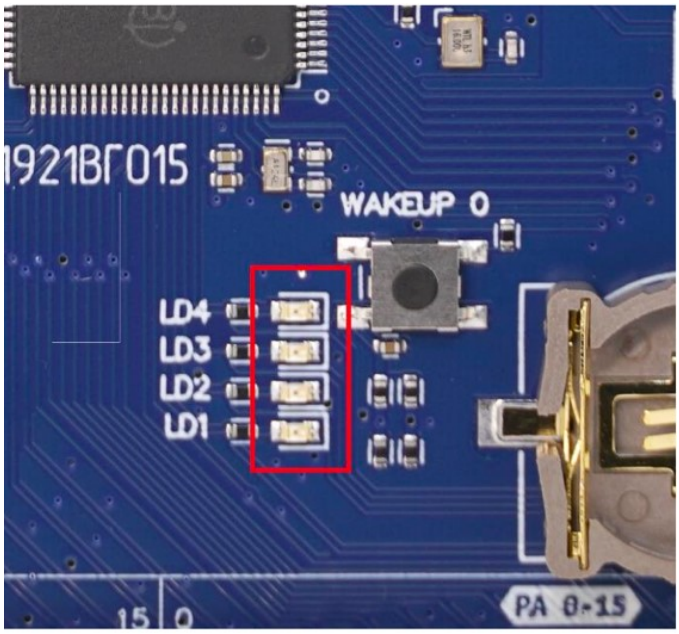



Рисунок 2.2. Расположение пользовательской световой индикации

Управление светодиодами через GPIO позволяет организовывать включение и выключение индикации, мигание и отображение текущего состояния программы.

Для управления состоянием выводов используются регистры `OUTENSET`, `DATAOUT`, `DATAOUTSET`, `DATAOUTCLR` и `DATAOUTTGL`. Регистр `OUTENSET` переводит выбранные линии в режим выхода; `DATAOUT` записывает полное значение на выходной порт; `DATAOUTSET` и `DATAOUTCLR` изменяют только выбранные биты; `DATAOUTTGL` инвертирует текущее состояние выбранных линий.

Операции `DATAOUTSET` и `DATAOUTCLR` задают уровень на выводе, однако фактическое включение светодиода зависит от схемы подключения. Светодиод может иметь как прямую, так и инверсную логику, поэтому перед выполнением задания требуется экспериментально определить, какая операция соответствует включению индикации.

### 2.4 Эффект «Бегущий огонь»

Эффект «бегущий огонь» представляет собой последовательное включение светодиодов с фиксированным интервалом между переключениями. Изменение значения задержки или периода таймера позволяет изменять скорость работы эффекта.

В программной реализации обычно используется *битовая маска* `led_shift`, обозначающая активный светодиод, операции сдвига маски влево или вправо и возврат маски к начальному положению после выхода за допустимый диапазон.

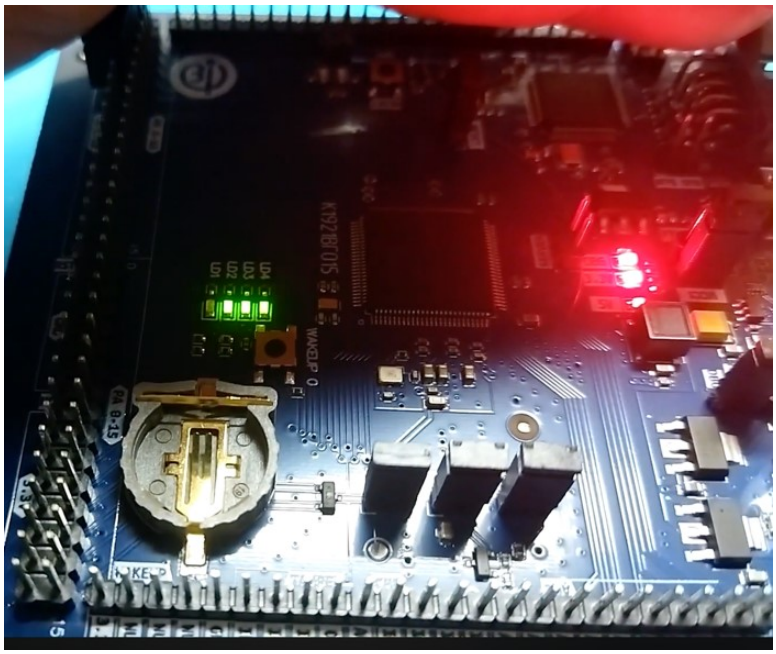


Рисунок 2.3. Эффект «Бегущий огонь»

В ряде демонстрационных примеров применяется маска светодиодов на битах 8…15 порта GPIOA. При выполнении лабораторной работы необходимо сопоставить программные маски с фактической разводкой светодиодов на используемой плате.

### 2.5 Таймер TMR32 и изменение скорости

Для формирования равномерных переключений используется 32-битный таймер TMR32, работающий в режиме периодических событий. Скорость эффекта определяется значением периода таймера; в базовом варианте используются режимы `slow_period` и `fast_period`.

После возникновения события WAKEUP0 программа изменяет значение регистра сравнения таймера `CAPCOM[0].VAL`, вследствие чего «бегущий огонь» ускоряется или замедляется.

Механическая кнопка подвержена *дребезгу*, поэтому в программе необходимо предусмотреть подавление ложных срабатываний. В простейшем случае применяется программная задержка после обработки события; допустимы также фильтрация по времени, раздельная обработка нажатия и отпускания и подтверждение события таймером.

# Практическая работа №2

Использование кнопки WAKEUP0 как источника внешнего прерывания. Изменение скорости «бегущего огня» на лету. Разбор типичной ошибки — дребезг контактов и программные методы его подавления.

**СОДЕРЖАНИЕ**

1. Цель и оборудование
3. Структура проекта и основные файлы
4. Подготовка оборудования и программной среды
4.1 Подключение платы
4.2 Создание и подготовка проекта в VS Code
4.3 Проверка системных файлов проекта
5. Реализация, сборка и прошивка проекта
5.1 Настройка светодиодов и входа WAKEUP0
5.2 Настройка таймера TMR32
5.3 Реализация эффекта «Бегущий огонь»
5.4 Сборка проекта
5.5 Прошивка и запуск
5.6 Проверка и отладка
6. Практические задания
6.1 Общие правила для всех вариантов
6.2 Вариант 1. Трехскоростной режим
6.3 Вариант 2. Смена направления движения
6.4 Вариант 3. Обработка кнопки в прерывании
6.5 Вариант 4. Плавное изменение скорости при удержании кнопки
6.6 Вариант 5. Умная кнопка
7. Контрольные вопросы
8. Тестовые вопросы

## ЦЕЛЬ И ОБОРУДОВАНИЕ

**Цель:** изучить работу входа WAKEUP0 микроконтроллера К1921ВГ015 и получить практические навыки управления цифровыми выходами при реализации эффекта «бегущий огонь» с изменением скорости в зависимости от состояния входного сигнала.

**Оборудование:** отладочная плата с микроконтроллером К1921ВГ015; персональный компьютер с доступом в интернет; USB Type-C для подключения платы к компьютеру; среда разработки Visual Studio Code.

## СТРУКТУРА ПРОЕКТА И ОСНОВНЫЕ ФАЙЛЫ

В качестве основы лабораторной работы используется демонстрационный проект run_leds из состава SDK. Пользовательская часть проекта дополняется кодом обработки WAKEUP0, логикой управления таймером TMR32 и программой индикации.

`main.c` – основной пользовательский файл, в котором размещается логика лабораторной работы.

`makefile` – сценарий автоматической сборки; обеспечивает компиляцию исходных файлов, компоновку исполняемого файла .elf и формирование файлов .hex и .dump.

`config.json` – конфигурационный файл проекта для среды разработки.

`niiet_aspect.json` – файл конфигурации инструментов NIIET Aspect; при необходимости в нем уточняется использование линковочного сценария для сборки проекта во Flash-память.

Системные файлы проекта изменять не требуется. К ним относятся:

`system_k1921vg015.c` и `system_k1921vg015.h` – файлы базовой инициализации системы и обновления значения `SystemCoreClock`.

`sys_init.c` – дополнительная системная инициализация перед запуском `main()`.

`startup_k1921vg015.S` – startup-файл с таблицей векторов прерываний, обработчиком сброса и переходом к `main()`.

`retarget.c` и `printf.c` – файлы перенаправления стандартного ввода-вывода и форматированного вывода.

`plic.c` – модуль программного контроллера прерываний PLIC.

Структура каталогов проекта включает:

`.vscode/` – настройки среды разработки, в том числе `launch.json`, `tasks.json` и settings.json.

`build/` – каталог автоматически создаваемых результатов сборки: объектных файлов, .elf, .hex и .dump.

`device/Include/` – заголовочные файлы микроконтроллера, определяющие регистры и структуры доступа к периферии.

`device/Ldscripts/` – линковочные сценарии размещения секций программы в памяти Flash и RAM.

`device/Source/` – исходные системные файлы, используемые при сборке проекта.

Корректность startup-файла, системных файлов и линковочного сценария определяет запуск программы, доступность обработчиков прерываний и правильность расчета периодов таймера. Поэтому перед началом работы необходимо убедиться, что системная часть проекта соответствует рабочему проекту лабораторной работы №1.

## ПОДГОТОВКА ОБОРУДОВАНИЯ И ПРОГРАММНОЙ СРЕДЫ

### 4.1 Подключение платы

Для выполнения лабораторной работы требуется подготовить отладочную плату К1921ВГ015, USB Type-C и рабочую папку проекта. Плата подключается к персональному компьютеру по USB; после подключения следует убедиться в наличии питания и корректном определении устройства системой.

Настройка перемычек и драйверов выполняется по той же схеме, что и в лабораторной работе №1. При использовании встроенного отладчика рекомендуется применять тот же набор параметров, что и для ранее проверенного проекта.

### 4.2 Создание и подготовка проекта в VS Code

Новый проект создается на основе демонстрационного примера `run_leds` из состава SDK. В окне импорта примеров в качестве интерфейса отладки выбирается `onboard_ftdi`.

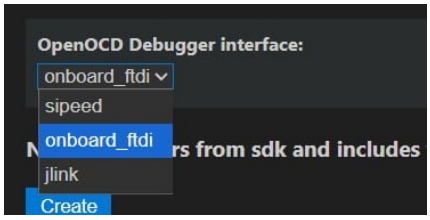



Рисунок 2.4. Окно импорта примеров

Допускается также создание отдельной папки Lab2 на базе рабочего проекта лабораторной работы №1 при условии сохранения корректной структуры каталогов, системных файлов и параметров сборки.

После создания проекта в файле `main.c` подготавливается пользовательская часть программы, в которой реализуются обработка WAKEUP0, управление таймером TMR32 и алгоритм светодиодной индикации.

### 4.3 Проверка системных файлов проекта

Перед реализацией логики работы необходимо проверить наличие в проекте системных файлов system_k1921vg015.c/.h, startup-файла и линковочного сценария. Параметры сборки должны содержать корректные include-пути, подключение системных модулей и целевую архитектуру, совпадающую с проектом лабораторной работы №1.

Особое значение имеют вызовы функций `SystemInit()` и `SystemCoreClockUpdate()`. Они обеспечивают корректную инициализацию системы и актуальное значение переменной `SystemCoreClock`, по которой рассчитываются периоды `slow_period` и `fast_period`.

## РЕАЛИЗАЦИЯ, СБОРКА И ПРОШИВКА ПРОЕКТА

### 5.1 Настройка светодиодов и входа WAKEUP0

На этапе инициализации необходимо включить тактирование порта GPIO, разрешить соответствующие линии в режим выхода через `OUTENSET` и установить начальное состояние светодиодов с учетом фактической логики их подключения.

Для обработки WAKEUP0 требуется задать полярность входа, разрешить его использование, сбросить флаг WAKE0 перед началом работы и предусмотреть небольшую задержку стабилизации после включения входа.

При реализации рекомендуется проверить корректность программных масок `LED0_MSK`, `LED7_MSK` и `LEDS_MSK` относительно используемой платы.

### 5.2 Настройка таймера TMR32

Тактирование TMR32 включается через блок RCU. Период переключения задается записью значения compare-регистра по формуле `CAPCOM[0].VAL = period − 1`. Далее настраиваются режим таймера, разрешение прерывания, регистрация обработчика в PLIC, приоритет источника и глобальное разрешение прерываний.

Периоды `slow_period` и `fast_period` рассчитываются по значению `SystemCoreClock`. При необходимости допускается ввод промежуточных режимов скорости.

### 5.3 Реализация эффекта «Бегущий огонь»

Для реализации эффекта целесообразно использовать глобальную переменную `volatile uint32_t led_shift`, содержащую маску активного светодиода. В обработчике прерывания таймера выполняются обновление состояния светодиодов, сдвиг маски и ее возврат к начальному положению после достижения границы диапазона.

После обработки события таймера необходимо сбросить флаг прерывания. Изменение скорости по сигналу WAKEUP0 выполняется в основном цикле или в обработчике соответствующего прерывания – в зависимости от варианта индивидуального задания.

### 5.4 Сборка проекта

Сборка проекта выполняется средствами VS Code или командой `make` во встроенном терминале. По завершении без ошибок в каталоге build должны быть сформированы исполняемый файл .elf, файл прошивки .hex и файл дизассемблированного листинга .dump.

При возникновении ошибок следует проверить include-пути, подключение системных файлов, корректность линковочного сценария и написание макросов, связанных с масками светодиодов и именами регистров.

### 5.5 Прошивка и запуск

Загрузка программы в микроконтроллер выполняется тем же способом, что и в лабораторной работе №1, через JTAG/USB-интерфейс и инструменты SDK. После прошивки, при необходимости, производится перезапуск платы и наблюдение за работой индикации.

В штатном режиме должен воспроизводиться эффект «бегущий огонь», а нажатие кнопки WAKEUP0 должно изменять скорость либо иной параметр, установленный вариантом задания.

### 5.6 Проверка и отладка

Если эффект не воспроизводится, необходимо проверить тактирование GPIO, настройку линий на вывод, правильность масок светодиодов и запуск таймера. При отсутствии реакции на кнопку следует проверить разрешение WAKEUP0, полярность входа, корректность сброса флага WAKE0 и соответствие аппаратного подключения линии WAKE0 конкретной плате.

Для локализации неисправности допускается временно использовать дополнительную светодиодную индикацию при поступлении события WAKE0 либо вывод диагностических сообщений через UART.

## ПРАКТИЧЕСКИЕ ЗАДАНИЯ

### 6.1 Общие правила для всех вариантов

Перед выполнением индивидуального варианта должен быть реализован и проверен базовый вариант лабораторной работы: эффект «бегущий огонь» воспроизводится устойчиво, а по событию WAKEUP0 изменяется скорость.

Изменения рекомендуется вносить только в пользовательскую часть проекта, как правило в `main.c` и, при необходимости, в дополнительный пользовательский модуль. Системные файлы, startup-файл и линковочный сценарий должны остаться без изменений.

Во всех вариантах требуется корректная обработка события WAKE0: после обнаружения события флаг должен быть своевременно сброшен. Переменные совместного доступа между прерыванием и основным кодом следует объявлять с квалификатором `volatile`.

Логика переключения скорости реализуется изменением периода таймера TMR32. Основной алгоритм «бегущего огня» рекомендуется сохранять в обработчике таймера; вариант задания определяет способ выбора режима скорости, направления и места обработки кнопки.

### 6.2 Вариант 1. Трехскоростной режим

Краткое описание: реализуется циклическое переключение между тремя скоростями работы эффекта.

1. Предусмотреть переменную режима `speed_mode` со значениями 0, 1 и 2.

2. Сформировать набор периодов для медленного, среднего и быстрого режимов.

3. Обеспечить переход к следующему режиму при каждом событии WAKEUP0.

4. Стартовым режимом считать медленный.

**Ожидаемый результат:** последовательность переключения скоростей имеет вид «медленно → средне → быстро → медленно».

### 6.3 Вариант 2. Смена направления движения

Краткое описание: кроме скорости необходимо реализовать изменение направления движения «бегущего огня».

1. Организовать цикл из четырех режимов: вперед быстро, вперед медленно, назад быстро, назад медленно.

2. Ввести переменную режима и признак направления сдвига.

3. В обработчике таймера предусмотреть сдвиг маски как влево, так и вправо.

4. Возврат маски после достижения границы диапазона выполнить для обоих направлений.

**Ожидаемый результат:** после четырех последовательных нажатий завершается полный цикл режимов, при этом направление движения индикации изменяется.

### 6.4 Вариант 3. Обработка кнопки в прерывании

Краткое описание: обработка WAKEUP0 переносится из основного цикла в обработчик соответствующего прерывания.

1. Определить по startup-файлу и заголовочным файлам SDK фактическое имя источника прерывания и имя обработчика.

2. Разрешить прерывание по событию WAKEUP0 и обеспечить сброс его флага.

3. Организовать передачу события в основной код через флаговую переменную либо выполнить переключение режима непосредственно в обработчике.

4. Основной цикл программы свести к минимальной служебной функции.

**Ожидаемый результат:** изменение режима выполняется по аппаратному прерыванию, а основной цикл не содержит опроса кнопки.

### 6.5 Вариант 4. Плавное изменение скорости при удержании кнопки

Краткое описание: скорость эффекта должна изменяться не только по событию, но и по уровню сигнала на входе WAKEUP0.

1. Реализовать чтение текущего уровня входа WAKE0.

2. При удержании кнопки организовать пошаговое ускорение эффекта до заданного предела.

3. После отпускания кнопки вернуть период к базовому значению.

4. При выборе параметров шага и паузы обеспечить визуально различимое изменение скорости.

**Ожидаемый результат:** при удержании кнопки наблюдается плавное ускорение, после отпускания восстанавливается исходный режим.

### 6.6 Вариант 5. Умная кнопка

Краткое описание: одной кнопкой необходимо различать короткое и длинное нажатие и связывать их с различными действиями.

1. Организовать измерение длительности нажатия по счетчику RTC либо по программному счетчику тиков таймера.

2. Короткое нажатие связать с переключением скорости.

3. Длинное нажатие связать с изменением направления движения.

4. Предусмотреть *антидребезг* и корректную обработку нажатия и отпускания.

**Ожидаемый результат:** кнопка различает два типа воздействия: короткое нажатие изменяет скорость, длительное – направление движения эффекта.

## КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Для чего предназначен вход WAKEUP0 в микроконтроллере и какие задачи он решает?

2. Чем отличается обработка уровня сигнала на входе от обработки события WAKE0?

3. В каком регистре фиксируется событие WAKEUP0 и каким образом выполняется сброс флага?

4. Что означает настройка полярности «активный низкий уровень» для WAKEUP0?

5. Что такое дребезг кнопки и как он проявляется в программе?

6. Какие способы программного подавления дребезга могут быть применены в данной работе?

7. Для чего используется таймер TMR32 в данной лабораторной работе?

8. Как формируется период таймера и что означает запись `CAPCOM[0].VAL = period - 1`?

9. Почему перед расчетом периодов таймера необходимо вызывать `SystemInit()` и `SystemCoreClockUpdate()`?

10. Почему реализацию эффекта «бегущий огонь» целесообразно выполнять в обработчике прерывания таймера?

11. Что представляет собой битовая маска `led_shift` и почему для управления индикацией используются операции сдвига?

12. Чем отличается управление светодиодами через `DATAOUTSET/CLR` от управления через `DATAOUTTGL`?

13. Почему в обработчике `TMR32_IRQHandler()` необходимо сбрасывать флаг прерывания таймера?

14. Как реализуется цикличность «бегущего огня» при переходе от последнего светодиода к первому?

15. В чем различие между опросом кнопки в цикле `while(1)` и обработкой кнопки через прерывание?

16. Как организуется трехскоростной режим с использованием `speed_mode` и массива периодов?

17. Какие изменения в алгоритме требуются для реализации обратного направления движения?

18. Каким образом можно различить короткое и длинное нажатие в варианте «Умная кнопка»?

## ТЕСТОВЫЕ ВОПРОСЫ

Выберите один правильный ответ.

**1. Для чего в данной лабораторной работе используется вход WAKEUP0?**

А) Для формирования сигнала управления программой и изменения режима работы индикации.

Б) Для записи программы во Flash-память микроконтроллера.

В) Для обмена данными по интерфейсу UART.

**2. В каком поле регистра фиксируется событие WAKEUP0?**

А) `RTC_HISTORY.WAKE0`.

Б) `DATAOUTSET`.

В) `CAPCOM[0].VAL`.

**3. Каким действием обычно выполняется сброс флага WAKE0?**

А) Записью логической единицы в соответствующее поле сброса.

Б) Установкой нуля в регистр `OUTENSET`.

В) Перезапуском таймера TMR32.

**4. Для чего вызывается функция `SystemCoreClockUpdate()`?**

А) Для пересчета текущей частоты ядра, используемой при вычислении периодов таймера.

Б) Для включения светодиодов на порту GPIOA.

В) Для очистки флага прерывания WAKE0.

**5. Что задает запись `CAPCOM[0].VAL = period - 1`?**

А) Порог сравнения таймера, определяющий период его срабатывания.

Б) Номер активного светодиода.

В) Приоритет источника прерывания в PLIC.

**6. Что представляет собой переменная `led_shift`?**

А) Битовую маску активного светодиода.

Б) Список обработчиков прерываний.

В) Значение системной частоты `SystemCoreClock`.

**7. Какой регистр GPIO инвертирует состояние выбранных выводов?**

А) `DATAOUTTGL`.

Б) `OUTENSET`.

В) `RTC_HISTORY`.

**8. Для чего в программе предусматривается антидребезг кнопки?**

А) Для предотвращения нескольких ложных срабатываний при одном нажатии.

Б) Для ускорения работы таймера TMR32.

В) Для автоматического выбора направления движения.

**9. Чем обработка кнопки через прерывание отличается от опроса в основном цикле?**

А) При прерывании реакция выполняется по событию без постоянной проверки кнопки в цикле.

Б) При прерывании невозможно менять период таймера.

В) Между этими подходами отсутствуют различия.

**10. Каким действием в базовом варианте изменяется скорость «бегущего огня»?**

А) Изменением периода таймера TMR32 после события WAKEUP0.
Б) Изменением таблицы векторов прерываний startup-файла.
В) Переносом программы из Flash-памяти в RAM.

**11. Какой регистр переводит выбранные линии GPIO в режим выхода?**

А) `OUTENSET`.
Б) `DATAOUTCLR`.
В) `RTC_HISTORY`.

**12. Для чего используется регистр `DATAOUTSET`?**

А) Для установки логической единицы на выбранных линиях вывода.
Б) Для переключения режима таймера.
В) Для чтения состояния кнопки WAKEUP0.

**13. Что означает инверсная логика подключения светодиода?**

А) Светодиод включается при подаче логического нуля.
Б) Светодиод всегда включен независимо от состояния вывода.
В) Светодиод может работать только в режиме прерывания.

**14. Для чего в проекте формируются файлы .elf, .hex и .dump?**

А) Для получения исполняемого файла, файла прошивки и листинга программы.
Б) Для настройки кнопки WAKEUP0.
В) Для задания направления движения светодиодов.

**15. Какой каталог проекта обычно содержит результаты сборки?**

А) build/.
Б) device/Include/.
В) .vscode/.

**16. Для чего в переменных совместного доступа между прерыванием и основным кодом используется квалификатор `volatile`?**

А) Чтобы компилятор не оптимизировал обращения к переменной, значение которой может изменяться асинхронно.
Б) Чтобы увеличить частоту работы таймера.
В) Чтобы автоматически подавлять дребезг кнопки.

**17. Для чего в лабораторной работе используется контроллер прерываний PLIC?**

А) Для регистрации и обработки источников прерываний микроконтроллера.
Б) Для записи программы во Flash-память.
В) Для формирования битовой маски светодиодов.

**18. Что должно происходить в штатном режиме после прошивки программы в микроконтроллер?**

А) Должен воспроизводиться эффект «бегущий огонь», а кнопка WAKEUP0 должна изменять режим его работы.
Б) Должны автоматически очищаться все системные файлы проекта.
В) Должен запускаться только режим отладки без световой индикации.

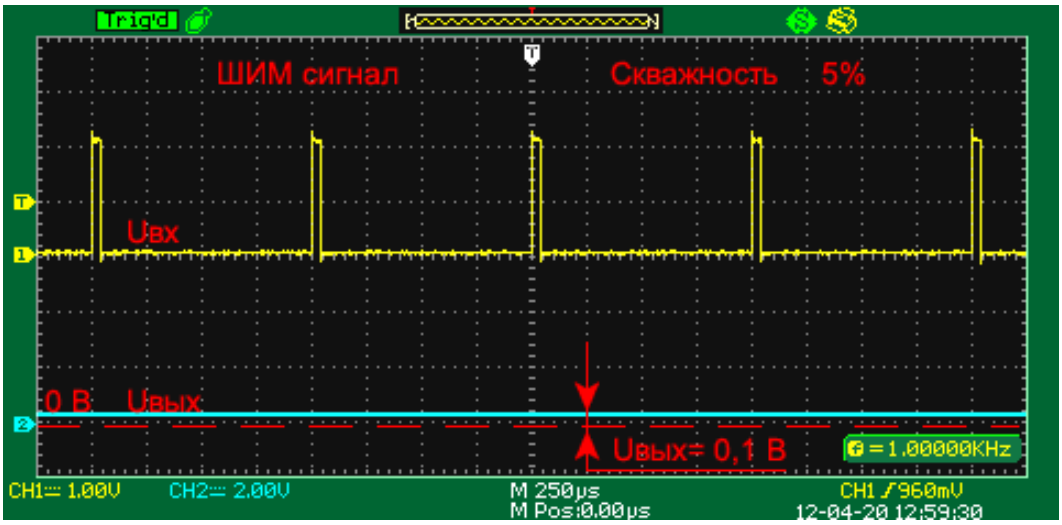
**19. Какое условие необходимо для корректного расчета периода таймера?**

А) Актуальное значение переменной `SystemCoreClock`.
Б) Наличие только одного светодиода на плате.
В) Отключение всех прерываний в системе.

**20. Что является ожидаемым результатом варианта «Умная кнопка»?**

А) Короткое нажатие изменяет скорость, а длинное – направление движения эффекта.
Б) Любое нажатие полностью останавливает выполнение программы.
В) Кнопка используется только для перезапуска таймера TMR32.

# Часть II. Преобразование данных: Аналоговая периферия и генераторы



# Глава 3. Генерация аналоговых сигналов: ШИМ и силовая электроника

*Широтно-импульсная модуляция* (*PWM — Pulse Width Modulation*) является основным способом управления мощностью в современной электронике. Вместо того чтобы подавать на нагрузку постоянное напряжение или использовать переменный резистор (который выделяет излишки энергии в виде тепла), микроконтроллер подает последовательность импульсов высокой частоты.

**Тема 3. Широтно-импульсная модуляция и методы ее исследования.**

Широтно-импульсная модуляция, или ШИМ, является одним из основных способов управления нагрузкой в микроконтроллерных системах. Она позволяет изменять среднее значение напряжения или мощности, подаваемой на устройство, без формирования настоящего аналогового сигнала. Это особенно важно для микроконтроллеров, так как их цифровые выводы обычно могут находиться только в двух состояниях: логический ноль и логическая единица.

Принцип ШИМ основан на быстром переключении выходного сигнала между низким и высоким уровнем. При этом период сигнала остается постоянным, а изменяется длительность активного состояния. Если высокий уровень занимает небольшую часть периода, средняя мощность на нагрузке будет низкой. Если высокий уровень занимает большую часть периода, средняя мощность увеличивается. За счет этого можно управлять яркостью светодиода, скоростью двигателя, нагревом элемента или другим параметром нагрузки.

Главное преимущество ШИМ заключается в том, что управление выполняется цифровым способом, но результат воспринимается как плавное изменение мощности. Например, если цифровой выход микроконтроллера может выдавать только 0 В или 3,3 В, то напрямую получить промежуточное напряжение он не может. Однако если сигнал половину времени находится в состоянии 3,3 В, а половину времени – в состоянии 0 В, среднее значение будет примерно соответствовать половине максимального уровня.

ШИМ широко применяется во встраиваемых системах. С его помощью регулируют яркость светодиодов, скорость вращения электродвигателей, мощность нагревателей, работу вентиляторов, звуковые сигналы и сервоприводы. В учебных лабораторных работах ШИМ удобно изучать на светодиодах, потому что изменение коэффициента заполнения сразу заметно визуально: чем больше активная часть периода, тем выше воспринимаемая яркость.

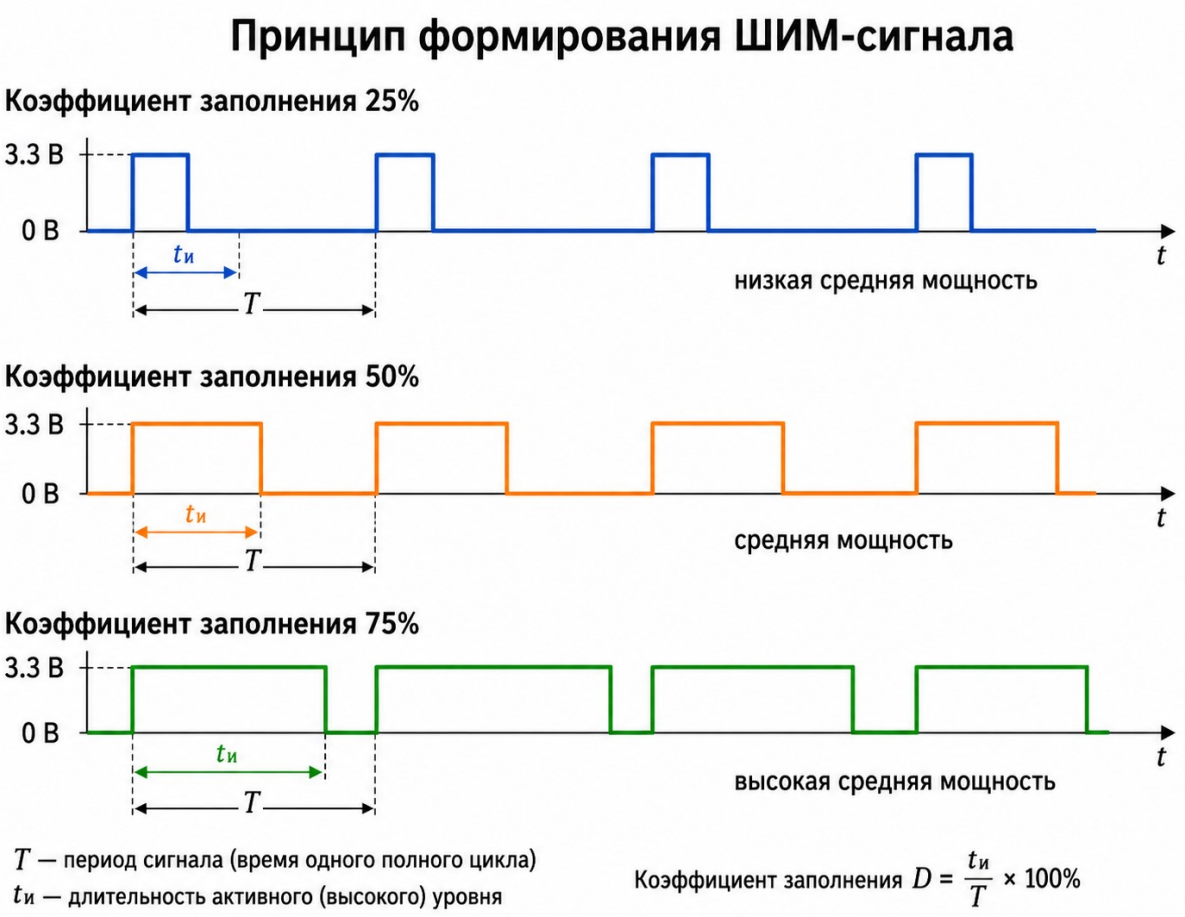



Рисунок 3.1. Принцип формирования ШИМ-сигнала при разных коэффициентах заполнения

**Основные параметры ШИМ-сигнала.**

ШИМ-сигнал описывается несколькими основными параметрами: периодом, частотой, длительностью активного уровня и коэффициентом заполнения. Период показывает время одного полного цикла сигнала. Частота показывает, сколько таких циклов происходит за одну секунду. Длительность активного уровня показывает, сколько времени в течение периода выход находится во включенном состоянии.

Коэффициент заполнения показывает, какую часть периода сигнал находится в активном состоянии. Например, при заполнении 25% высокий уровень занимает четверть периода, при 50% – половину периода, а при 75% – три четверти периода. Именно коэффициент заполнения определяет среднюю мощность, передаваемую нагрузке.

Важно не путать коэффициент заполнения и скважность. В учебной практике чаще используют именно коэффициент заполнения, потому что он напрямую показывает долю активного состояния сигнала. Чем больше коэффициент заполнения, тем больше среднее значение напряжения или мощности на нагрузке.

Таблица 3.1

| Параметр | Обозначение | Смысл |
|---|---|---|
| Период | T | Время одного полного цикла сигнала |
| Частота | f | Количество периодов сигнала за одну секунду |
| Длительность активного уровня | $t_a$ | Время, в течение которого сигнал находится в высоком уровне |
| Коэффициент заполнения | D | Доля периода, занятая активным уровнем |
| Амплитуда | U | Разность между низким и высоким уровнем сигнала |
| Среднее значение | $U_{ср}$ | Значение, воспринимаемое нагрузкой при быстром переключении |

Основные параметры ШИМ-сигнала

**Аппаратный и программный ШИМ.**

ШИМ может быть реализован программным или аппаратным способом. Программный ШИМ формируется непосредственно кодом программы: микроконтроллер включает выход, выполняет задержку, выключает выход и снова выполняет задержку. Такой способ прост для понимания, но имеет существенные ограничения. Он зависит от загрузки процессора, точность задержек может снижаться, а выполнение других задач становится сложнее.

Аппаратный ШИМ формируется специальным периферийным модулем микроконтроллера, чаще всего таймером. Таймер самостоятельно отсчитывает такты, сравнивает текущее значение счетчика с заданными регистрами и формирует выходной сигнал. Процессор при этом не занят постоянным переключением вывода и может выполнять другие операции.

Для практических задач аппаратный ШИМ является более правильным решением. Он обеспечивает стабильную частоту, точное заполнение и меньшую нагрузку на процессор. Программный ШИМ подходит в основном для простых демонстраций, когда нужно показать сам принцип включения и выключения сигнала.

Таблица 3.2 Сравнение аппаратного и программного ШИМ

| Критерий | Аппаратный ШИМ | Программный ШИМ |
|---|---|---|
| Способ формирования | Таймером микроконтроллера | Командами программы и задержками |
| Точность | Высокая | Ниже, зависит от загрузки процессора |
| Нагрузка на CPU | Минимальная | Высокая |
| Стабильность частоты | Высокая | Может изменяться |
| Критерий | Аппаратный ШИМ | Программный ШИМ |
| Использование в реальных задачах | Предпочтительно | Ограниченно |
| Простота понимания | Требует знания таймеров | Проще для начального объяснения |

**Формирование ШИМ с помощью таймера.**

В микроконтроллерных системах ШИМ обычно формируется с помощью *таймера*. Таймер считает такты от нуля до заданного значения периода. Когда счетчик достигает определенного значения сравнения, уровень выходного сигнала изменяется. После завершения периода счетчик начинается заново, и процесс повторяется.

Период ШИМ задается значением регистра периода, а коэффициент заполнения – значением регистра сравнения. Если значение сравнения составляет четверть периода, заполнение будет примерно 25%. Если половину периода – 50%. Если три четверти периода – 75%. Таким образом, изменение значения сравнения позволяет управлять средней мощностью на нагрузке.

В микроконтроллере К1921ВГ015 для формирования ШИМ может использоваться таймер TMR1. В учебной задаче ШИМ формируется на двух каналах, например на выводах PA.8 и PA.9. Это позволяет одновременно управлять двумя сигналами с разными коэффициентами заполнения. В базовом варианте один канал может иметь заполнение 25%, а второй – 75%, что удобно для сравнения яркости двух светодиодов.

Отдельно может использоваться таймер TMR32. Он не формирует сам ШИМ-сигнал, а задает моменты, когда программа должна изменить параметры ШИМ: увеличить или уменьшить заполнение, переключить частоту, изменить режим работы или выполнить другой

алгоритм. Такой подход позволяет разделить формирование сигнала и управление его параметрами.

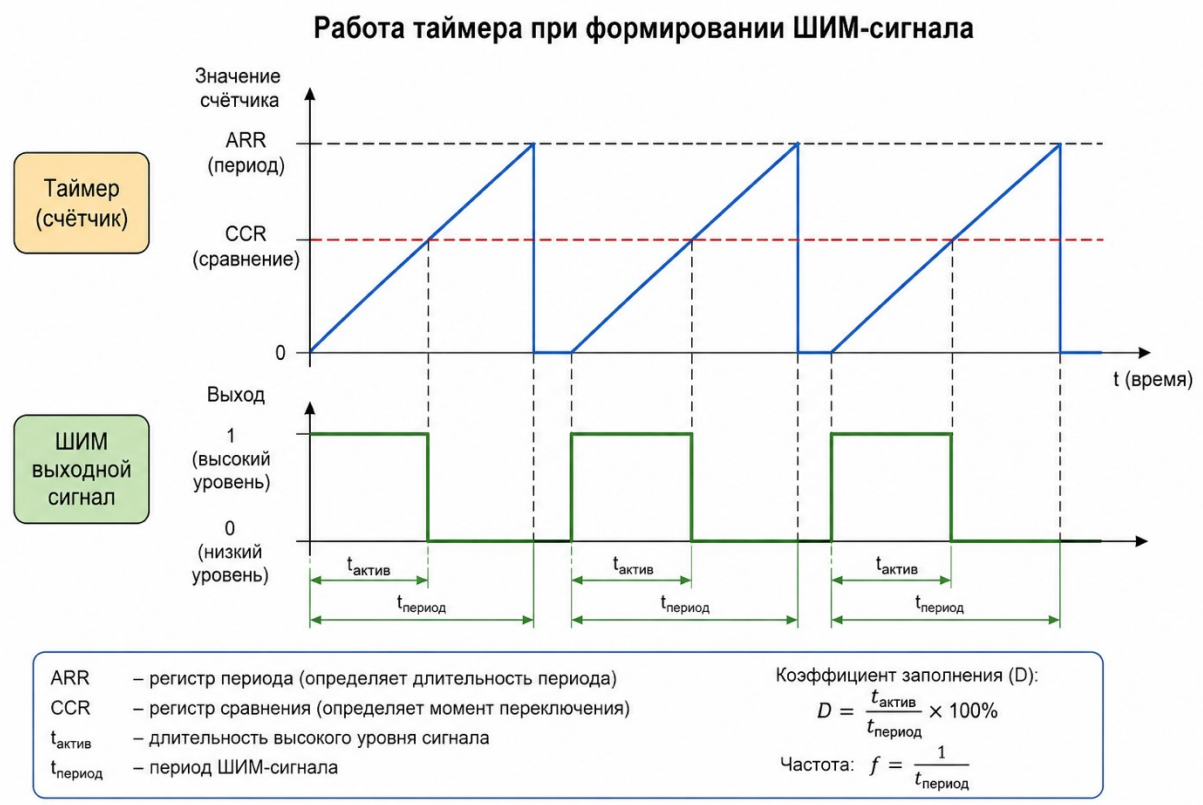



Рисунок 3.2. Работа таймера при формировании ШИМ-сигнала

**Исследование ШИМ с помощью осциллографа.**

Для анализа ШИМ-сигнала используется *осциллограф*. Он позволяет увидеть реальную форму сигнала, измерить период, частоту, амплитуду и коэффициент заполнения. В отличие от светодиода, который показывает только усредненную яркость, осциллограф отображает саму последовательность прямоугольных импульсов.

При подключении осциллографа важно правильно выбрать точки измерения. Сигнальный щуп подключается к выводу, на котором формируется ШИМ, например PA.8 или PA.9. Общий провод щупа обязательно подключается к GND платы. Если земля подключена неправильно или не подключена вообще, сигнал может быть нестабильным, искаженным или «плавающим».

На экране осциллографа по горизонтальной оси отображается время, а по вертикальной – напряжение. Для ШИМ с частотой 1 кГц период составляет около 1 мс, поэтому временную развертку нужно подобрать так, чтобы на экране был виден хотя бы один

полный период. По длительности высокого уровня и общему периоду можно вычислить коэффициент заполнения.

При исследовании двухканального ШИМ удобно подключить первый канал осциллографа к PA.8, второй канал – к PA.9, а общий провод – к GND платы. В этом случае можно сравнить два сигнала одновременно. Например, если на одном канале заполнение 25%, а на другом 75%, на экране будет видно, что у второго сигнала высокий уровень длится значительно дольше.

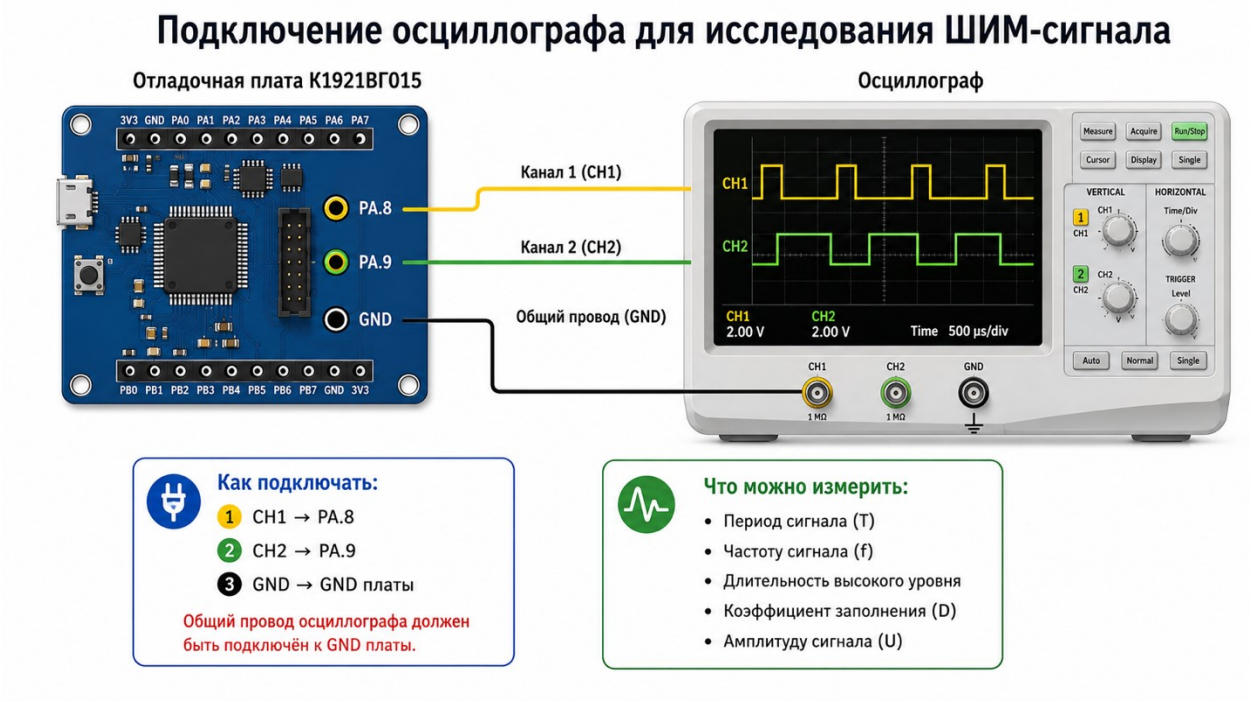



Рисунок 3.3. Подключение осциллографа для исследования ШИМ-сигнала

**Типичные ошибки при исследовании ШИМ.**

При работе с ШИМ и осциллографом часто возникают ошибки, связанные не с программой, а с измерением. Одна из самых распространенных ошибок – отсутствие общего провода между осциллографом и платой. В этом случае измеряемый сигнал может выглядеть нестабильным или не соответствовать реальной форме.

Также возможна ошибка подключения щупа не к тому выводу. Если программа формирует ШИМ на PA.8 и PA.9, а щуп подключен к другому контакту, сигнал может отсутствовать или иметь другую форму. Поэтому перед измерением нужно проверить, какие выводы используются в программе и где они расположены на разъемах платы.

Еще одна частая проблема – неправильная настройка временной развертки. Если масштаб выбран слишком крупным, отдельные импульсы будут сливаться. Если слишком

мелким, на экране может быть виден только небольшой фрагмент сигнала. Для анализа ШИМ нужно подобрать масштаб так, чтобы был виден полный период и можно было измерить длительность активного уровня.

Ошибки могут возникать и при интерпретации параметров. Например, студент может перепутать частоту и период или принять длительность активного уровня за весь период. Поэтому при анализе ШИМ важно последовательно определять: сначала полный период сигнала, затем длительность высокого уровня, после этого коэффициент заполнения.

**Таймер TMR1 и режим ШИМ**

В работе используется 16-разрядный таймер TMR1 микроконтроллера К1921ВГ015. Таймеры семейства TMR0-TMR2 входят в состав периферии микроконтроллера наряду с 32-разрядным таймером TMR32 и другими интерфейсными блоками, что подтверждается документацией быстрого старта для К1921ВГ015.

Режим ШИМ основан на периодическом счете таймера от нуля до заданного предельного значения. За один период на выходе формируется последовательность из двух уровней: активного и неактивного. Отношение длительности активного состояния ко всему периоду определяет заполнение сигнала, а обратная величина характеризует скважность.

Для данной работы используются два канала сравнения таймера TMR1. Согласно заданию, ШИМ должен формироваться одновременно на линиях PA.8 и PA.9. Базовая частота сигнала - 1 кГц, а начальные заполнения составляют 25% для канала PA.8 и 75% для канала PA.9.

При настройке ШИМ важно различать три уровня управления: выбор тактовой частоты таймера, задание периода счета и установка момента переключения выхода внутри периода. Именно поэтому лабораторная работа построена вокруг настройки периода через CAPCOM[0] и сравнения через CAPCOM[2] и CAPCOM[3].

**Регистры CAPCOM и расчет периода/заполнения**

В базовом варианте период ШИМ задается регистром CAPCOM[0].VAL. Если таймер считает от 0 до значения period_ticks, то количество тиков за период равно period_ticks + 1. При неизменном делителе частота ШИМ определяется отношением тактовой частоты таймера к числу тиков периода.

Для заданной частоты можно использовать соотношение: f_pwm = f_tmr / ((PSC + 1) * (CAPCOM[0].VAL + 1)), где f_tmr - частота источника таймера, PSC - коэффициент деления,

а CAPCOM[0].VAL - значение регистра периода. В учебной реализации удобно сначала выбрать делитель, а затем вычислить число тиков периода для 1 кГц.

Значение compare-регистра для заданного заполнения вычисляют пропорционально периоду. Практически удобно пользоваться формулой compare = round((CAPCOM[0].VAL+1) *duty/100), после чего контролировать, чтобы compare не выходило за допустимые пределы. Для двухканального режима это означает отдельный расчет CAPCOM[2].VAL и CAPCOM[3].VAL.

Если значение сравнения станет больше периода или отрицательным, выходной сигнал перестанет соответствовать требуемому заполнению. Поэтому во всех вариантах лабораторной работы обязательно контролируется диапазон 0...100% и корректность пересчета compare при каждом изменении заполнения или частоты.

**Формирование ШИМ на двух каналах**

Двухканальный режим в данной лабораторной работе нужен для одновременного управления двумя независимыми выходами. Это позволяет реализовать антифазное изменение яркости, фиксированное заполнение на одном канале, ступенчатую квантацию уровней, программный гейтинг и переключение частоты с сохранением процента заполнения.

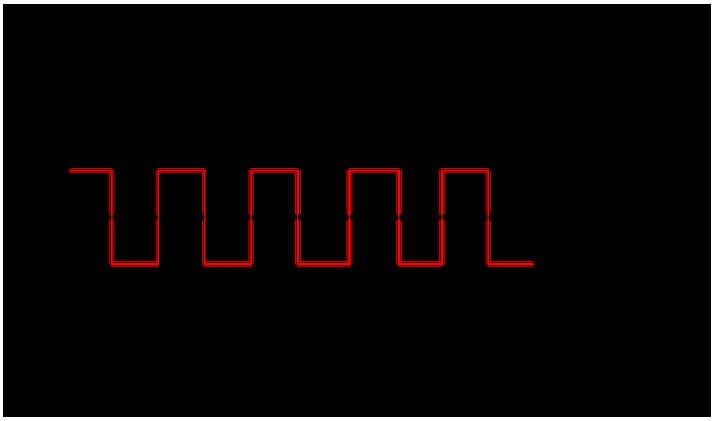


Рис. 3.4 - Цифровой сигнал (ШИМ)

С точки зрения наблюдения ШИМ светодиод не показывает прямоугольный сигнал как последовательность импульсов. Из-за инерционности глаза воспринимается усредненная яркость. Поэтому изменение duty-cycle визуально эквивалентно изменению яркости, а не простому переключению GPIO между логическими нулем и единицей.

Если на одном канале задать 25%, а на другом 75%, то при одинаковом подключении светодиодов яркость второго канала будет заметно выше. В вариантах с антифазным

управлением удобно использовать зависимость duty(PA.9) = 100% - duty(PA.8). Такой подход гарантирует синхронное поведение двух каналов и упрощает проверку результата.

Для ступенчатых сценариев полезно использовать таблицу уровней LUT (Look-Up Table). Тогда обработчик прерывания не вычисляет новое заполнение каждый раз заново, а лишь выбирает следующий элемент массива. Это делает алгоритм прозрачным, предсказуемым и удобным для отладки.

В контексте этой лабораторной работы LUT представляет собой массив заранее заданных значений заполнения, например {0, 25, 50, 75, 100}. При каждом срабатывании TMR32 программа увеличивает или уменьшает индекс таблицы и по выбранному элементу пересчитывает compare-регистры для TMR1. Если требуется "обратный ход", можно либо хранить полный маршрут 0 -> 25 -> 50 -> 75 -> 100 -> 75 -> 50 -> 25 -> ..., либо менять направление обхода отдельной переменной. Такой подход удобен тем, что набор допустимых уровней жестко ограничен, а значит студенту проще проверить правильность алгоритма и сопоставить наблюдаемую яркость с конкретным значением duty-cycle.

**Таймер TMR32 и изменение параметров ШИМ**

Таймер TMR32 в работе выполняет роль служебного таймера. Он не формирует сам ШИМ на PA.8 и PA.9, а задает моменты, когда программа должна изменить параметры TMR1: увеличить или уменьшить заполнение, переключить ступень таблицы, включить или выключить режим гейтинга либо сменить период ШИМ.

Последовательность работы типична для задач с прерыванием: настроить период TMR32, разрешить запрос прерывания, зарегистрировать обработчик TMR32_IRQHandler(), после срабатывания IRQ изменить нужные значения CAPCOM и очистить флаг события. Частота прерываний выбирается из задания варианта: например, 100 Гц для плавного изменения заполнения или 2 Гц для ступенчатой смены уровней.

При изменении только заполнения пересчитываются compare-регистры, а период остается неизменным. При изменении частоты необходимо сначала обновить CAPCOM[0].VAL, а затем заново вычислить CAPCOM[2].VAL и CAPCOM[3].VAL в тиках так, чтобы проценты 25% и 75% сохранились. Именно эта логика лежит в основе варианта с переключением 500 Гц -> 1 кГц -> 2 кГц -> 1 кГц.

Корректность алгоритма проверяется либо визуально по светодиодам, либо инструментально по осциллографу, либо программно через чтение регистров таймеров в режиме отладки. Во всех случаях требуется доказать не только факт запуска таймера, но и

соответствие периода, заполнения и закона изменения параметров условию конкретного варианта.

# Практическая работа №3

Независимое управление яркостью двух светодиодов через два канала `TMR1`. Подключение мощного светодиода через MOSFET-транзистор.

**СОДЕРЖАНИЕ**

1. Цель и оборудование
2. Структура проекта и основные файлы
3. Подготовка оборудования и программной среды
3.1 Подключение платы и конфигурация питания
3.2 Установка SDK и Toolchain
4. Создание, сборка и прошивка проекта
4.1 Создание проекта на основе примера
4.2 Подготовка файла `main.c`
4.3 Сборка проекта
4.4 Прошивка и запуск программы
4.5 Прошивка Flash
5. Практические задания
5.1 Общие правила для всех вариантов
5.2 Задание 3.1. Плавное изменение яркости
5.3 Задание 3.2. “Дыхание” одного канала
5.4 Задание 3.3. Ступенчатые уровни яркости
5.5 Задание 3.4. Модуляция включения ШИМ
5.6 Задание 3.5. Переключение частоты ШИМ
6. Контрольные вопросы
7. Тестовые вопросы

## ЦЕЛЬ И ОБОРУДОВАНИЕ

**Цель:** изучить формирование ШИМ-сигналов на двух каналах таймера `TMR1` микроконтроллера К1921ВГ015, научиться задавать период и заполнение через регистры `CAPCOM`, а также изменять параметры ШИМ в обработчике периодического прерывания таймера `TMR32`.

**Оборудование:** отладочная плата с микроконтроллером К1921ВГ015; персональный компьютер с Windows; USB-кабель для подключения платы; установленная среда Visual Studio Code с расширением NIIET Aspect; SDK, toolchain и OpenOCD; два светодиода с токоограничивающими резисторами 330-1000 Ом и соединительные провода; при наличии - осциллограф или логический анализатор для контроля частоты и заполнения сигналов.

## СТРУКТУРА ПРОЕКТА И ОСНОВНЫЕ ФАЙЛЫ

Основные файлы проекта можно увидеть на рисунке 1:

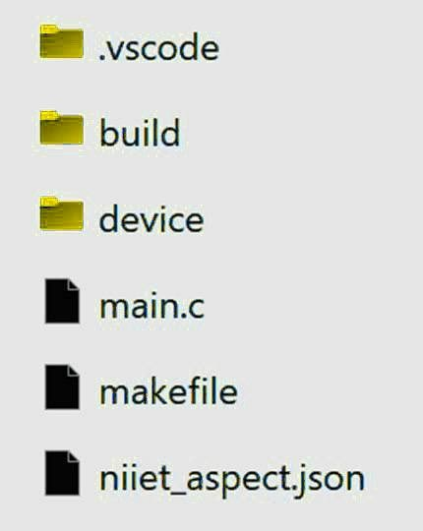



Рисунок 3.5. Файлы проекта

Таблица 3.3 Описание файлов проекта

| Категория | Файл | Назначение | Как работает |
|---|---|---|---|
| Основные файлы проекта | `main.c` | Ваш основной рабочий файл с кодом программы. | Пишете здесь всю логику для выполнения лабораторных работ. |
| Основные файлы проекта | `makefile` | Скрипт для автоматической сборки проекта. | При запуске команды `make` в терминале компилирует все `.c` файлы в `.o`, линкует их в `.elf` и формирует `.hex/.dump` для прошивки и отладки. |

| | | | |
|---|---|---|---|
| Основные файлы проекта | config.json | Конфигурация проекта для среды разработки. | Содержит настройки путей и параметров инструментария. |
| Основные файлы проекта | niiet_aspect.json | Специфичная конфигурация для инструментов NIIET Aspect. | Чтобы прошить плату во Flash, нужно заменить scriptFile с k1921vg015_ram.ld на k1921vg015_flash.ld. |
| Системные файлы (не трогайте без необходимости) | system_k1921vg015.c | Базовая системная инициализация. | Инициализирует системную частоту и служебные параметры МК. |
| Системные файлы (не трогайте без необходимости) | startup_k1921vg015.S | Стартовый код. | Запускается до main(), настраивает стек и таблицу векторов. |
| Категория | Файл | Назначение | Как работает |
| Системные файлы (не трогайте без необходимости) | printf.c | Поддержка printf и перенаправления вывода. | Используется для сообщений в UART. |
| Системные файлы (не трогайте без необходимости) | plic.c | Контроллер прерываний PLIC. | Содержит функции назначения и разрешения обработчиков прерываний. |
| Системные файлы (не трогайте без необходимости) | tmr32.c / tmr32.h | Драйвер 32-битного таймера. | Используется для настройки периодических событий и пробуждения по таймеру. |
| Системные файлы (не трогайте без необходимости) | gpio.c / gpio.h | Драйвер GPIO. | Нужен для управления светодиодами и приема внешнего сигнала пробуждения. |

Таблица 3.4 Директории проекта

| Директория | Назначение | Содержит |
|---|---|---|
| .vscode/ | Настройки среды разработки VS Code. | launch.json - настройки отладчика; tasks.json - задачи для сборки; settings.json - параметры редактора. |
| build/ | Директория для сгенерированных файлов. | .o - результат компиляции; .elf - исполняемый файл; .hex - файл прошивки; .dump - файл для отладки. |
| device/Include/ | Заголовочные файлы (.h). | K1921VG015.h - основной файл с определениями регистров МК. |
| device/Ldscripts/ | Скрипты линковщика. | k1921vg015_flash.ld; k1921vg015_ram.ld; |

| | | |
|---|---|---|
| | | k1921vg015.common.lds. |
| device/Source/ | Исходные системные файлы. | Системная инициализация, startup-код и базовые драйверы. |

Описание директорий проекта

## ПОДГОТОВКА ОБОРУДОВАНИЯ И ПРОГРАММНОЙ СРЕДЫ

**Подключение платы и конфигурация питания**

1) Подключите плату К1921ВГ015 к компьютеру с помощью USB-кабеля. Порт USB Type-C отмечен на рисунке 3.7;




Рисунок 3.6. Перемычка DBG MAIN REG




Рисунок 3.7. Перемычка FT_RST

2) Установите перемычку USB 5V JACK в положение питания от USB;

3) Установите перемычку DBG MAIN REG (рис. 3.6) в положение питания микроконтроллера от программатора (рис. 3.7);

4) Проверьте, что перемычка FT_RST (рис. 3.7) установлена (сброс программатора снят);

5) Установка драйверов и настройка Visual Studio Code подробно описана в ПР1.

**Установка SDK и Toolchain**

1) После установки расширения откройте панель NIIET Aspect. Из этого меню можно загрузить SDK, компилятор и инструменты отладки;

2) Для установки компонентов через расширение используйте пункт Download toolchain и выберите, что требуется: SDK, компилятор или отладочные инструменты;

3) Если архивы устанавливаются вручную, откройте Settings (**Ctrl+**,), введите в поиске niiet-aspect и задайте пути к SDK, компилятору, make и OpenOCD;

4) Без установленного SDK расширение не сможет создать новый проект и открыть пример.

## СОЗДАНИЕ, СБОРКА И ПРОШИВКА ПРОЕКТА

**Создание нового проекта на основе примера**

Это обязательный шаг – нельзя просто открыть папку из архива Lab3.

1) Откройте VS Code;

2) В левой панели нажмите NIIET Aspect → Open Example;

3) Выберите пример: tmr_pwm (bare) *(*рис. 3.8);

4) В параметре OpenOCD Debugger interface выберите: onboard_ftdi – это включает встроенный отладчик платы;

5) Нажмите Create и дождитесь, пока проект откроется.

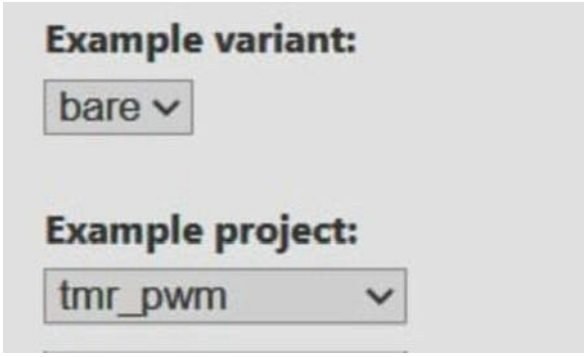



Рисунок 3.8. Панель Open Example

**Замена файла `main.c`:**

1) В VS Code откройте папку проекта;
2) Найдите файл: `main.c`;
3) Удалите его или переименуйте;
4) Откройте архив Lab3;
5) Скопируйте файл `main.c` из архива в папку проекта;
6) Сохраните файл.

Теперь ваш проект соответствует лабораторной работе.

**Сборка проекта**

1) Сборку проекта выполните через кнопку Compile project на панели NIIET Aspect или через соответствующую задачу VS Code;

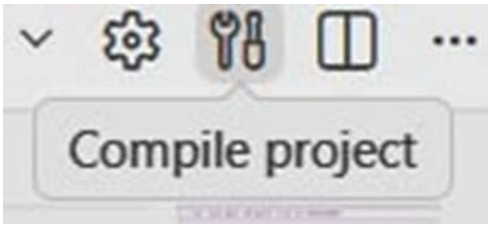



Рисунок 3.9. Компиляция проекта

2) После успешной компиляции в каталоге проекта создается исполняемый файл проекта с расширением .elf;
3) При необходимости также могут формироваться вспомогательные файлы сборки (.hex, .dump и др.);
4) Если сборка завершилась без ошибок, проект готов к прошивке и отладке;
5) Нажимаем на Build Project.

**Прошивка микроконтроллера**

1) Убедитесь, что для проекта выбрана отладочная конфигурация Debug NIIET trng;

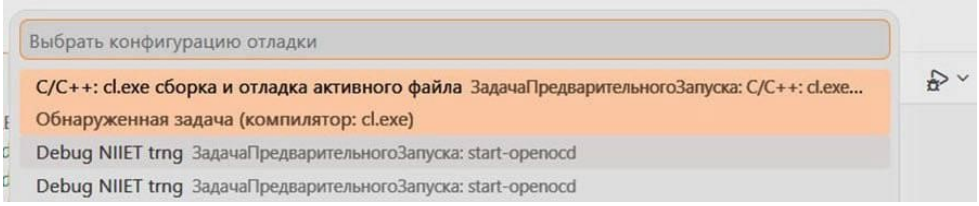



Рисунок 3.10. Debug

2) Для прошивки микроконтроллера запустите эту конфигурацию из списка отладки VS Code;

3) Дождитесь завершения загрузки программы в память микроконтроллера;

4) После запуска отладки можно использовать точки останова и пошаговое выполнение программы;

5) Нажать кнопку Reset.

**Прошивка Flash**

Необходимо изменить конфигурацию линковки/скрипта загрузки:

- В файле `niiet_aspect.json` переключить скрипт линковщика (или целевой режим) **с ram на flash**;
- Подробно этот шаг описан в **ПР №1** (раздел “ВАЖНО!!! … заменить ram на flash в `niiet_aspect.json`”).

После изменения:

1. пересобрать проект;
2. выполнить прошивку (через NIIET Aspect/OpenOCD по инструкции ПР №1).

## ПРАКТИЧЕСКИЕ ЗАДАНИЯ

**Общие правила для всех вариантов**

Объект: микроконтроллер К1921ВГ015, таймер `TMR1`, каналы `CAPCOM[2]` и `CAPCOM[3]`.

Выводы: `PA.8` и `PA.9` (альтернативная функция `TMR1`).

Таймер обслуживания: `TMR32` (используется для периодического изменения параметров без UART).

Базовая частота ШИМ: 1 кГц (если иное не указано в варианте).

Начальные заполнения: 25% (`PA.8`) и 75% (`PA.9`), если иное не указано.

Условие корректности: на `PA.8` и `PA.9` формируются ШИМ-сигналы; изменение параметров (скважность/частота/режим) происходит согласно варианту.

В файл `main.c` допускается добавлять только необходимую логику варианта: инициализацию дополнительных переменных, собственные функции выше `main()` и код обработки в основном цикле/IRQ.

При изменении параметров ШИМ необходимо пересчитывать и записывать значения `CAPCOM[2].VAL` и `CAPCOM[3].VAL` строго в соответствии с требуемым заполнением.

Нельзя допускать выхода заполнения за диапазон 0...100% и появления недопустимых значений compare >= period.

Пример вывода в UART показан на рисунке 3.11.

```
--- PWM state ---
CAPCOM0 (period VAL): 6249  (ticks=6250)
CAPCOM2 (PA8) VAL   : 1562  duty~25%
CAPCOM3 (PA9) VAL   : 4687  duty~75%
OUTMODE: 7
```

Рисунок 3.11. Пример вывод в UART

**Способы проверки работы**

В ходе выполнения работы формируются ШИМ-сигналы на выводах микроконтроллера. Для проверки корректности работы допускается использование светодиодов с токоограничивающими резисторами, осциллографа или логического анализатора, встроенных средств отладки и программной индикации изменения параметров в отладочном режиме.

**1. Проверка с использованием светодиодов**

Для визуальной проверки формирования ШИМ-сигнала выводы микроконтроллера подключаются к светодиодам: `PA.8 -> LED1 -> резистор 330-1000 Ом -> GND; PA.9 -> LED2 -> резистор -> GND`. При обратной полярности подключения логика изменения яркости может быть инвертирована. Ожидаемый результат: при базовой настройке PA.8 светится слабее, а PA.9 ярче; в изменяемых вариантах яркость меняется плавно или ступенчато.

**2. Проверка с использованием осциллографа (при его наличии)**

Для точной проверки параметров ШИМ рекомендуется использовать двухканальный осциллограф.

Перед началом измерений убедитесь, что плата подключена, программа прошита и ШИМ уже формируется. Если у пробников есть переключатель 1x/10x, рекомендуется использовать режим 10x и выставить такое же значение в настройках канала осциллографа.

Точки подключения: CH1 -> PA.8, CH2 -> PA.9, общий провод осциллографа -> GND платы. На отладочной плате линии PA.8 и PA.9 выведены на разъем X6 и соответствуют альтернативным функциям `TMR1_OUT2` и `TMR1_OUT3`.

Порядок настройки осциллографа:

1) Подключить оба щупа к каналам CH1 и CH2, а зажим "земля" каждого пробника подключить только к общему проводу платы GND.

2) Включить оба канала и выбрать режим DC coupling. Для первичной настройки можно использовать AutoSet/Autoscale, а затем подстроить параметры вручную.

3) Установить вертикальный масштаб примерно 1-2 В/дел. Подберите его так, чтобы сигнал занимал заметную часть экрана по высоте и не обрезался.

4) Установить горизонтальный масштаб 0,2-0,5 мс/дел. Для ШИМ 1 кГц период составляет примерно 1 мс, поэтому на экране должно помещаться 1-3 периода сигнала.

5) Настроить запуск (Trigger): источник CH1, тип Edge, фронт Rising, уровень запуска примерно посередине амплитуды сигнала. После этого изображение должно стать устойчивым и не "плыть" по экрану.

6) Включить автоматические измерения Frequency или Period и Duty Cycle для каждого канала. Если автоматические измерения недоступны, частоту можно определить по периоду по временной шкале, а заполнение - как отношение длительности высокого уровня к полному периоду.

Что должен увидеть студент в базовом варианте:

для PA.8: частота около 1 кГц, период около 1 мс, заполнение около 25%;

для PA.9: частота около 1 кГц, период около 1 мс, заполнение около 75%.

В модифицированных вариантах следует контролировать не только наличие импульсов, но и закон изменения параметров. Например, в варианте "перетекание яркости" заполнение CH1 должно плавно расти и уменьшаться, а заполнение CH2 - изменяться в противофазе. В варианте с переключением частоты должны наблюдаться режимы 500 Гц, 1 кГц и 2 кГц при сохранении заданных процентов заполнения.

Что зафиксировать в отчете: фотографию экрана осциллографа или скриншот, значения измеренных Frequency/Period и Duty Cycle для обоих каналов, а также краткий вывод о соответствии сигнала заданию.

Типовые ошибки при проверке: отсутствие общего провода GND; подключение "земли" щупа не к общему проводу платы; слишком крупный или слишком мелкий масштаб по времени; неправильный уровень триггера; измерение только одного канала без сравнения второго.

### 3. Проверка без внешнего оборудования

Корректность работы может быть подтверждена программно: таймер `TMR1` запущен в режиме счета от 0 до `CAPCOM[0].VAL`; значения `CAPCOM[2].VAL` и `CAPCOM[3].VAL` соответствуют расчетным заполнениям; при изменении параметров значения compare-регистров меняются согласно алгоритму. Для базового периода 6250 тиков примерные значения составляют `CAPCOM[2].VAL` ≈ 1562 (25%) и `CAPCOM[3].VAL` ≈ 4687 (75%).

### 4. Критерии правильного выполнения

Работа считается выполненной правильно, если `TMR1` настроен в режим ШИМ, период задается через `CAPCOM[0].VAL`, ШИМ-сигналы формируются на `PA.8` и `PA.9`, заполнение

каналов соответствует заданию, а в модифицированных вариантах параметры изменяются по требуемому закону и подтверждены одним из предложенных способов проверки.

**Задание 3.1. Плавное изменение яркости (антифазное изменение заполнения)**

Таблица 3.5

| № | Параметр | Значение |
|---|---|---|
| 1 | Назначение | Освоить изменение заполнения ШИМ в прерывании и синхронное управление двумя каналами в противофазе. |
| 2 | Задание | Реализовать плавное изменение заполнения:<br>PA.8: заполнение увеличивается 0% → 100% → 0% (циклически);<br>PA.9: заполнение вычисляется как 100% − duty(PA.8). |
| 3 | Исходные данные | Частота ШИМ: 1 кГц;<br>Частота прерываний TMR32: 50…200 Гц (рекомендуется 100 Гц);<br>Шаг изменения заполнения: 1…3% за одно прерывание. |
| 4 | Требования | Заполнение не выходит за 0…100%;<br>Изменение плавное: скачок не больше заданного шага. |
| 5 | Порядок выполнения | 1. Настроить TMR1 на PWM (2 канала, PA.8/PA.9);<br>2. Настроить TMR32 на 100 Гц (или выбранную частоту);<br>3. В TMR32_IRQHandler():<br>хранить текущий duty для PA.8 и направление изменения (вверх/вниз);<br>обновлять duty PA.8 на шаг;<br>вычислять duty PA.9 = 100 − duty(PA.8);<br>пересчитывать compare_ticks и записывать в CAPCOM[2].VAL и CAPCOM[3].VAL.<br>4. Проверить осциллографом, что при росте duty на PA.8 duty на PA.9 уменьшается. |
| 6 | Содержание отчета | Схема/описание алгоритма изменения заполнения; Формулы пересчета заполнения в значение регистра сравнения;<br>Временные диаграммы (ожидаемые) для двух каналов; Листинг программы (ключевые фрагменты). |

Описание задания 3.1.

**Задание 3.2. “Дыхание” одного канала, второй - фиксированный**

Таблица 3.6

| № | Параметр | Значение |
|---|---|---|
| 1 | Назначение | Отработать плавное изменение заполнения на одном канале при фиксированных параметрах другого. |
| 2 | Задание | PA.8: заполнение изменяется 0% → 100% → 0%;<br>PA.9: заполнение фиксированное 50% (или 30% по указанию преподавателя). |
| 3 | Исходные данные | Частота ШИМ: 1 кГц; TMR32: 50…200 Гц; Шаг заполнения: 1…2%. |
| 4 | Требования | PA.9 не изменяется во времени;<br>Изменение PA.8 циклическое и монотонное на каждом участке (рост/спад). |

| № | Параметр | Значение |
|---|---|---|
| 5 | Порядок выполнения | Реализовать PWM на двух каналах;<br>Установить PA.9 на постоянный compare_ticks (например, 50%);<br>В прерывании TMR32 изменять только CAPCOM[2].VAL (PA.8). |
| 6 | Содержание отчета | Описание закона изменения заполнения;<br>Таблица значений заполнения для нескольких шагов (пример);<br>Листинг обработчика прерывания. |

Описание задания 3.2.

**Задание 3.3. Ступенчатые уровни яркости (квантация заполнения)**

Таблица 3.7

| № | Параметр | Значение |
|---|---|---|
| 1 | Назначение | Освоить дискретное управление заполнением по таблице уровней (LUT). |
| 2 | Задание | Заполнение принимает только значения из набора {0, 25, 50, 75, 100}:<br>PA.8: 0 → 25 → 50 → 75 → 100 → 75 → 50 → 25 → 0 → … ;<br>PA.9: 100 → 75 → 50 → 25 → 0 → 25 → 50 → 75 → 100 → … |
| 3 | Исходные данные | Частота ШИМ: 1 кГц;<br>Частота смены уровня: 2…5 Гц (рекомендуется 2 Гц). |
| 4 | Требования | Заполнение принимает только значения из заданного набора {0, 25, 50, 75, 100};<br>Смена уровня должна происходить строго с заданной периодичностью. |
| 5 | Порядок выполнения | 1. Настроить TMR1 как в базовом варианте;<br>2. Настроить TMR32 как “такт переключения” (например, 2 Гц);<br>3. В прерывании TMR32:<br>хранить индекс в таблице уровней;<br>по индексу брать duty для PA.8;<br>для PA.9 использовать “зеркальный” уровень (например, 100 − duty(PA.8)) или отдельную таблицу;<br>записывать compare в CAPCOM[2]/CAPCOM[3].<br>4. Проверить, что уровни действительно ступенчатые, без промежуточных значений. |
| 6 | Содержание отчета | Таблица уровней и порядок обхода;<br>Алгоритм переключения индекса;<br>Временные диаграммы. |

Описание задания 3.3.

**Задание 3.4. Модуляция включения ШИМ (гейтинг, мигание ШИМ)**

Таблица 3.8

| № | Параметр | Значение |
|---|---|---|
| 1 | Назначение | Освоить включение/выключение ШИМ без отключения таймера и без изменения режима портов (управление только compare). |
| 2 | Задание | Режим “ШИМ-мигание”:<br>каждые 0,5 с ШИМ на обоих каналах “включается” и “выключается”;<br>включено: PA.8 = 25%, PA.9 = 75%;<br>выключено: заполнение 0% на обоих каналах (или 100% – по указанию преподавателя, в зависимости от полярности нагрузки). |
| 3 | Исходные данные | Частота ШИМ: 1 кГц;<br>Частота мигания: 1 Гц (период 1 с, 50% скважность мигания). |
| 4 | Требования | Не менять режимы GPIO и не останавливать `TMR1`;<br>Переключение режима выполнять только изменением регистров сравнения (`CAPCOMx.VAL`). |
| 5 | Порядок выполнения | Настроить PWM на двух каналах;<br>Настроить `TMR32` на 2 Гц (удобно для полупериода) или 1;<br>В прерывании попеременно устанавливать заполнение “рабочее” и “нулевое”. |
| 6 | Содержание отчета | Алгоритм переключения состояний.<br>Временная диаграмма: внешний “гейтинг” + внутренний ШИМ. |

Описание задания 3.4.

**Задание 3.5. Переключение частоты ШИМ при сохранении заполнения в процентах)**

Таблица 3.9

| № | Параметр | Значение |
|---|---|---|
| 1 | Назначение | Освоить изменение периода ШИМ и пересчет значений сравнения для сохранения заполнения в процентах. |
| 2 | Задание | Циклическое изменение частоты ШИМ:<br>500 Гц → 1 кГц → 2 кГц → 1 кГц → …<br>При этом заполнение сохраняется:<br>`PA.8` = 25% во всех режимах;<br>`PA.9` = 75% во всех режимах. |
| 3 | Исходные данные | Частота переключения режимов: 1 Гц (или 0,5 Гц);<br>Делитель `TMR1`: постоянный (например, /8 – конкретный выбор зависит от базовой частоты тактирования таймера). |
| 4 | Требования | При смене `CAPCOM[0].VAL` обязательно пересчитывать `CAPCOM[2].VAL` и `CAPCOM[3].VAL`;<br>Не допускать недопустимых значений (например compare ≥ period). |
| 5 | Порядок выполнения | Настроить PWM на 1 кГц (базовый режим);<br>Настроить `TMR32` на частоту переключения (например, 1 Гц);<br>В обработчике `TMR32` циклически менять период (`CAPCOM0`) и пересчитывать compare по процентам. |

| № | Параметр | Значение |
|---|---|---|
| 6 | Содержание отчета | Формулы для частоты и пересчета compare;<br>Таблица режимов: (freq, period_ticks, compare_ticks для 25% и 75%);<br>Алгоритм переключения режимов. |

Описание задания 3.5.

**КОНТРОЛЬНЫЕ ВОПРОСЫ**

1. Как связаны `CAPCOM[0].VAL` и период ШИМ?

2. Как вычисляется значение `CAPCOM[x].VAL` по заданному заполнению?

3. Почему нельзя допускать выход заполнения за диапазон 0...100%?

4. Чем отличается управление яркостью ШИМ от управления уровнем GPIO?

5. Как обеспечивается плавность изменения заполнения в вариантах с "дыханием" и "перетеканием яркости"?

6. Что такое квантация заполнения и как она влияет на наблюдаемую яркость?

7. Почему в варианте со ступенчатыми уровнями удобно использовать таблицу LUT?

8. Чем отличается "выключение ШИМ" через `duty = 0` от остановки таймера?

9. Как частота ШИМ зависит от `CAPCOM[0]` и делителя таймера?

10. Почему при изменении периода нужно пересчитывать compare-регистры?

**ТЕСТОВЫЕ ВОПРОСЫ**

Выберите один правильный ответ.

**1. Какой таймер в лабораторной работе формирует ШИМ на выводах PA.8 и PA.9?**

А) `TMR32`.
Б) `TMR1`.
В) `RTC`.

**2. Какой регистр используется для задания периода ШИМ в базовом варианте?**

А) `CAPCOM[0].VAL`.
Б) `DATAOUTSET`.
В) `IM`.

**3. Какие compare-регистры используются для двух каналов ШИМ в работе?**

А) `CAPCOM[1]` и `CAPCOM[4]`.
Б) `CAPCOM[2]` и `CAPCOM[3]`.
В) `CAPCOM[0]` и `CAPCOM[1]`.

**4. Какая базовая частота ШИМ задана в общих сведениях для всех вариантов?**

А) 100 Гц.
Б) 500 Гц.
В) 1 кГц.

**5. Каковы начальные заполнения каналов в базовом режиме?**

А) 50% и 50%.
Б) 25% и 75%.
В) 0% и 100%.

**6. Для чего в данной работе используется таймер TMR32?**

А) Для формирования питания LDO.
Б) Для периодического изменения параметров ШИМ.
В) Для хранения таблицы LUT.

**7. Что нужно сделать после изменения частоты ШИМ в варианте 3.5?**

А) Только изменить GPIO.
Б) Пересчитать compare-регистры по процентам заполнения.
В) Остановить и удалить проект.

**8. Какой закон изменения выполняется для второго канала в варианте "перетекание яркости"?**

А) `duty(PA.9) = duty(PA.8)`.
Б) `duty(PA.9) = 100% - duty(PA.8)`.
В) `duty(PA.9)` всегда 0%.

**9. Какой набор уровней допускается в варианте со ступенчатой яркостью?**

А) Только 0% и 100%.
Б) {10, 20, 30, 40, 50}.
В) {0, 25, 50, 75, 100}.

**10. Чем должна выполняться смена режима в варианте с гейтингом?**

А) Переключением режимов портов GPIO.

Б) Изменением compare-регистров без отключения `TMR1`.

В) Выключением питания платы.

**11. Что будет правильным критерием проверки без внешнего оборудования?**

А) Наблюдение цвета платы.

Б) Соответствие значений `CAPCOM[2].VAL` и `CAPCOM[3].VAL` расчетным.

В) Переименование `makefile`.

**12. Какой инструмент наиболее удобен для точной проверки частоты и заполнения ШИМ?**

А) Осциллограф.

Б) Текстовый редактор.

В) Калькулятор Windows.

**13. Что означает для светодиода увеличение `duty-cycle` при одинаковом подключении?**

А) Снижение яркости.

Б) Рост средней яркости.

В) Полное выключение канала.

**14. Какой диапазон заполнения допустим для корректной работы вариантов?**

А) От -100% до 100%.

Б) От 0% до 100%.

В) Только 25% и 75%.

**15. Что произойдет при compare >= period?**

А) Формирование требуемого ШИМ становится некорректным.

Б) Частота UART увеличится.

В) OpenOCD автоматически исправит конфигурацию.

**16. Что в работе понимается под квантацией заполнения?**

А) Непрерывное изменение частоты процессора.

Б) Использование только заранее заданных дискретных уровней `duty-cycle`.

В) Полное отключение таймера `TMR1`.

**17. Какой вывод верен для варианта “дыхание” одного канала?**

А) Оба канала должны изменяться одинаково.

Б) `PA.8` изменяется, а `PA.9` остается с фиксированным заполнением.

В) `PA.9` работает только как вход GPIO.

**18. Что требуется сделать в варианте с гейтингом при “выключении” ШИМ?**

А) Остановить таймер `TMR1`.

Б) Изменить режим порта `PA.8` и `PA.9` на обычный GPIO.

В) Установить compare-регистры так, чтобы заполнение стало нулевым или заданным по условию.

**19. Какой параметр должен оставаться неизменным в варианте 3.5 при переключении частоты ШИМ?**

А) Процент заполнения каналов.

Б) Частота прерываний `TMR32` обязательно должна стать равной частоте ШИМ.

В) Значение `CAPCOM[0].VAL` должно оставаться постоянным.

**20. Для чего в вариантах со ступенчатым или плавным изменением удобно использовать обработчик TMR32_IRQHandler()?**

А) Чтобы периодически пересчитывать и обновлять параметры ШИМ.

Б) Чтобы прошивать микроконтроллер без OpenOCD.
В) Чтобы заменить файл `startup_k1921vg015.S`.

# Глава 4. Измерение физического мира: АЦП и датчики

**АЦП: назначение и основные параметры**

*Аналого-цифровой преобразователь (АЦП)* преобразует аналоговое напряжение на входе в цифровой код.

АЦП необходим в тех случаях, когда микроконтроллеру требуется работать не только с логическими уровнями «0» и «1», но и с изменяющимися по величине электрическими сигналами. К таким сигналам относятся напряжения от датчиков температуры, освещенности, давления, положения и других аналоговых источников. Без АЦП центральное ядро не может напрямую обработать такой сигнал, поскольку вычисления во встроенной программе выполняются только над цифровыми значениями.

Процесс преобразования включает выборку входного сигнала, его сравнение с внутренними уровнями и формирование числового результата, пропорционального входному напряжению. Таким образом, АЦП служит связующим звеном между физическим процессом и программной обработкой данных в микроконтроллерной системе.

Основные параметры АЦП: разрядность (например, 12 бит), опорное напряжение $V_{ref}$ (в данной работе – AREF), время выборки и преобразования, а также погрешности, включая квантование, шум и нелинейность.

Для N-разрядного АЦП диапазон кодов составляет от 0 до ($2^N$ - 1). При известном $V_{ref}$ код можно пересчитать в напряжение по формуле:

$$U = \frac{Code}{2^N - 1} \cdot V_{ref} \quad (7)$$

Дополнительно следует учитывать, что результат преобразования всегда принадлежит ограниченному диапазону кодов. Для N-разрядного преобразователя максимальный код определяется числом уровней квантования, а переход от одного кода к следующему соответствует минимально различимому изменению входного напряжения. Эта величина называется *шагом квантования* или *разрешающей способностью АЦП*. Чем меньше шаг квантования, тем точнее цифровой код отражает изменение измеряемого сигнала.

$$Code_{max} = 2^N - 1 \quad (8)$$

При фиксированном опорном напряжении шаг квантования определяется отношением опорного напряжения к максимальному числу кодов. На практике это означает, что любое аналоговое напряжение в пределах диапазона преобразования округляется к ближайшему дискретному уровню. По этой причине в измерениях всегда присутствует квантование, которое является неотъемлемым свойством любого АЦП.

$$\Delta U = \frac{V_{ref}}{(2^N - 1)}$$

Если входное напряжение превышает опорное напряжение, АЦП переходит в режим насыщения и формирует код, близкий к максимальному значению. Если же входное напряжение близко к нулю, формируется минимальный код. Поэтому корректный выбор опорного напряжения и режима измерения непосредственно влияет на полезный диапазон измеряемых значений и точность вычислений в прикладной программе.

$$U = \frac{Code}{2^N - 1} * V_{ref} \quad (9)$$

**2.2 Модуль ADCSAR и секвенсоры измерений**

В микроконтроллере К1921ВГ015 преобразования выполняются модулем ADCSAR. Для удобства измерений могут использоваться секвенсоры – последовательности измерения каналов.

Модуль ADCSAR реализует преобразование по принципу последовательного приближения. Такой способ позволяет получить цифровой код за конечное число тактов при достаточно высокой точности и скорости работы. В практических задачах это делает модуль удобным для опроса как внешних аналоговых входов, так и внутренних источников сигнала, например термодатчика.

Секвенсор представляет собой аппаратно задаваемую последовательность действий: выбор канала, выдержка времени подключения, запуск преобразования и сохранение результата. Использование секвенсора упрощает программную организацию измерений, так как часть операций выполняется периферийным модулем автоматически и не требует постоянного участия центрального процессора.

В данной работе секвенсор 0 настраивается на измерение внутреннего датчика температуры, подключенного к каналу 10.

Для корректного измерения важно задать достаточное время подключения канала, чтобы входная цепь успела установиться перед началом преобразования.

Особенно это важно при работе с внутренними источниками и высокоомными цепями, для которых напряжение не устанавливается мгновенно. Если время подключения выбрано недостаточным, АЦП может начать преобразование до завершения переходного процесса, что приводит к дополнительной ошибке результата.

### 2.3 Калибровка АЦП

Калибровка уменьшает *систематические погрешности АЦП*, включая смещение и коэффициент передачи.

Под систематическими погрешностями понимаются такие отклонения, которые повторяются от измерения к измерению и обусловлены особенностями конкретного экземпляра микросхемы, технологическим разбросом параметров и внутренними неточностями схемы преобразователя. Без калибровки результаты могут быть смещены относительно реального значения даже при стабильном входном сигнале.

Выполнение калибровки перед началом работы позволяет повысить достоверность измерений и уменьшить последующую вычислительную коррекцию в прикладной программе. По этой причине калибровка обычно выполняется на этапе инициализации периферийного модуля до начала регулярных измерений.

В работе используется автоматическая калибровка, включаемая при инициализации модуля АЦП.

### 2.4 Внутренний термодатчик и вычисление температуры

Внутренний термодатчик формирует напряжение, зависящее от температуры кристалла. АЦП измеряет это напряжение как обычный канал.

Следует учитывать, что внутренний термодатчик характеризует прежде всего температуру самого кристалла микроконтроллера, а не температуру окружающего воздуха в удаленной точке. Поэтому получаемое значение удобно использовать для контроля теплового режима микроконтроллера, оценки нагрева платы и реализации защитных либо диагностических алгоритмов.

Перевод цифрового кода в температуру выполняется поэтапно. Сначала код АЦП пересчитывается в напряжение с учетом опорного напряжения AREF и разрядности преобразователя, после чего к полученному напряжению применяется формула или калибровочные коэффициенты, указанные в документации на микроконтроллер. Такой

подход позволяет связать электрический сигнал датчика с физической величиной в градусах Цельсия.

Алгоритм пересчета кода АЦП в температуру обычно включает перевод кода в напряжение с учетом AREF и применение формулы либо коэффициентов, указанных в документации на микроконтроллер.

При вычислении температуры по внутреннему термодатчику обычно используется линейная аппроксимация, в которой напряжение датчика сравнивается с опорной точкой, приведенной в документации. Сначала цифровой код АЦП пересчитывается в напряжение на выходе термодатчика, а затем по коэффициенту чувствительности определяется температура. Такой подход удобен тем, что отделяет аппаратный этап измерения от программного этапа вычисления физической величины.

$$U_{TS} = \frac{Code}{(2^N - 1)} \cdot A_{REF} \tag{10}$$

В общем виде пересчет в температуру может быть записан через опорное напряжение датчика при температуре T0 и коэффициент чувствительности K. Конкретные значения коэффициентов принимаются по документации или калибровочным данным используемого микроконтроллера.

$$T = \frac{U_{TS} - U_0}{K} + T_0 \tag{11}$$

Следует учитывать, что на итоговый результат влияют шум источника питания, разброс параметров внутреннего датчика, собственный нагрев микроконтроллера и нестабильность опорного напряжения. Для повышения устойчивости результата в практических системах часто используют повторные измерения, усреднение нескольких выборок и программную фильтрацию.

### 2.5 Периодические измерения: таймер и флаг

*Периодические измерения* удобно запускать с фиксированным интервалом времени. Для этого используется таймер `TMR32`, формирующий прерывание с периодом 1 секунда.

Применение аппаратного таймера позволяет получить устойчивый временной интервал между измерениями и не зависит от задержек в основном цикле программы. Это особенно важно в системах, где кроме опроса датчика выполняются и другие задачи, влияющие на длительность прохода по циклу `while(1)`.

Использование программного флага в сочетании с прерыванием делает структуру программы более надежной: обработчик прерывания только сообщает о наступлении события, а все длительные действия – запуск АЦП, вычисления и вывод по UART – выполняются вне контекста прерывания. Такой подход снижает вероятность потери событий и упрощает отладку программы.

В обработчике прерывания рекомендуется выполнять минимальный объем действий: устанавливать программный флаг, например `meas_tick = 1`. Основные вычисления и вывод по UART выполняются в основном цикле при установленном флаге.

### 2.6 Вывод результатов по UART

Для наблюдения результатов измерений используется вывод в терминал по UART. На практике часто применяется `retarget printf`, то есть перенаправление стандартной функции `printf` в UART.

UART является простым и широко распространенным последовательным интерфейсом, который позволяет передавать диагностические сообщения, результаты измерений и команды управления между микроконтроллером и персональным компьютером.

Для лабораторных и отладочных работ такой способ удобен тем, что не требует сложного пользовательского интерфейса: достаточно подключить терминальную программу и открыть соответствующий COM-порт.

Передача по UART выполняется асинхронно, поэтому одинаковые параметры соединения должны быть настроены как в прошивке, так и в терминальной программе. Наиболее распространенная конфигурация 8N1 означает передачу восьми бит данных без бита четности и с одним стоп-битом. При несовпадении скорости или формата символов текстовый вывод будет искажен либо не появится в терминале.

Параметры порта – скорость, биты данных, четность и стоп-биты – задаются в соответствии с настройками проекта; в учебных примерах чаще всего используется режим 115200, 8N1.

При выводе данных по UART удобным считается человекочитаемый формат, в котором кроме самой температуры могут передаваться служебные подписи, единицы измерения и дополнительные диагностические данные. Это упрощает контроль корректности программы и позволяет быстрее обнаруживать ошибки настройки АЦП, таймера или последовательного интерфейса. При необходимости в терминал можно выводить не только итоговую температуру, но и исходный код АЦП, рассчитанное напряжение, усредненное значение и сообщения о состоянии программы.

# Практическая работа №4 (бывшая 6)

## АЦП и внутренний термодатчик

**СОДЕРЖАНИЕ**

1. Цель и оборудование
2. Задание
3. Дополнительные варианты заданий
3.1 Вариант 1. Пороговый контроль с гистерезисом
3.2 Вариант 2. Калибровка термодатчика
3.3 Вариант 3. Усреднение и фильтрация
3.4 Вариант 4. Два режима измерения
3.5 Вариант 5. Статистическая обработка
3.6 Вариант 6. Температурный логгер
3.7 Дополнительное задание
4. Ход работы
4.1 Подключение платы и открытие проекта
4.2 Сборка проекта и настройка терминала
4.3 Анализ исходного кода проекта
4.4 Выполнение варианта задания
4.5 Подготовка материалов отчета
5. Алгоритм работы программы
6. Результат
7. Требования к отчету
8. Контрольные вопросы
10. Тестовые вопросы

## ЦЕЛЬ И ОБОРУДОВАНИЕ

**Цель:** изучить принципы работы аналого-цифрового преобразователя (АЦП) микроконтроллера К1921ВГ015 и освоить методику преобразования кода АЦП в физическую величину на примере измерения температуры внутреннего термодатчика.

**Оборудование:** макетно-отладочная плата с микроконтроллером К1921ВГ015; персональный компьютер; USB Type-C для подключения платы; среда разработки (VS Code/IDE по методическим указаниям); терминальная программа для приема данных UART.

## ЗАДАНИЕ

Выполнить лабораторную работу согласно следующему содержанию:

- Открыть или создать проект для микроконтроллера К1921ВГ015 и обеспечить в нем корректный вывод значения температуры по UART.
- Проинициализировать модуль АЦП (ADCSAR) в 12-разрядном режиме с автоматической калибровкой.
- Настроить секвенсор 0 на измерение внутреннего термодатчика (канал 10) и задать дополнительную задержку подключения канала.
- Организовать периодический запуск измерений с периодом 1 секунда с использованием таймера TMR32 через прерывание и программный флаг.
- Перевести код АЦП в напряжение с учетом AREF и вычислить температуру в градусах Цельсия по формуле из документации.
- Выводить результат измерения в терминал по UART в удобном формате.

## ДОПОЛНИТЕЛЬНЫЕ ВАРИАНТЫ ЗАДАНИЙ

### 4.1 Вариант 1. Пороговый контроль с гистерезисом

Реализовать систему контроля температуры с двумя порогами – верхним и нижним. При превышении верхнего порога выводить сообщение о перегреве, при снижении температуры ниже нижнего порога – сообщение о возврате в нормальный режим.

### 4.2 Вариант 2. Калибровка термодатчика

Реализовать программную калибровку термодатчика. Предусмотреть возможность задания смещения и коэффициента масштабирования температуры. Откалиброванное значение выводить по UART.

### 4.3 Вариант 3. Усреднение и фильтрация

Реализовать цифровую фильтрацию результатов измерения температуры с использованием скользящего среднего по N последним измерениям, где N >= 8. По UART выводить текущее и отфильтрованное значения температуры.

### 4.4 Вариант 4. Два режима измерения

Реализовать два режима работы системы измерения температуры: нормальный режим – измерение 1 раз в секунду и ускоренный режим – измерение 10 раз в секунду. Переключение режимов выполнить программно, например по команде через UART или по температурному событию.

### 4.5 Вариант 5. Статистическая обработка

Реализовать накопление статистики измерений температуры за заданный интервал времени, например за 1 минуту: минимальное, максимальное и среднее значения. По завершении интервала вывести результаты по UART.

### 4.6 Вариант 6. Температурный логгер

Реализовать температурный логгер: сохранять результаты измерений температуры в массиве фиксированного размера. При заполнении массива передавать накопленные данные по UART и очищать буфер.

### 4.7 Дополнительное задание

Изменение скважности ШИМ на светодиоде, реализованном в лабораторной работе №3, в зависимости от температуры.

## ХОД РАБОТЫ

### 5.1 Подключение платы и открытие проекта

Подключить макетно-отладочную плату к персональному компьютеру по USB и убедиться, что драйверы и интерфейсы отладчика установлены корректно.

Открыть среду разработки и загрузить любой проект для микроконтроллера К1921ВГ015 либо создать новый проект на основе доступного шаблона. При настройке проекта необходимо выбрать параметры, соответствующие используемой плате и интерфейсу отладки, например Example variant = bare и OpenOCD Debugger interface = onboard_ftdi.

### 5.2 Сборка проекта и настройка терминала

Собрать проект и выполнить прошивку микроконтроллера через стандартные команды Build и Flash (Program).

Открыть терминальную программу для UART и подключиться к соответствующему COM-порту. Параметры порта установить в соответствии с проектом, как правило 115200, 8N1.

Запустить программу на плате и убедиться, что в терминале появляется периодический вывод температуры.

### 5.3 Анализ исходного кода проекта

Проанализировать исходный код используемого проекта и определить, где выполняется инициализация ADCSAR, каким образом настроены секвенсор 0 и таймер TMR32, а также как выполняется пересчет кода АЦП в температуру.

### 5.4 Выполнение варианта задания

Реализовать выбранный вариант дополнительного задания и проверить корректность его работы по выводу UART.

### 5.5 Подготовка материалов отчета

Сделать скриншот работы программы с выводом UART и включить его в отчет.

**Пошаговая инструкция выполнения работы**

Рекомендуемая последовательность выполнения лабораторной работы приведена ниже.

1. Подключить макетно-отладочную плату к персональному компьютеру по USB, дождаться определения отладчика и виртуального COM-порта в операционной системе**.**

2. Открыть среду разработки и загрузить любой проект для микроконтроллера К1921ВГ015 либо создать новый проект на основе доступного шаблона, после чего проверить настройки сборки и интерфейса отладки.

3. Проверить наличие в проекте инициализации системного тактирования, UART и модуля ADCSAR, а при отсутствии соответствующих фрагментов добавить их в исходный код.

4. Настроить модуль АЦП на 12-разрядное преобразование, включить автоматическую калибровку и подготовить секвенсор 0 для измерения внутреннего термодатчика, подключенного к каналу 10.

5. Задать дополнительную задержку подключения канала, достаточную для установления сигнала на входе преобразователя перед началом измерения.

6. Настроить таймер TMR32 на формирование прерывания с периодом 1 секунда и в обработчике прерывания устанавливать программный флаг запуска измерения.

7. В основном цикле программы организовать проверку флага, запуск преобразования, ожидание его завершения, чтение кода АЦП и дальнейший пересчет результата в напряжение и температуру.

8. Настроить вывод диагностической информации по UART и убедиться, что параметры терминальной программы совпадают с параметрами, заданными в прошивке, например 115200 бод, 8 бит данных, без четности и 1 стоп-бит.

9. Выполнить сборку проекта, прошивку микроконтроллера и открыть UART-терминал для наблюдения результатов измерения температуры.

10. Проверить корректность периодического вывода, выполнить выбранный вариант дополнительного задания и зафиксировать результаты работы программы в отчете в виде скриншота и кратких пояснений.

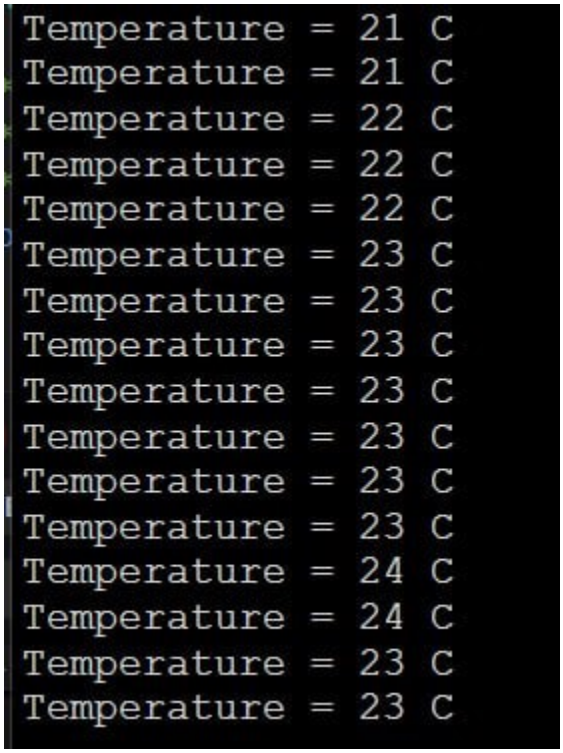



Рисунок 4.1 Пример вывода результата измерения температуры по UART

**АЛГОРИТМ РАБОТЫ ПРОГРАММЫ**

Ниже приведен рекомендуемый каркас программы для периодического измерения температуры с минимальной нагрузкой в обработчике прерывания:

```
	volatile int meas_tick = 0;

void TMR32_IRQHandler(void)
{
/* Очистить флаг прерывания таймера. */
meas_tick = 1;
}

int main(void)
{
/* Инициализация: тактирование, UART, ADCSAR, секвенсор 0 (канал
10), калибровка. */
/* Таймер TMR32: период 1 секунда, прерывания разрешены. */

while (1)
{
if (meas_tick)
{
meas_tick = 0;

/* Запуск преобразования (секвенсор 0). */
/* Ожидание окончания и чтение кода АЦП. */
/* Пересчет Code -> напряжение -> температура (по формуле из
документации). */

printf("Temperature = %d C\r\n", temp_c);
}
}
}
```

**РЕЗУЛЬТАТ**

Реализована система периодического измерения температуры внутреннего термодатчика с выводом результата по UART. Полученные данные позволяют наглядно проследить связь между кодом АЦП и физической величиной.

Ниже приведен пример текстового вывода программы в терминале UART:

Temperature = 29 C
Temperature = 29 C
Temperature = 30 C
Temperature = 30 C

**ТРЕБОВАНИЯ К ОТЧЕТУ**

- Титульные данные: номер и название работы, ФИО, группа, дата выполнения.
- Краткое описание принципа измерения: что измеряет АЦП, какая разрядность и какое опорное напряжение используется.
- Описание структуры программы: инициализация ADCSAR, настройка секвенсора, таймера и UART, алгоритм основного цикла.
- Результаты: скриншот вывода UART; при выполнении варианта – подтверждение его работы в виде сообщений, режимов, статистики и других результатов.
- Ответы на контрольные вопросы.

## КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое разрядность АЦП и как она влияет на шаг квантования?
2. Как пересчитать код АЦП в напряжение при известном опорном напряжении AREF?
3. Зачем нужна калибровка АЦП и что она компенсирует?
4. Почему для внутренних и высокоомных источников важно время выборки, то есть задержка подключения канала?
5. Чем удобна схема «прерывание + флаг» по сравнению с выполнением вывода в обработчике прерывания?
6. Какие причины вызывают шум измерений и как программно снизить его влияние?
7. Почему при работе с внутренним термодатчиком важно правильно выбирать время выборки и задержку подключения канала?
8. Для чего в лабораторной работе используется секвенсор 0 модуля ADCSAR и какие преимущества он дает по сравнению с ручным запуском одиночного преобразования?
9. Почему внутренний термодатчик характеризует прежде всего температуру кристалла микроконтроллера, а не температуру окружающей среды?
10. Какие преимущества дает вывод результатов измерения температуры по UART при отладке и проверке работы программы?

## ТЕСТОВЫЕ ВОПРОСЫ

Выберите один правильный ответ.

**1. Для чего в микроконтроллерной системе используется АЦП?**

А) Для преобразования аналогового напряжения в цифровой код.
Б) Для хранения программы пользователя.
В) Для формирования тактового сигнала ядра.

**2. Что определяет разрядность АЦП?**

А) Количество уровней квантования и точность представления сигнала.
Б) Номер используемого UART-порта.
В) Число таймеров в микроконтроллере.

**3. Для чего необходимо опорное напряжение AREF?**

А) Для задания диапазона преобразования АЦП и пересчета кода в напряжение.
Б) Для включения внутреннего термодатчика.
В) Для синхронизации работы UART.

**4. Какова основная функция секвенсора ADCSAR?**

А) Автоматическая организация последовательности измерений каналов.
Б) Передача данных в терминал по UART.
В) Хранение программы прерывания таймера.

**5. Зачем выполняется калибровка АЦП?**

А) Для уменьшения систематических погрешностей преобразования.
Б) Для увеличения объема оперативной памяти.
В) Для выбора номера COM-порта.

**6. Что измеряет внутренний термодатчик микроконтроллера?**

А) Температуру кристалла микроконтроллера.
Б) Температуру воздуха в помещении с высокой точностью.
В) Напряжение питания USB-порта.

**7. Для чего в программе используется таймер TMR32?**

А) Для периодического запуска измерений через заданный интервал времени.
Б) Для хранения результатов измерений в энергонезависимой памяти.
В) Для преобразования кода АЦП в температуру.

**8. Почему в обработчике прерывания рекомендуется устанавливать только флаг события?**

А) Чтобы сократить время работы обработчика и выполнить основные действия в основном цикле.
Б) Чтобы повысить опорное напряжение АЦП.
В) Чтобы автоматически включить калибровку UART.

**9. Что означает конфигурация UART 8N1?**

А) 8 бит данных, без четности, 1 стоп-бит.
Б) 8 стоп-битов, нормальный режим, 1 бит данных.
В) 8 каналов АЦП, нулевая четность, 1 таймер.

**10. При каком условии текст в терминале UART будет отображаться корректно?**

А) При совпадении параметров соединения в прошивке и терминальной программе.

Б) Только при отключенном таймере TMR32.

В) Только при максимальной разрядности АЦП.

**11. Что характеризует шаг квантования АЦП?**

А) Минимальное изменение входного напряжения, различимое преобразователем.

Б) Число UART-пакетов в секунду.

В) Максимальную длину строки терминала.

**12. Что произойдет при превышении входным напряжением уровня опорного напряжения АЦП?**

А) Результат преобразования приблизится к максимальному коду.

Б) Таймер TMR32 автоматически остановится.

В) UART перейдет в режим приема.

**13. Для чего задается дополнительная задержка подключения канала перед преобразованием?**

А) Для установления сигнала на входе АЦП перед началом измерения.

Б) Для увеличения объема оперативной памяти.

В) Для переключения режима работы UART.

**14. Какой физический смысл имеет величина Code_max = 2^N − 1?**

А) Это максимальный код, который может сформировать N-разрядный АЦП.

Б) Это число доступных COM-портов.

В) Это коэффициент усиления термодатчика.

**15. Для чего используется программный флаг meas_tick?**

А) Для передачи события из обработчика прерывания в основной цикл программы.

Б) Для хранения коэффициента калибровки UART.

В) Для выбора номера канала АЦП.

**16. Почему вывод большого объема данных не рекомендуется выполнять непосредственно в обработчике прерывания?**

А) Это увеличивает время работы обработчика и ухудшает реакцию системы на другие события.

Б) Это отключает внутренний термодатчик.

В) Это повышает опорное напряжение AREF.

**17. Что позволяет сделать усреднение нескольких измерений температуры?**

А) Снизить влияние случайного шума на результат.

Б) Уменьшить разрядность АЦП.

В) Отказаться от использования таймера.

**18. Что обозначает коэффициент чувствительности K в формуле пересчета температуры?**

А) Изменение выходного напряжения датчика при изменении температуры.

Б) Количество разрядов UART.

В) Частоту прерываний таймера.

**19. Что необходимо для корректного отображения результатов в UART-терминале?**

А) Совпадение скорости и формата передачи в прошивке и терминальной программе.

Б) Использование только внешнего датчика температуры.

В) Отключение калибровки АЦП.

**20. Какое преимущество дает представление температуры в человекочитаемом формате по UART?**

А) Упрощает контроль работы программы и анализ результатов измерения.

Б) Увеличивает аппаратную разрядность АЦП.

В) Полностью устраняет систематическую погрешность.

# Глава 5. Истинная случайность: TRNG

### 1. Блок *TRNG* в составе К1921ВГ015

В работе используется аппаратный истинный генератор случайных чисел *TRNG*. TRNG входит в состав микроконтроллера К1921ВГ015 наряду с UART, таймерами, CRC, HASH и другими периферийными блоками. Для данной работы важно понимать, что генератор является аппаратным источником случайных данных и не требует программного вычисления псевдослучайной последовательности.

Практическая ценность TRNG состоит в том, что программа получает уже сформированные случайные слова через регистровый интерфейс. Поэтому задача состоит не в построении алгоритма генерации, а в корректной инициализации блока, контроле его состояния, чтении FIFO и безопасной передаче результатов по UART.




Рисунок 5.1. Структура макетно-отладочной платы К1921ВГ015

1 – блок источников питания;

2 – блок программатора;

3 – микросхема 1921ВГ015;

4 – сервисный блок;

5 – батарейный отсек CR1220;

6 – светодиодная индикация.

### 2. Принцип работы TRNG: прогрев, выборка и *FIFO*

Работа TRNG в этой работе строится как последовательность этапов: сначала блок включается, затем ему задаются интервалы прогрева и выборки, после чего случайные

данные начинают поступать во встроенный *FIFO*. Прогрев нужен для выхода внутренней аналоговой части генератора в устойчивый режим, а период выборки определяет темп формирования новых значений.

FIFO позволяет развязать момент появления случайных данных и момент их чтения программой. Это особенно удобно при работе по *polling*: основная программа периодически опрашивает уровень заполнения FIFO и только при наличии данных считывает нужное число слов. В данной работе базовый сценарий - чтение 8 случайных слов подряд и их вывод в шестнадцатеричном виде.

Такой режим хорошо показывает разницу между «данные уже готовы» и «данных пока нет». Нужно опираться не на задержки в коде, а на регистр уровня FIFO и статусные признаки блока. Именно поэтому работа ориентирована на дисциплинированный обмен с периферией через регистры состояния.

Таблица 5.1

| Этап работы TRNG | Что происходит | Настроить | Практический признак |
|---|---|---|---|
| Прогрев | Внутренняя часть генератора выводится в устойчивый режим до начала штатной генерации данных. | Задается регистром WARMPERIOD до запуска TRNG. | Сразу после включения данные могут еще не быть готовы для чтения. |
| Выборка | Генератор формирует очередные случайные значения с заданным темпом. | Задается регистром SAMPERIOD. | Частота накопления слов во FIFO зависит от выбранного периода. |
| Накопление во FIFO | Сформированные случайные слова помещаются во внутренний буфер FIFO. | Контролируется по регистру FIFOLEV. | Программа видит, когда можно безопасно считывать пакет данных. |
| Чтение по polling | Основной цикл программы опрашивает статус и уровень FIFO без прерываний. | Реализуется логикой while(1) и командами UART. | По команде r считываются 8 слов подряд только при наличии данных. |
| Диагностика через UART | Пользователь получает случайные числа и состояние блока в текстовом виде. | Выводятся DATA, STAT и FIFOLEV. | Можно доказать корректную работу TRNG без осциллографа и без прерываний. |

Этапы работы с блоком TRNG

TRNG в работе рассматривается как периферийный блок с внутренним FIFO, накапливающим случайные слова до чтения программой. Чем аккуратнее организован контроль FIFOLEV и регистра состояния, тем надежнее работает программа и тем легче объяснить полученный результат.

### 3. Основные регистры TRNG

Для выполнения работы достаточно использовать минимальный набор регистров TRNG: регистры задания временных параметров, управляющий регистр, регистр уровня FIFO, регистр состояния и регистр данных. Важен сам алгоритм работы: сначала записать периоды прогрева и выборки, затем включить TRNG, после чего в основном цикле проверять наличие данных.

Таблица 5.2

| Регистр | Назначение | Проверить |
|---|---|---|
| WARMPERIOD | Задает длительность прогрева TRNG перед штатной выдачей данных. | Записан ли параметр до запуска генератора и соответствует ли он выбранному сценарию. |
| SAMPERIOD | Задает период выборки/формирования случайных данных. | Влияет ли выбранное значение на темп накопления слов во FIFO. |
| FIFOLEV / STAT | Показывают уровень заполнения FIFO и состояние блока TRNG. | Не считываются ли данные раньше времени и корректно ли программа реагирует на статус. |

Основные регистры блока TRNG

При написании программы полезно вынести действия по настройке в отдельные функции, например TRNG_Init(), TRNG_Read8Words() и TRNG_PrintStatus(). Это упрощает логику main() и делает очевидной последовательность действий: конфигурация -> ожидание готовности данных -> чтение -> вывод через UART.

**4. Работа по polling и обмен через *UART***

В данной работе принципиально не используются прерывания TRNG. Все действия выполняются по polling: программа в основном цикле проверяет регистр FIFOLEV, анализирует регистр STAT и принимает решение, можно ли считывать очередную порцию данных. Такой подход нужен для того, чтобы увидеть прямую связь между состоянием периферии и программной логикой.

*UART* используется как пользовательский интерфейс и как средство диагностики. По команде **r** программа должна считать 8 случайных слов и вывести их в шестнадцатеричном формате, а по команде **s** - показать текущее состояние блока TRNG: значение регистра STAT и текущий уровень FIFO. Благодаря этому легко проверить, что чтение действительно зависит от состояния FIFO, а не от искусственных задержек.

Таблица 5.3

| Регистр/ узел | Назначение | Использование |
|---|---|---|
| CTRL | Управляющий регистр TRNG: запуск, программный сброс, базовая конфигурация. | После записи WARMPERIOD и SAMPERIOD переводит блок в рабочий режим без тестов и прерываний. |
| DATA | Регистр чтения случайных слов. | Считывается, когда FIFOLEV показывает наличие данных; значения сохраняются в массив и печатаются через UART. |
| FIFOLEV | Текущий уровень заполнения внутреннего FIFO. | Используется в основном цикле для решения, можно ли читать пакет из 8 слов подряд. |
| STAT | Статусный регистр TRNG. | Выводится по команде s и после чтения для контроля состояния блока. |
| UART0 | Канал пользовательского интерфейса и диагностики. | Принимает команды r и s и выводит считанные числа в шестнадцатеричном формате. |

Регистры и узлы при работе TRNG в режиме polling

Важно: если FIFO еще не накопил требуемое количество слов, программа не должна считывать «пустые» данные. Корректный вариант поведения - дождаться достаточного

уровня заполнения, сообщить пользователю о текущем состоянии или продолжить опрос до готовности. Такой контроль обязателен для всех вариантов практической части.

Для работы через UART на отладочной плате удобно использовать тот же механизм протоколирования, что и в предыдущих лабораторных работах: короткие команды пользователя, текстовые сообщения о состоянии периферии и наглядный вывод чисел. Это превращает TRNG из «черного ящика» в наблюдаемый и проверяемый блок.

### 5. Практическое использование TRNG в учебных задачах

TRNG полезен не только как изолированный периферийный блок. Случайные числа можно использовать для тестирования алгоритмов сортировки и поиска, для моделирования событий, для заполнения буферов и для учебной обработки потоков данных. Поэтому в практических заданиях после базового чтения предлагаются варианты с софтовым FIFO, статистикой, обработкой массивов и событийной логикой по порогу FIFO.

С точки зрения аппаратной платформы лабораторная работа удобно сочетается с возможностями платы: UART служит для управления и вывода данных, а пользовательские светодиоды на PA12-PA15 могут применяться для индикации событий и пороговых состояний FIFO

# Практическая работа № 5

**Генератор случайных чисел TRNG микроконтроллера К1921ВГ015**

**СОДЕРЖАНИЕ**

ЦЕЛЬ И ОБОРУДОВАНИЕ
СТРУКТУРА ПРОЕКТА И ОСНОВНЫЕ ФАЙЛЫ
ПОДГОТОВКА ОБОРУДОВАНИЯ И ПРОГРАММНОЙ СРЕДЫ
СОЗДАНИЕ, СБОРКА И ПРОШИВКА ПРОЕКТА
Создание нового проекта на основе примера
Замена файла main.c
Сборка проекта
Прошивка микроконтроллера
ПРАКТИЧЕСКИЕ ЗАДАНИЯ
Общие правила для всех вариантов
Задание 7.1. Кольцевой буфер для случайных слов
Задание 7.2. Вывод статистики случайных чисел
Задание 7.3. Журнал пакетов TRNG
Задание 7.4. Покомпонентное опустошение FIFO
КОНТРОЛЬНЫЕ ВОПРОСЫ
ТЕСТОВЫЕ ВОПРОСЫ

## ЦЕЛЬ И ОБОРУДОВАНИЕ

Цель: познакомиться с аппаратным генератором случайных чисел TRNG микроконтроллера К1921ВГ015, научиться включать блок TRNG, настраивать параметры прогрева и выборки, считывать данные по polling через FIFO и выводить результаты через UART.

Оборудование: отладочная плата с микроконтроллером К1921ВГ015; персональный компьютер с Windows; USB-кабель для подключения платы; установленная среда Visual Studio Code с расширением NIIET Aspect; SDK, toolchain и OpenOCD; терминальная программа для работы по UART.

## СТРУКТУРА ПРОЕКТА И ОСНОВНЫЕ ФАЙЛЫ

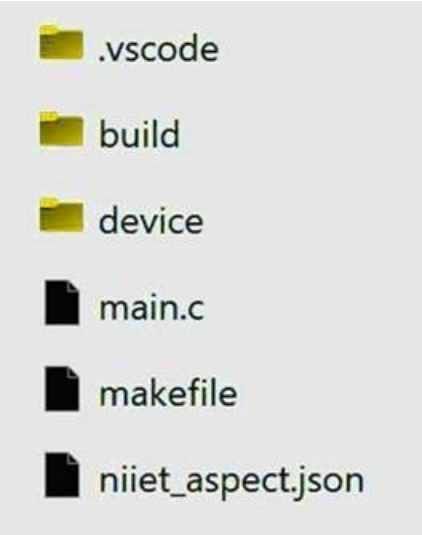



Рисунок 5.2. Структура файлов проекта

Таблица 5.4 Основные файлы проекта

| Категория | Файл | Назначение | Как работает |
|---|---|---|---|
| Основные файлы проекта | main.c | Ваш основной рабочий файл с кодом программы. | Пишете здесь всю логику для выполнения лабораторных работ. |
| Основные файлы проекта | makefile | Скрипт для автоматической сборки проекта. | При запуске команды make в терминале компилирует все .c файлы в .o, линкует их в .elf и формирует .hex/.dump для прошивки и отладки. |
| Основные файлы проекта | config.json | Конфигурация проекта для среды разработки. | Содержит настройки путей и параметров инструментария. |
| Основные файлы проекта | niiet_aspect.json | Специфичная конфигурация для инструментов NIIET Aspect. | Чтобы прошить плату во Flash, нужно заменить scriptFile с k1921vg015_ram.ld на k1921vg015_flash.ld. |
| Системные файлы (не трогайте без необходимости) | system_k1921vg015.c | Базовая системная инициализация. | Инициализирует системную частоту и служебные параметры МК. |

| | | | |
|---|---|---|---|
| Системные файлы (не трогайте без необходимости) | startup_k1921vg015.S | Стартовый код. | Запускается до main(), настраивает стек и таблицу векторов. |
| Системные файлы (не трогайте без необходимости) | printf.c | Поддержка printf и перенаправления вывода. | Используется для сообщений в UART. |
| Системные файлы (не трогайте без необходимости) | plic.c | Контроллер прерываний PLIC. | Содержит функции назначения и разрешения обработчиков прерываний. |
| Системные файлы (не трогайте без необходимости) | tmr32.c / tmr32.h | Драйвер 32-битного таймера. | Используется для настройки периодических событий и пробуждения по таймеру. |
| Системные файлы (не трогайте без необходимости) | gpio.c / gpio.h | Драйвер GPIO. | Нужен для управления светодиодами и приема внешнего сигнала пробуждения. |

Таблица 5.5 Директории проекта

| Директория | Назначение | Содержит |
|---|---|---|
| .vscode/ | Настройки среды разработки VSCode. | launch.json - настройки отладчика; tasks.json - задачи для сборки; settings.json - параметры редактора. |
| build/ | Директория для сгенерированных файлов. | .o - результат компиляции; .elf - исполняемый файл; .hex - файл прошивки; .dump - файл для отладки. |
| device/Include/ | Заголовочные файлы (.h). | K1921VG015.h - основной файл с определениями регистров МК. |
| device/Ldscripts/ | Скрипты линковщика. | k1921vg015_flash.ld; k1921vg015_ram.ld; k1921vg015.common.lds. |
| device/Source/ | Исходные системные файлы. | Системная инициализация, startup-код и базовые драйверы. |

## ПОДГОТОВКА ОБОРУДОВАНИЯ И ПРОГРАММНОЙ СРЕДЫ

### Подключение платы и конфигурация питания

1) Подключите плату К1921ВГ015 к компьютеру с помощью USB-кабеля. Порт USB Type-C отмечен на рисунке 5.4;




Рисунок 5.3. Подключение платы




Рисунок 5.4. Положение перемычек

2) Установите перемычку USB 5V JACK в положение питания от USB;

3) Установите перемычку DBG MAIN REG *(рис. 5.3)* в положение питания микроконтроллера от программатора *(рис. 5.4)*

4) Проверьте, что перемычка FT_RST *(рис. 5.4)* установлена (сброс программатора снят);

5) Установка драйверов и настройка Visual Studio Code подробно описана в ПР1.

**Установка SDK и Toolchain**

1) После установки расширения откройте панель `NIIET Aspect`. Из этого меню можно загрузить SDK, компилятор и инструменты отладки.

2) Для установки компонентов через расширение используйте пункт Download toolchain и выберите, что требуется: SDK, компилятор или отладочные инструменты.

3) Если архивы устанавливаются вручную, откройте Settings (**Ctrl+,**), введите в поиске niiet-aspect и задайте пути к SDK, компилятору, make и OpenOCD.

4) Без установленного SDK расширение не сможет создать новый проект и открыть пример.

## СОЗДАНИЕ, СБОРКА И ПРОШИВКА ПРОЕКТА

### Создание нового проекта на основе примера

**Это обязательный шаг – нельзя просто открыть папку из архива Lab7.**

1) Откройте VS Code;

2) В левой панели нажмите NIIET Aspect → Open Example;

3) Выберите пример: `pmu (bare)` (рис. 5.5);

4) В параметре `OpenOCD` Debugger interface выберите: `onboard_ftdi` – это включает встроенный отладчик платы;

5) Нажмите Create и дождитесь, пока проект откроется.

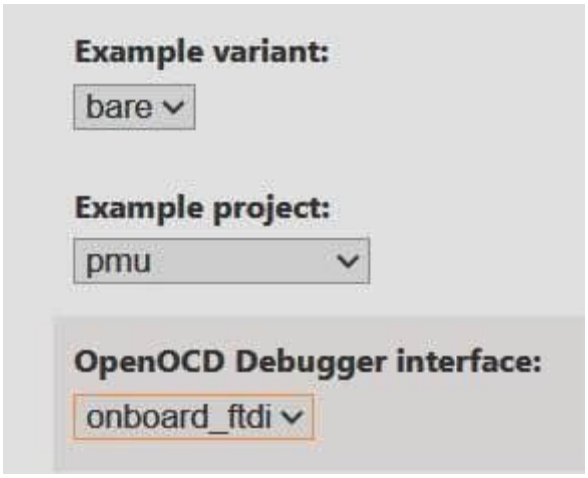



Рисунок 5.5. Создание проекта на основе примера

Замена файла `main.c`

1) В VS Code откройте папку проекта;

2) Найдите файл: `main.c`;

3) Удалите его или переименуйте;

4) Откройте архив Lab7;

5) Скопируйте файл `main.c` из архива в папку проекта;

6) Сохраните файл.

Теперь ваш проект соответствует лабораторной работе.

### Сборка проекта

1) Сборку проекта выполните через кнопку Compile project на панели NIIET Aspect или через соответствующую задачу VS Code;




Рисунок 5.6. Компиляция проекта

2) После успешной компиляции в каталоге проекта создается исполняемый файл проекта с расширением .elf;

3) При необходимости также могут формироваться вспомогательные файлы сборки (.hex, .dump и др.);

4) Если сборка завершилась без ошибок, проект готов к прошивке и отладке;

5) Нажимаем на `Build Project`.

**Прошивка микроконтроллера**

1) Убедитесь, что для проекта выбрана отладочная конфигурация Debug NIIET TRNG;

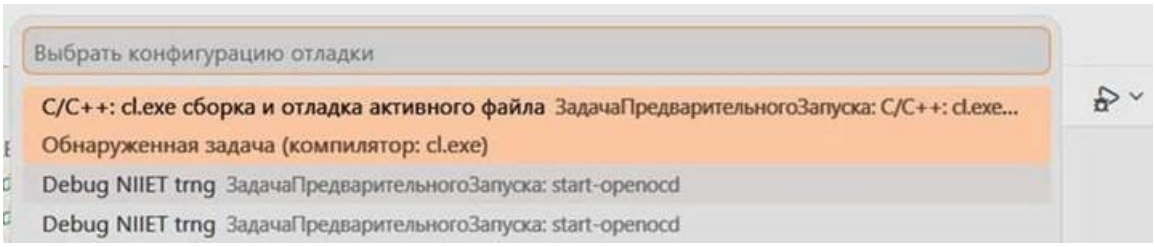



Рисунок 5.7. Режим отладки

2) Для прошивки микроконтроллера запустите эту конфигурацию из списка отладки VS Code;

3) Дождитесь завершения загрузки программы в память микроконтроллера;

4) После запуска отладки можно использовать точки останова и пошаговое выполнение программы;

5) Нажать кнопку **Reset**.

**ПРАКТИЧЕСКИЕ ЗАДАНИЯ**

**Общие правила для всех вариантов**

1. Файл `main.c` не переписывать. Дописывать код только:

· внутри `while(1) { … }` (основное место);

· и/или добавить свои функции выше `main()`.

**2. Базовая логика лабораторной работы, которую необходимо учитывать при выполнении любого варианта:**

• Включить блок TRNG и настроить периоды прогрева и выборки (регистры `WARMPERIOD`, `SAMPERIOD` и др.);

• Настроить управляющий регистр TRNG: запуск генератора случайных чисел, программный сброс, базовая конфигурация без использования тестов и прерываний;

• В основном цикле программы выполнять опрос уровня заполнения FIFO (`FIFOLEV`) и регистра состояния `STAT`;

• Не использовать программные задержки как замену проверке готовности данных;

• Для пакетного чтения считывать 8 слов подряд только после проверки, что FIFO накопил достаточное количество данных;

• Для диагностических сценариев допускается читать данные по одному слову с обязательным выводом `FIFOLEV` до и после чтения;

• Поддерживать интерфейс пользователя через UART: команда `r` - пакетное чтение 8 слов, команда `s` - отображение STAT и FIFOLEV;

• В вариантах с кольцевым буфером и журналом разрешается добавить собственные команды UART (например, b, n, o, j), но базовые команды `r` и s должны сохраняться;

• Все действия выполнять по polling, а вывод через UART использовать как основной способ доказать корректную работу TRNG и FIFO.

```
Random numbers:
                0x1D623393 0x0C25F56D 0x6605286
2 0xA27EE9E4 0xA0CD7DE9 0x707D8C3E 0x64D55C49
0x6F891C81

           Status: 0x00000084
                           FIFO level: 32
```

Рисунок 5.8. Пример вывода в UART

Задание 7.1. *Кольцевой буфер* для случайных слов

**Цель:** реализовать кольцевой буфер в ОЗУ для временного хранения случайных слов, считанных из аппаратного FIFO TRNG.

**Пояснение:**

В этом варианте не требуется дублировать аппаратный FIFO TRNG «ради FIFO». Задача состоит в том, чтобы показать типичную для встроенных систем схему: TRNG выдает данные пакетами, а программа складывает их в кольцевой буфер и затем выдает пользователю в удобном темпе. Это позволяет отдельно наблюдать работу аппаратного FIFO и программной буферизации в ОЗУ.

**Ход работы:**

1. Реализовать кольцевой буфер на 16 или 32 слова и завести для него `head`, `tail` и `count`;

2. По команде r считывать пакет из 8 слов из аппаратного FIFO TRNG и помещать их в кольцевой буфер;

3. По отдельной команде UART (например, n) выдавать одно слово из кольцевого буфера, а по команде `b` – печатать состояние буфера;

4. Предусмотреть корректную обработку состояний «буфер пуст» и «буфер полон».

**Ожидаемый результат:**

• Пакетное чтение из TRNG и поэлементная выдача через UART работают независимо;

• По UART видно текущее состояние `head`, `tail`, `count` и сохранение порядка слов;

• Студент демонстрирует понимание различия между аппаратным FIFO TRNG и кольцевым буфером в оперативной памяти.

**Задание 7.2. Вывод статистики случайных чисел**

**Цель:** показать практическое применение TRNG для простого анализа случайных данных.

**Пояснение:**

Вариант направлен не на сам факт чтения данных, а на их последующую обработку. После накопления выборки из нескольких десятков или сотен слов студент должен продемонстрировать, что аппаратный TRNG можно использовать как источник входных данных для вычислений и статистики.

**Ход работы:**

1. Считать из TRNG несколько десятков или сотен чисел и сохранить ее в массив;
2. Подсчитать количество четных и нечетных значений;
3. Найти диапазон значений и частоту повторений;
4. Выводить результаты через UART.

**Ожидаемый результат:**

- Через UART выводится понятный статистический отчет по реально считанным случайным числам;
- Пользователь видит объем выборки, диапазон значений, количество четных/нечетных и сведения о повторениях;
- Обработка выполняется после корректного чтения данных из TRNG без нарушения логики polling.

**Задание 7.3. Журнал пакетов TRNG**

**Цель:** организовать журнал успешно считанных пакетов случайных слов и использовать его для диагностики работы TRNG.

**Пояснение:**

Вместо искусственной алгоритмической обработки в этом варианте студент наблюдает сам процесс обмена с TRNG. После каждого успешного пакетного чтения программа сохраняет краткую запись в журнал: номер пакета, уровень FIFO перед чтением, уровень FIFO после чтения, значение `STAT` и, например, первое либо последнее слово пакета. Такой сценарий лучше показывает общую картину работы блока.

**Ход работы:**

1. Реализовать в ОЗУ массив записей журнала фиксированного размера, например на 8 или 16 пакетов;

2. После каждого чтения 8 слов добавлять в журнал новую запись: номер пакета, `FIFOLEV` до чтения, `FIFOLEV` после чтения, `STAT` и одно контрольное слово;

3. По отдельной команде UART (например, j) выводить накопленный журнал в табличном или компактном текстовом виде;

4. При заполнении журнала можно использовать кольцевую логику замещения старых записей новыми.

**Ожидаемый результат:**

- Пользователь видит не только сами случайные числа, но и историю успешных обращений к TRNG;
- По журналу можно проверить, как программа фиксирует состояние FIFO и `STAT` в момент чтения;

**Задание 7.4. Покомпонентное опустошение FIFO**

**Цель:** показать, как меняется уровень аппаратного FIFO TRNG при чтении данных по одному слову.

**Пояснение:**

В базовом сценарии пользователь часто видит только готовый пакет из 8 слов и уровень FIFO, равный 8. Из-за этого неочевидно, зачем вообще нужен контроль `FIFOLEV`. В этом варианте нужно специально организовать «выплевывание» данных по одному слову и наблюдать, как меняется `FIFOLEV` после каждого чтения.

**Ход работы:**

1. Добавить команду UART (например, o), которая считывает ровно одно слово из `DATA`, только если `FIFOLEV` больше нуля;

2. Перед чтением и сразу после него выводить `FIFOLEV` и `STAT`, а также само считанное слово;

3. Повторить чтение по одному слову до полного опустошения FIFO и зафиксировать момент, когда `FIFOLEV` становится равным нулю;

4. Сравнить этот режим с пакетным чтением по команде r и описать разницу в поведении.

**Ожидаемый результат:**

• По UART отчетливо видно понижение `FIFOLEV` при каждом одиночном чтении;

• Пользователь наблюдает, что аппаратный FIFO - это реально изменяющийся буфер данных;

• Студент демонстрирует понимание различия между пакетным режимом и поэлементным чтением из TRNG.

## КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Для чего в микроконтроллере К1921ВГ015 используется аппаратный блок TRNG?

2. Зачем перед запуском TRNG задаются параметры прогрева и выборки?

3. Почему в лабораторной работе чтение данных из TRNG выполняется по polling, а не по прерываниям?

4 Какую роль играет внутренний FIFO блока TRNG?

5. Для чего программе нужен регистр `FIFOLEV`?

6. Что следует проверить перед считыванием пакета из 8 случайных слов подряд?

7. Зачем выводить через UART не только случайные числа, но и регистр состояния `STAT`?

8. Чем отличается аппаратный FIFO блока TRNG от кольцевого буфера, реализованного в ОЗУ?

9. Зачем в варианте с журналом пакетов фиксировать `FIFOLEV` и `STAT` до/после чтения?

10. Что позволяет увидеть режим покомпонентного опустошения FIFO по сравнению с пакетным чтением?

## ТЕСТОВЫЕ ВОПРОСЫ

Выберите один правильный ответ.

**1. Какой аппаратный блок изучается в лабораторной работе №7?**

**А) PMU.**

**Б) TRNG.**

**В) CRC.**

**2. Какой регистр задает период прогрева генератора случайных чисел?**

А) `WARMPERIOD`.

Б) `FIFOLEV`.

**В) IC.**

3. Что показывает регистр `FIFOLEV`?

А) Уровень заполнения FIFO.

Б) Частоту UART.

В) Напряжение питания платы.

**4. Какая базовая команда по UART должна считывать пакет из 8 случайных слов?**

А) s.

Б) r.

В) t.

**5. Что должно происходить по команде s?**

А) Полный сброс Flash-памяти.

Б) Вывод `STAT` и текущего уровня FIFO.

В) Изменение тактовой частоты ядра.

6. Когда допустимо читать данные из регистра `DATA` для пакетного чтения?

А) Только после проверки, что FIFO содержит достаточное количество данных.

Б) В любой момент без проверки статуса.

В) Только после нажатия кнопки RESET.

**7. Какая практика обязательна для всех вариантов работы?**

А) Проверять статус и уровень FIFO перед чтением данных.

Б) Полностью переписывать системные файлы проекта.

В) Отказываться от использования UART.

**8. Для чего в варианте 7.1 используется кольцевой буфер?**

А) Для независимой выдачи ранее считанных слов в удобном темпе.

Б) Для изменения напряжения питания микроконтроллера.

В) Для выбора скрипта линкера.

**9. Какие параметры обычно описывают состояние кольцевого буфера?**

А) `head`, `tail` и `count`.

Б) `WARMPERIOD`, REGEN и ALARMRST.

В) PLL, LDO и BOR.

**10. Что позволяет показать режим покомпонентного «выплевывания» данных из FIFO?**

А) Как меняется содержимое FIFO при последовательном чтении слов, а не только его уровень.

Б) Как автоматически меняется частота `UART0`.

В) Как TRNG переводит микроконтроллер в режим сна.

**11. Что фиксируется в журнале пакетов варианта 7.3?**

А) Данные о каждом успешном чтении пакета из TRNG.

Б) Только цвет интерфейса VS Code.

В) Только имя пользователя Windows.

12. Почему в вариантах работы недостаточно наблюдать только постоянное значение `FIFOLEV` = 8?

А) Потому что по одному уровню заполнения не видно, как меняется содержимое FIFO при чтении отдельных слов.

Б) Потому что значение `FIFOLEV` всегда показывает температуру TRNG.

В) Потому что `FIFOLEV` используется только для настройки UART.

**13. Что должна делать команда одиночного чтения в варианте 7.4?**

А) Считывать одно слово только при `FIFOLEV` > 0 и печатать `FIFOLEV` до и после чтения.

Б) Сразу очищать всю память SRAM.

В) Переключать режим питания платы.

14. Почему в лабораторной работе не следует заменять контроль `FIFOLEV` программными задержками?

А) Задержки не гарантируют реальную готовность данных TRNG.

Б) Задержки автоматически увеличивают энтропию.

В) Задержки меняют разрядность регистра `DATA`.

**15. Что позволяет увидеть режим чтения по одному слову?**

А) Как `FIFOLEV` уменьшается после каждого чтения.

Б) Только температуру кристалла.

В) Только скорость JTAG.

**16. Какой интерфейс в работе используется как пользовательский канал управления и вывода?**

А) `UART0`.

Б) JTAG.

В) CRC.

**17. Какой файл является основным для написания логики лабораторной работы?**

А) `main.c`.

Б) startup_k1921vg015.S.

В) config.json.

**18. Какой принцип должен сохраняться даже при добавлении новых команд UART?**

А) Базовые команды `r` и s должны оставаться рабочими.

Б) Нужно удалить вывод `STAT`.

В) Нужно отказаться от polling.

19. Для чего в варианте 7.3 полезно сохранять `FIFOLEV` до и после чтения пакета?

А) Чтобы диагностировать реальное поведение FIFO при обмене с TRNG.

Б) Чтобы измерять напряжение USB.

В) Чтобы выбирать режим PowerOff.

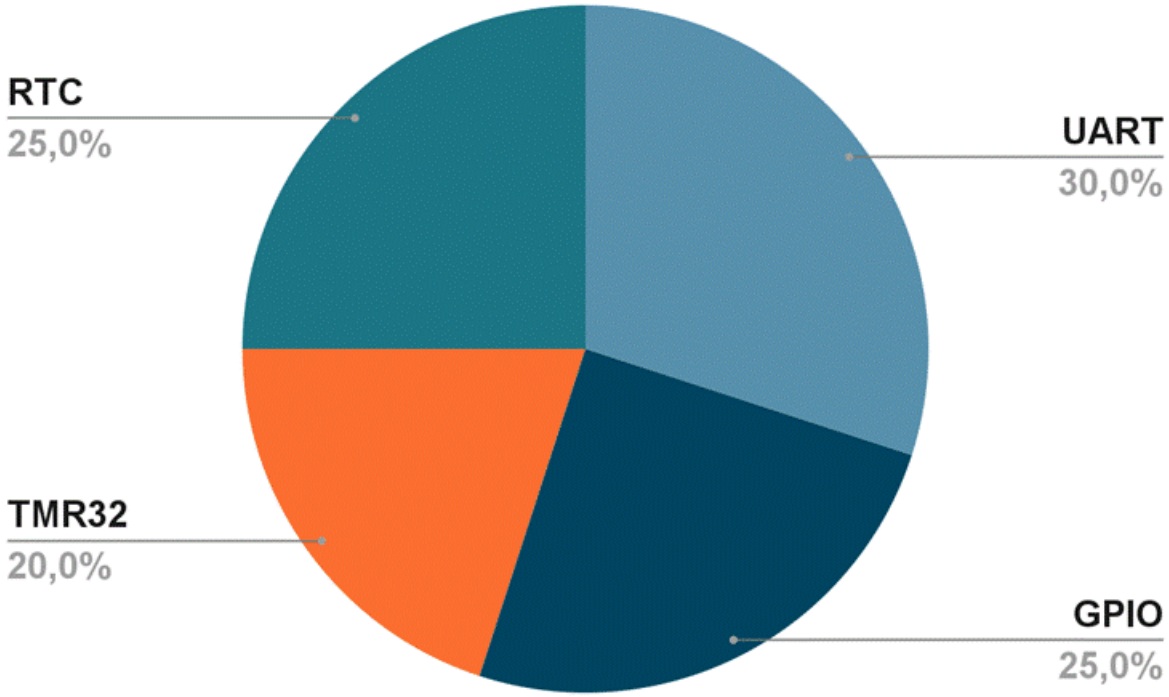
**20. Что является корректным признаком готовности пакета из 8 слов?**

А) `FIFOLEV` показывает наличие не менее 8 слов.

Б) Прошло ровно 10 секунд после старта.

В) Светодиод на плате один раз мигнул.

# Часть III. Коммуникации и хранение: Диалог с внешним миром




Распределение использования периферийных модулей микроконтроллера К1921ВГ015

# Глава 6. Текстовый интерфейс: UART и DMA

## 1.UART и консольный интерфейс

UART представляет собой последовательный интерфейс обмена данными, широко применяемый для отладки встраиваемых систем. При использовании `retarget` стандартные функции ввода-вывода, в частности `printf()`, `retarget`_put_char() и `retarget`_get_char(), связываются с аппаратным UART, что позволяет организовать текстовую консоль без дополнительного графического интерфейса.

В исследуемом проекте консоль применяется для приема одиночных символьных команд и отображения текстовых сообщений о состоянии программы. Такой подход удобен для проверки логики управления светодиодами, таймером и часами реального времени.

## 2. Порты GPIO и светодиоды PA12–PA15

GPIO – универсальные линии ввода-вывода микроконтроллера. В лабораторной работе светодиоды подключены к выводам PA12–PA15, поэтому для их использования необходимо включить тактирование порта GPIOA и перевести соответствующие линии в режим цифрового выхода.

В программе применяется маска LEDS_MASK = 0xFF00, охватывающая восемь светодиодов. Дополнительно введены отдельные маски LED12_MASK … LED15_MASK, что упрощает последовательное управление каждым светодиодом при формировании эффекта «бегущий огонь».

## 3.Таймер TMR32 и эффект «бегущий огонь»

Бегущий огонь реализуется за счет периодического переключения активного светодиода. Для формирования устойчивого интервала используется 32-битный таймер TMR32. В обработчике его прерывания состояние текущего светодиода инвертируется, после чего активная битовая маска сдвигается к следующей позиции.

Период срабатывания определяется значением регистра CAPCOM[0].VAL. В базовом варианте таймер инициализируется значением SystemCoreClock / 8, что формирует визуально различимый темп переключения.

## 4. RTC и хранение времени

RTC (Real-Time Clock) служит для ведения непрерывного отсчета времени. В архиве проекта RTC настраивается на секундный тик 1 Hz от внешнего кварцевого генератора 32,768

кГц. Начальное время устанавливается программно и затем хранится в регистре RTC_TIME в виде количества секунд от выбранной точки отсчета.

В представленном коде в качестве начала отсчета используется 2000 год, а текущее время хранится и обрабатывается в секундах. Такой способ удобен для математических преобразований и дальнейшего вывода календарной даты.

### 5. Преобразование времени и формат вывода

Для пользовательского отображения значение RTC_TIME необходимо преобразовать в календарные поля: год, месяц, день, час, минуту и секунду. С этой целью в программе введена структура datetime_t и реализованы функции datetime_to_unix() и unix_to_datetime().

Вывод времени осуществляется строкой формата YYYY-MM-DD HH:MM:SS. Такой формат однозначен, удобен для визуального контроля и соответствует условиям задания.

### 6. Организация обработки команд

Обработка команд строится вокруг бесконечного цикла `main()`, в котором выполняются программный вызов RTC_IRQHandler(), чтение символа из UART и разбор команды в операторе switch. Команды 1 и 0 запускают и останавливают бегущий огонь, команда t выводит текущее время, команда h печатает справку по доступным командам.

Подобная архитектура позволяет расширять функциональность без изменения общей структуры программы: новые команды могут быть добавлены в блок switch, а дополнительные режимы работы – оформлены как отдельные функции.

### 7. Терминальные программы PuTTY и SmarTTY

Для обмена данными с микроконтроллером по интерфейсу UART применяются терминальные программы, в частности PuTTY и SmarTTY. Программа PuTTY поддерживает работу в режиме последовательного соединения, при котором пользователь указывает номер последовательного порта и скорость обмена, после чего открывает терминальную сессию. Программа SmarTTY также позволяет устанавливать соединение с устройством через COM-порт и содержит встроенный терминал для последовательного интерфейса. Она поддерживает просмотр данных в текстовом и шестнадцатеричном виде, а также сохранение журнала обмена.

Использование указанных программ позволяет контролировать работу разработанной UART-консоли, передавать команды с клавиатуры и наблюдать текстовые сообщения, формируемые прошивкой микроконтроллера. Такие средства удобны при отладке встроенных систем, поскольку обеспечивают простой способ взаимодействия с программой без применения дополнительных графических интерфейсов.

## 8. Основные параметры интерфейса UART

UART (Universal Asynchronous Receiver/Transmitter) представляет собой универсальный асинхронный интерфейс последовательного обмена данными. Передача по UART выполняется без отдельной линии тактирования, поэтому передающее и принимающее устройства должны быть заранее настроены на одинаковые параметры обмена. Формат кадра UART включает стартовый бит, информационные биты, при необходимости бит четности и один или два стоп-бита. Наиболее распространенным форматом является 8N1, то есть 8 бит данных, без контроля четности и 1 стоп-бит.

Одним из основных параметров UART является baud rate – скорость передачи символов по последовательному каналу. На практике в настройках терминальной программы и прошивки микроконтроллера это значение должно совпадать, иначе принимаемые данные будут искажаться либо не будут распознаваться вовсе. Помимо скорости обмена, при настройке UART учитываются следующие параметры: количество бит данных, наличие и тип бита четности, число стоп-битов и способ управления потоком. В терминальных программах данные параметры задаются явно в окне настройки последовательного соединения.

Биты данных определяют количество информационных разрядов в одном передаваемом символе. Чаще всего используются 8 бит. Бит четности служит для простого контроля ошибок и может иметь режимы none, even или odd. Стоп-биты обозначают завершение передачи символа; обычно используется один стоп-бит. Управление потоком применяется для согласования скорости передачи между устройствами и может быть аппаратным или программным, однако в простых лабораторных работах по UART обычно используется режим без управления потоком.

Одним из основных параметров интерфейса UART является baud rate – скорость обмена по последовательной линии. В учебных и практических задачах под этим параметром обычно понимают количество бит, передаваемых за одну секунду. Чем выше значение baud rate, тем быстрее осуществляется передача данных, однако для устойчивого обмена необходимо, чтобы одинаковая скорость была установлена и в прошивке микроконтроллера, и в терминальной программе на компьютере. При несовпадении параметров принимаемые символы отображаются некорректно либо обмен не устанавливается вовсе.

К числу наиболее распространенных скоростей UART относятся 1200, 2400, 4800, 9600, 19200, 38400, 57600 и 115200 бод. В лабораторных работах часто используются стандартные значения, так как они поддерживаются большинством микроконтроллеров, USB-UART преобразователей и терминальных программ. Значение 115200 бод применяется достаточно

часто, поскольку обеспечивает более высокую скорость обмена по сравнению с 9600 и 57600, оставаясь при этом стандартным и широко поддерживаемым режимом.

# Практическая работа 6

**UART-консоль управления бегущим огнем и временем RTC**

**СОДЕРЖАНИЕ**

1. Цель и оборудование
3. Структура проекта и основные файлы
4. Задание
5. Ход работы
5.1 Подготовка проекта и анализ исходных файлов
5.2 Настройка светодиодов и GPIOA
5.3 Настройка таймера TMR32
5.4 Настройка RTC и секундного тика
5.5 Реализация UART-консоли и команд
5.6 Организация основного цикла программы
5.7 Сборка, прошивка и проверка работы
5.8 Настройка терминальной программы и проверка вывода
6. Ожидаемый результат
7. Практические задания
7.1 Общие правила для всех вариантов
7.2 Базовый вариант. Консоль управления
7.3 Вариант 1. Управление частотой бегущего огня
7.4 Вариант 2. Меню и настройка времени
7.5 Вариант 3. Расширенные функции RTC
7.6 Вариант 4. Двоичные часы
8. Контрольные вопросы
9. Тестовые вопросы
10. Приложение. Пример исходного кода (main.c)

## ЦЕЛЬ И ОБОРУДОВАНИЕ

**Цель:** освоить разработку UART-консоли для микроконтроллера К1921ВГ015, получить практические навыки управления светодиодной индикацией через GPIO и таймер TMR32, а также научиться выводить текущее время из RTC в формате YYYY-MM-DD HH:MM:SS.

**Оборудование:** макетно-отладочная плата с микроконтроллером К1921ВГ015; персональный компьютер; USB-кабель Type-C для подключения платы; среда разработки Visual Studio Code или иная IDE, совместимая с SDK; UART-терминал для наблюдения результатов работы программы.

## СТРУКТУРА ПРОЕКТА И ОСНОВНЫЕ ФАЙЛЫ

Архив проекта содержит пользовательские и системные файлы, необходимые для сборки и запуска программы на микроконтроллере К1921ВГ015.

main.c – основной пользовательский файл, в котором реализованы инициализация периферии, обработчики событий, функции преобразования времени и логика UART-консоли.

makefile – сценарий автоматической сборки проекта, формирующий исполняемый файл .elf, загрузочный файл .hex и отладочный дамп .dump.

niiet_aspect.json и device/config.json – конфигурационные файлы инструментальной среды.

device/Include и device/Source – каталоги системных заголовочных и исходных файлов, содержащих определения регистров микроконтроллера, код startup, системную инициализацию, retarget и поддержку стандартного вывода.

build – каталог результатов сборки, включающий объектные файлы и готовые артефакты tsensor.elf, tsensor.hex и tsensor.dump.

Каталог .vscode содержит служебные файлы launch.json, tasks.json и settings.json, обеспечивающие удобную сборку и отладку проекта в Visual Studio Code.

## ЗАДАНИЕ

В базовом варианте лабораторной работы требуется реализовать консоль управления для микроконтроллера К1921ВГ015 со следующими возможностями:

1. По команде 1 запускать эффект «бегущий огонь» на светодиодах PA12–PA15.
2. По команде 0 останавливать бегущий огонь и переводить светодиоды в неактивное состояние.
3. По команде t выводить текущее время из RTC в формате YYYY-MM-DD HH:MM:SS.
4. По команде h выводить справку по доступным командам.

Программа должна быть собрана, прошита в микроконтроллер и проверена в UART-терминале. Дополнительные индивидуальные задания приведены в разделе практических вариантов.

## ХОД РАБОТЫ

### 5.1 Подготовка проекта и анализ исходных файлов

Для выполнения работы был использован архив Lab8, содержащий готовый каркас проекта и реализованный базовый вариант. После распаковки архива была проверена структура каталогов, наличие основного файла main.c, сценария сборки makefile и системных файлов микроконтроллера.

Анализ main.c показал, что программа уже содержит необходимые логические блоки: описание масок светодиодов, структуру datetime_t, функции преобразования времени, инициализацию GPIO, TMR32 и RTC, обработчик таймера, текстовый интерфейс UART и основной цикл программы.

В начале файла определены следующие маски и структура времени:

```
#define LEDS_MASK 0xFF00
#define LED4_MASK (1 << 12)
#define LED5_MASK (1 << 13)
#define LED6_MASK (1 << 14)
#define LED7_MASK (1 << 15)
typedef struct {
uint16_t year;
uint8_t month;
uint8_t day;
uint8_t hour;
uint8_t minute;
uint8_t second;
} datetime_t;
```

### 5.2 Настройка светодиодов и GPIOA

Инициализация светодиодов выполнена в функции gpio_init(). В ней разрешается тактирование порта GPIOA, линии PA12–PA15 переводятся в режим выхода, после чего формируется начальное выключенное состояние индикации.

```
void gpio_init(void) {
RCU->CGCFGAHB_bit.GPIOAEN = 1;
RCU->RSTDISAHB_bit.GPIOAEN = 1;
GPIOA->OUTENSET = LEDS_MASK;
GPIOA->DATAOUTSET = LEDS_MASK;
}
```

Таким образом подготавливается аппаратная база для вывода бегущего огня и для световой индикации состояния программы.

### 5.3 Настройка таймера TMR32

В функции timer_init(uint32_t period) включается тактирование таймера TMR32, устанавливается период по регистру CAPCOM[0].VAL, разрешается прерывание по совпадению и выполняется регистрация обработчика в контроллере прерываний PLIC.

```
void timer_init(uint32_t period) {
RCU->CGCFGAPB_bit.TMR32EN = 1;
RCU->RSTDISAPB_bit.TMR32EN = 1;
TMR32->CAPCOM[0].VAL = period - 1;
TMR32->CTRL_bit.MODE = 1;
    TMR32->IM = 2;
PLIC_SetIrqHandler(Plic_Mach_Target, IsrVect_IRQ_TMR32, TMR32_IRQHandler);
    PLIC_SetPriority(IsrVect_IRQ_TMR32, 1);
    PLIC_IntEnable(Plic_Mach_Target, IsrVect_IRQ_TMR32);
TMR32->CTRL &= ~0x1;
    }
```

Сам эффект «бегущий огонь» реализован в TMR32_IRQHandler(). При активном флаге running_light_enabled происходит инвертирование текущего светодиода, сдвиг битовой маски влево и переход к следующему светодиоду. После обработки выполняется сброс флага прерывания.

```
    void TMR32_IRQHandler(void) {
    if(running_light_enabled) {
GPIOA->DATAOUTTGL = led_pattern;
led_pattern <<= 1;
    if(led_pattern > LED7_MASK) {
led_pattern = LED0_MASK;
    }
GPIOA->DATAOUTTGL = led_pattern;
    }

    TMR32->IC = 3;
    }
```

### 5.4 Настройка RTC и секундного тика

Функция RTC_init() конфигурирует RTC на работу от внешнего кварца 32,768 кГц, задает секундный тик 1 Hz и программно устанавливает стартовое время 2026-01-21 17:05:00. После этого RTC переводится в рабочий режим.

```
    void RTC_init(void)
    {
```

```
        PMURTC->RTC_CFG0 |= (1 << 4);
for (volatile uint32_t i = 0; i < 10000; i++);
uint32_t reg_value = PMURTC->RTC_CFG0;
reg_value &= ~0x03;
reg_value |= 0x01;
PMURTC->RTC_CFG0 = reg_value;
        datetime_t init_time = {
        .year = 2026,
        .month = 1,
        .day = 21,
        .hour = 17,
        .minute = 5,
        .second = 0
        };
PMURTC->RTC_TIME = datetime_to_unix(&init_time);
        PMURTC->RTC_CFG0 |= (1 << 1);
        PMURTC->RTC_CFG1 |= (1 << 0);
        PMURTC->RTC_HISTORY = 0;
        }
```

В проекте используется программный опрос RTC через функцию RTC_IRQHandler(), которая вызывается из основного цикла. Внутри функции сравнивается текущее и предыдущее значение RTC_TIME; при изменении фиксируется новый секундный тик и устанавливается программный флаг time_tick.

```
        void RTC_IRQHandler(void)
        {
static uint32_t last_rtc_time = 0;
uint32_t current_rtc_time = PMURTC->RTC_TIME;
if (current_rtc_time != last_rtc_time)
        {
PMURTC->RTC_HISTORY = 0;
time_tick = 1;
last_rtc_time = current_rtc_time;
        }
        }
```

### 5.5 Реализация UART-консоли и команд

Текстовый интерфейс формируется функцией print_intro(), которая выводит название консоли, значение SystemCoreClock и список доступных команд. После инициализации периферии эта функция вызывается один раз, обеспечивая пользователю стартовую подсказку.

```
void print_intro(void) {
```

```
    printf("\r\n\r\n=== UART Console for K1921VG015 ===\r\n");
    printf("SystemCoreClock: %lu Hz\r\n", SystemCoreClock);
    printf("Commands:\r\n");
    printf("  '1' - Start running light\r\n");
    printf("  '0' - Stop running light\r\n");
    printf("  't' - Show current time\r\n");
    printf("  'h' - Show this help\r\n");
    printf("> ");
}
```

Разбор команд выполняется в основном цикле по символу, считанному функцией retarget_get_char(). Для известных команд формируются соответствующие действия и сообщения в терминал.

```
switch(c) {
    case '1':
        printf("\r\nStarting running light...\r\n");
        running_light_enabled = 1;
        TMR32->CTRL |= 0x1;
        printf("> ");
        break;

    case '0':
        printf("\r\nStopping running light...\r\n");
        running_light_enabled = 0;
        TMR32->CTRL &= ~0x1;
        GPIOA->DATAOUTSET = LEDS_MASK;
        printf("> ");
        break;

    case 't':
    case 'T': {
        printf("\r\nCurrent time: ");
        uint32_t curr_time = PMURTC->RTC_TIME;
```

```
        datetime_t dt;
        unix_to_datetime(curr_time, &dt);
        printf("%04u-%02u-%02u %02u:%02u:%02u\r\n> ",
              dt.year, dt.month, dt.day,
              dt.hour, dt.minute, dt.second);
        break;
    }

    case 'h':
    case 'H':
        print_intro();
        break;
}
```

Таким образом реализован простой и наглядный консольный интерфейс, достаточный для выполнения базового варианта лабораторной работы.

**5.6 Организация основного цикла программы**

После вызовов SystemInit(), SystemCoreClockUpdate(), retarget_init(), gpio_init(), timer_init() и RTC_init() программа разрешает глобальные прерывания и переходит в бесконечный цикл. В цикле последовательно выполняются: программный вызов RTC_IRQHandler(), чтение символа из UART, обработка команды, сброс программного флага time_tick и короткая задержка delay_ms(1).

Такой подход сочетает событийную обработку таймера через аппаратное прерывание и обслуживание интерфейса пользователя через опрос UART, что делает логику проекта прозрачной и удобной для учебного анализа.

**5.7 Сборка, прошивка и проверка работы**

После анализа и подтверждения корректности исходного кода проект собирается штатными средствами makefile. Результатом успешной сборки являются файлы tsensor.elf и tsensor.hex в каталоге build.

Полученный загрузочный файл прошивается в микроконтроллер, после чего открывается UART-терминал с параметрами, соответствующими настройкам проекта. При запуске на экране появляется приветствие консоли, далее проверяются команды 1, 0, t и h.

По команде 1 запускается бегущий огонь на светодиодах PA12–PA15, по команде 0 таймер останавливается и индикация выключается, по команде t в терминал выводится текущее время, а по команде h отображается краткая справка по работе консоли.

### 5.8 Настройка терминальной программы и проверка вывода

После компиляции программы и загрузки прошивки в микроконтроллер была выполнена проверка обмена по интерфейсу UART с использованием терминальной программы PuTTY или SmarTTY. На компьютере был определен номер виртуального COM-порта, соответствующий подключенной плате. Далее была открыта терминальная программа и выбран режим подключения по последовательному интерфейсу. В программе PuTTYдля этого используется тип соединения Serial, после чего задаются параметры Serial line и Speed. Дополнительные параметры последовательного порта настраиваются в разделе Serial, где указываются количество бит данных, четность, число стоп-битов и режим управления потоком.

При использовании SmarTTY создается новое соединение через пункт Setup new serial or TCP connection, после чего выбирается нужный COM-порт и параметры обмена.

Терминал позволяет наблюдать сообщения от микроконтроллера и передавать управляющие команды с клавиатуры. При необходимости может вестись журнал обмена, что удобно при анализе работы программы.

Для корректного отображения вывода параметры терминальной программы должны полностью совпадать с параметрами UART, заданными в прошивке микроконтроллера. После открытия соединения в терминальном окне появляется консольный интерфейс программы. Если начальное приглашение не выводится автоматически, допускается передача первого символа или нажатие клавиши Enter, чтобы инициировать обмен и получить ответ от устройства. Далее выполняется ввод команд с клавиатуры, предусмотренных заданием, например включение и выключение бегущего огня, вывод текущего времени и отображение справки по доступным командам.

В ходе проверки было установлено, что после ввода команд в терминале микроконтроллер корректно принимает данные по UART и формирует ожидаемый текстовый вывод. Это подтверждает правильность настройки последовательного интерфейса, корректную работу UART-консоли и работоспособность реализованной программы управления светодиодами и временем.

### 5.9 Настройка PuTTY для работы с UART

После компиляции программы и загрузки прошивки в микроконтроллер была выполнена проверка работы UART-консоли с использованием программы PuTTY. Для этого в главном окне программы на вкладке Session в поле Connection type выбирается режим Serial. Данный режим предназначен для соединения не по сети, а с локальной последовательной линией, то есть с COM-портом компьютера, через который осуществляется обмен данными с подключенной платой. При выборе режима Serial поля Host Name и Port, используемые для сетевых подключений, заменяются полями Serial line и Speed.

В поле Serial line указывается номер последовательного порта, к которому подключена плата, например COM3, COM4 и т.д. В операционной системе Windows данный номер определяется автоматически при подключении устройства и отображается как виртуальный COM-порт. Таким образом, параметр Serial line обозначает конкретный последовательный интерфейс компьютера, через который терминальная программа будет взаимодействовать с микроконтроллером.

В поле Speed указывается значение 115200. Этот параметр задает скорость обмена по UART, то есть скорость передачи данных между компьютером и микроконтроллером. Для корректной работы терминала значение 115200должно совпадать с baud rate, установленным в программе микроконтроллера. Если скорость в PuTTY и в прошивке различается, текст в окне терминала будет выводиться с ошибками или не будет читаться.

Для корректного обмена в дополнительных параметрах последовательного соединения обычно устанавливаются следующие значения: Data bits – 8, Stop bits – 1, Parity – None, Flow control – None. Такой формат соответствует наиболее распространенной конфигурации 8N1, используемой при работе с UART. После задания параметров открывается соединение, и в терминальном окне отображается текстовый вывод программы. Если после открытия окна данные сразу не появились, допускается нажать Enter, чтобы инициировать обмен с устройством.

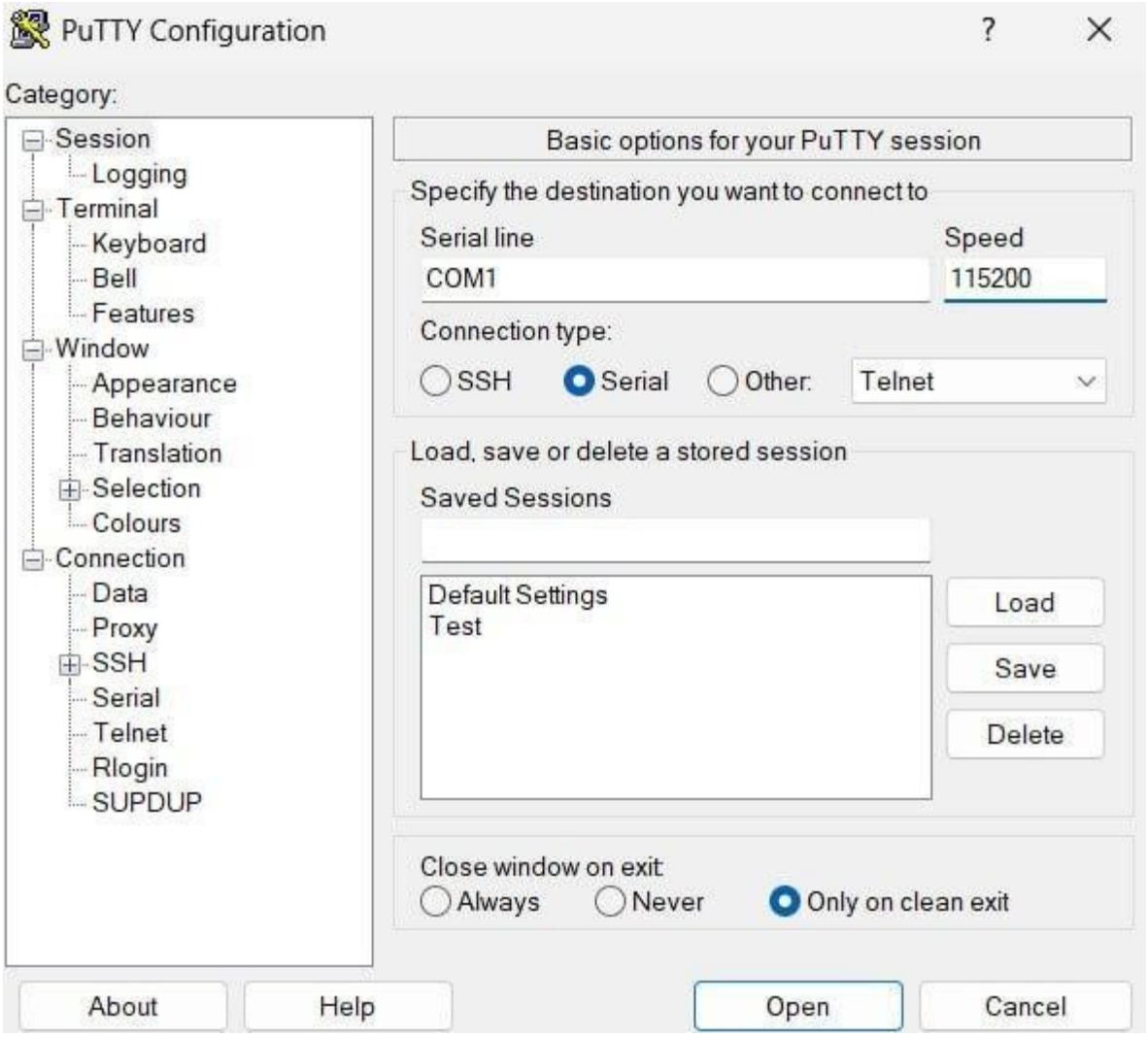



Рис. 1 – настройка PuTTY

Настройка терминальной программы PuTTY выполняется на вкладке Session. В поле Connection type выбирается режим Serial, предназначенный для работы с локальной последовательной линией. В поле Serial line указывается COM-порт, к которому подключена плата, а в поле Speed задается скорость обмена 115200 бод. Данные параметры должны соответствовать настройкам UART, заданным в прошивке микроконтроллера.

**ОЖИДАЕМЫЙ РЕЗУЛЬТАТ**

В результате выполнения работы микроконтроллер должен реализовывать устойчиво работающую UART-консоль управления. Пользователь получает возможность запускать и останавливать бегущий огонь на светодиодах, а также считывать текущее время из RTC в удобном календарном формате.

Пример фрагмента работы программы приведен на рисунке 1. На изображении видно, что команды 1 и 0 корректно запускают и останавливают бегущий огонь, команда h выводит справку по доступным командам, а команда t отображает текущее время из RTC.

```
1
Starting running light...
> 0
Stopping running light...
> h

=== UART Console for K1921VG015 ===
SystemCoreClock:  Commands:
  '1' - Start running light
  '0' - Stop running light
  't' - Show current time
  'h' - Show this help
> t
Current time: 2026-01-21 17:05:16
> t
Current time: 2026-01-21 17:05:19
> 1
Starting running light...
> 0
Stopping running light...
>
```

Рис. 2 – пример работы UART-консоли

## ПРАКТИЧЕСКИЕ ЗАДАНИЯ

### 7.1 Общие правила для всех вариантов

Каждый вариант выполняется на основе базового проекта, представленного в архиве Lab8. Допускается переработка структуры основного цикла, расширение набора команд, введение дополнительных глобальных переменных и изменение конфигурации таймера или RTC при условии сохранения корректной сборки и работоспособности UART-консоли.

Во всех вариантах требуется обеспечить корректный вывод сообщений пользователю, устойчивую работу светодиодной индикации и предсказуемое поведение программы при вводе допустимых команд.

### 7.2 Базовый вариант. Консоль управления

Базовый вариант соответствует текущей реализации проекта и предусматривает консоль управления со следующими командами:

1 – включить бегущий огонь на светодиодах a

0 – выключить бегущий огонь.

t – вывести текущее время из RTC в формате YYYY-MM-DD HH:MM:SS.

h – вывести справку по командам.

### 7.3 Вариант 1. Управление частотой бегущего огня

В данном варианте требуется реализовать регулировку скорости бегущего огня. Консоль должна поддерживать следующие команды:

1 – включить бегущий огонь.

0 – выключить бегущий огонь.

+ – увеличить скорость бегущего огня.

- – уменьшить скорость бегущего огня.

s – вывести текущую скорость, то есть значение таймера.

t – вывести текущее время.

d – переключить направление бегущего огня влево или вправо.

Дополнительно необходимо хранить в глобальной переменной текущее значение скорости, связанное с периодом таймера. При каждом изменении скорости требуется выполнять переинициализацию таймера с новым периодом.

### 7.4 Вариант 2. Меню и настройка времени

В данном варианте требуется создать иерархическое меню управления.

Главное меню должно включать следующие пункты:

l – управление светодиодами, переход в подменю.

t – управление временем, переход в подменю.

s – вывод статуса системы.

Подменю светодиодов должно поддерживать команды:

1 – режим бегущего огня.

2 – режим мерцания всех светодиодов.

3 – включить все светодиоды.

4 – выключить все светодиоды.

Подменю времени должно поддерживать команды:

c – показать текущее время.

s – установить новое время через ввод в формате YYYYMMDDHHMMSS.

a – включить или выключить автообновление времени на экране.

### 7.5 Вариант 3. Расширенные функции RTC

В данном варианте требуется реализовать часы с будильником. Консоль должна поддерживать следующие команды:

t – показать текущее время.

a – показать установленный будильник.

s – установить время будильника в формате HHMM.

e – включить или выключить будильник.

1/0 – управление светодиодами как индикатором сработавшего будильника.

r – сброс, то есть отключение сработавшего будильника.

Дополнительно при срабатывании будильника необходимо включить все светодиоды и вывести сообщение "ALARM!". В основном цикле требуется реализовать функцию проверки срабатывания будильника. Предусматривается возможность установки только времени без задания даты.

### 7.6 Вариант 4. Двоичные часы

В данном варианте требуется реализовать визуализацию времени на светодиодах в двоичном виде. Консоль должна поддерживать следующие команды:

t – текстовый вывод времени.

h – показать часы на светодиодах в двоичном виде в диапазоне 0–23.

m – показать минуты на светодиодах в двоичном виде в диапазоне 0–59.

s – показать секунды на светодиодах в двоичном виде в диапазоне 0–59.

1 – включить авторежим с циклическим отображением часов, минут и секунд каждые 3 секунды.

0 – выключить авторежим.

e – отображать четность секунд миганием на нечетной секунде.

Особенность варианта состоит в использовании светодиодной индикации как двоичного индикатора времени.

**КОНТРОЛЬНЫЕ ВОПРОСЫ**

1. Для чего в работе используется интерфейс UART и какие функции обеспечивают обмен символами с терминалом?

2. Почему светодиоды, подключенные к линиям PA12–PA15, удобно описывать битовыми масками?

3. Какое назначение имеет таймер TMR32 в реализации эффекта «бегущий огонь»?

4. Почему управление переключением светодиодов удобно выполнять в обработчике прерывания таймера?

5. Для чего в программе реализованы функции datetime_to_unix() и unix_to_datetime()?

6. Каким образом в проекте организован секундный тик RTC и почему применяется внешний кварц 32,768 кГц?

7. Чем отличается аппаратное прерывание таймера от программного опроса RTC в основном цикле?

8. Какие действия выполняются по командам 1, 0, t и h в базовом варианте лабораторной работы?

9. Для чего после остановки бегущего огня выполняется запись GPIOA->DATAOUTSET = LEDS_MASK?

10. Какие преимущества и ограничения имеет консольный интерфейс на основе одиночных символьных команд?

## ТЕСТОВЫЕ ВОПРОСЫ

Выберите один правильный ответ.

**1. Какой интерфейс используется в работе для текстового взаимодействия с пользователем?**

А) UART.

Б) SPI.

В) I2C.

**2. Какие линии порта GPIOA используются для бегущего огня в базовом варианте?**

А) PA8–PA15.

Б) PA0–PA7.

В) PB12–PB15.

**3. Какой регистр таймера определяет период переключения светодиодов?**

А) CAPCOM[0].VAL.

Б) RTC_TIME.

В) DATAOUTSET.

**4. Какая переменная используется как программный флаг секундного тика RTC?**

А) time_tick.

Б) led_pattern.

В) SystemCoreClock.

**5. Какая функция преобразует число секунд в календарную дату и время?**

А) unix_to_datetime().

Б) delay_ms().

В) gpio_init().

**6. Какая команда выводит справку по доступным командам?**

А) h.

Б) 0.

В) d.

**7. Какое действие выполняется по команде 1?**

А) Запуск бегущего огня и запуск таймера.

Б) Установка нового времени в RTC.

В) Сброс содержимого RTC_TIME.

**8. Какой регистр RTC хранит текущее время в секундах?**

А) RTC_TIME.

Б) RTC_HISTORY.

В) RTC_CFG1.

**9. Для чего в конце обработки TMR32_IRQHandler() выполняется запись TMR32->IC = 3?**

А) Для сброса флага прерывания таймера.

Б) Для установки нового времени в RTC.

В) Для изменения направления бегущего огня.

**10. Что описывает структура datetime_t?**

А) Календарные поля даты и времени.

Б) Параметры настройки UART.

В) Битовую маску всех светодиодов.

**11. Какая команда в UART-консоли запускает эффект «бегущий огонь»?**

А) t

Б) l

В) h

**12. Какая команда останавливает эффект «бегущий огонь»?**

А) 0

Б) s

В) q

**13. Какая команда выводит текущее время в терминал?**

А) h

Б) l

В) t

**14. Для чего используется команда h в консольном интерфейсе программы?**

А) Для очистки буфера UART

Б) Для вывода списка доступных команд

В) Для запуска передачи данных

**15. Какой интерфейс используется для обмена данными между платой и терминальной программой?**

А) SPI

Б) UART

В) I2C

**16. Что необходимо выбрать в PuTTY для подключения к плате по UART?**

А) SSH

Б) Telnet

В) Serial

**17. Что указывается в поле Serial line терминальной программы?**

А) IP-адрес платы

Б) Номер COM-порта, к которому подключена плата

В) Название исходного файла программы

**18. Почему параметры терминала должны совпадать с настройками прошивки микроконтроллера?**

А) Иначе программа не скомпилируется

Б) Иначе символы в терминале будут отображаться некорректно

В) Иначе отключится питание платы

**19. Что означает baud rate в настройках UART?**

А) Размер буфера памяти

Б) Скорость обмена данными по последовательному интерфейсу

В) Количество светодиодов на плате

**20. Какой результат подтверждает корректную работу программы в ЛР8?**

А) В терминале отсутствуют любые сообщения

Б) Команды из терминала обрабатываются, бегущий огонь запускается и останавливается, текущее время выводится корректно

В) Плата работает только без подключения UART

**ПРИЛОЖЕНИЕ. ПРИМЕР ИСХОДНОГО КОДА (MAIN.C)**

```
#include <K1921VG015.h>
#include <stdint.h>
#include <stdio.h>
#include <system_k1921vg015.h>
#include "retarget.h"

// Дефайны для светодиодов
#define LED4_MASK (1 << 12)
#define LED5_MASK (1 << 13)
#define LED6_MASK (1 << 14)
#define LED7_MASK (1 << 15)

// Структура для хранения даты и времени (объявлена ДО использования)
typedef struct {
    uint16_t year;
    uint8_t month;
    uint8_t day;
    uint8_t hour;
    uint8_t minute;
    uint8_t second;
} datetime_t;

// Forward declarations
void TMR32_IRQHandler(void);
void delay_ms(uint32_t ms);
void gpio_init(void);
void timer_init(uint32_t period);
void RTC_init(void);
void RTC_IRQHandler(void);
void unix_to_datetime(uint32_t unix_time, datetime_t *dt);
uint32_t datetime_to_unix(datetime_t *dt);
```

```
void print_intro(void);

#define EPOCH_YEAR 2000

// Глобальные переменные
volatile uint8_t running_light_enabled = 0;
volatile uint32_t led_pattern = LED0_MASK;
volatile uint8_t time_tick = 0;  // Программный флаг для 1-секундного тика

static uint8_t is_leap_year(uint16_t year)
{
    return ((year % 4 == 0 && year % 100 != 0) || (year % 400 == 0));
}

static const uint8_t month_days[2][12] = {
    {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31},
    {31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31}
};

void unix_to_datetime(uint32_t unix_time, datetime_t *dt)
{
    uint32_t days, seconds_in_day;
    uint16_t year = EPOCH_YEAR;
    uint8_t month = 0;
    uint8_t leap;

    days = unix_time / 86400;
    seconds_in_day = unix_time % 86400;

    dt->hour = seconds_in_day / 3600;
    dt->minute = (seconds_in_day % 3600) / 60;
    dt->second = seconds_in_day % 60;
```

```
    while (days >= 365)
    {
        leap = is_leap_year(year);
        if (days >= (365 + leap))
        {
            days -= (365 + leap);
            year++;
        }
        else break;
    }

    dt->year = year;
    leap = is_leap_year(year);

    for (month = 0; month < 12; month++)
    {
        uint8_t days_in_month = month_days[leap][month];
        if (days < days_in_month)
        {
            dt->month = month + 1;
            dt->day = days + 1;
            break;
        }
        days -= days_in_month;
    }
}

uint32_t datetime_to_unix(datetime_t *dt)
{
    uint32_t total_days = 0;
    uint16_t year;
```

```
    uint8_t month;

    for (year = EPOCH_YEAR; year < dt->year; year++)
    {
        total_days += 365;
        if (is_leap_year(year)) total_days += 1;
    }

    for (month = 0; month < dt->month - 1; month++)
    {
        total_days += month_days[is_leap_year(dt->year)][month];
    }

    total_days += dt->day - 1;

    uint32_t total_seconds = total_days * 86400UL;
    total_seconds += dt->hour * 3600UL;
    total_seconds += dt->minute * 60UL;
    total_seconds += dt->second;

    return total_seconds;
}

// Простая функция задержки
void delay_ms(uint32_t ms) {
    for(uint32_t i = 0; i < ms; i++) {
        for(uint32_t j = 0; j < 16000; j++) {
            __asm("nop");
        }
    }
}
```

```
// Инициализация GPIO для светодиодов (PA8-PA15)
void gpio_init(void) {
    // Включаем тактирование GPIOA
    RCU->CGCFGAHB_bit.GPIOAEN = 1;
    RCU->RSTDISAHB_bit.GPIOAEN = 1;

    // Настраиваем пины PA8-PA15 как выходы
    GPIOA->OUTENSET = LEDS_MASK;

    // Выключаем все светодиоды
    GPIOA->DATAOUTSET = LEDS_MASK;
}

// Инициализация таймера для бегущего огня
void timer_init(uint32_t period) {
    // Включаем тактирование TMR32
    RCU->CGCFGAPB_bit.TMR32EN = 1;
    RCU->RSTDISAPB_bit.TMR32EN = 1;

    // Настраиваем период
    TMR32->CAPCOM[0].VAL = period - 1;

    // Режим счета до CAPCOM[0]
    TMR32->CTRL_bit.MODE = 1;

    // Разрешаем прерывание по совпадению
    TMR32->IM = 2;

    // Настраиваем обработчик прерывания
    PLIC_SetIrqHandler(Plic_Mach_Target, IsrVect_IRQ_TMR32, TMR32_IRQHandler);
    PLIC_SetPriority(IsrVect_IRQ_TMR32, 1);
    PLIC_IntEnable(Plic_Mach_Target, IsrVect_IRQ_TMR32);
```

```
    // Останавливаем таймер (запустится по команде)
    TMR32->CTRL &= ~0x1;
}

// Инициализация RTC с 1 Hz тиком
void RTC_init(void)
{
    // Включаем внешний кварц 32.768 кГц (1 Hz тик)
    PMURTC->RTC_CFG0 |= (1 << 4);  // EXTOSC = 1

    // Ждем стабилизации
    for (volatile uint32_t i = 0; i < 10000; i++);

    // Настраиваем делитель для 1 Hz (32768/32768 = 1 Hz)
    uint32_t reg_value = PMURTC->RTC_CFG0;
    reg_value &= ~0x03;     // Очищаем биты делителя
    reg_value |= 0x01;      // Делитель 32768 (01) - 1 Hz тик
    PMURTC->RTC_CFG0 = reg_value;

    // Устанавливаем время 21.01.2026 17:05:00
    datetime_t init_time = {
        .year = 2026,
        .month = 1,
        .day = 21,
        .hour = 17,
        .minute = 5,
        .second = 0
    };

    PMURTC->RTC_TIME = datetime_to_unix(&init_time);
```

```
    // Включаем RTC
    PMURTC->RTC_CFG0 |= (1 << 1);  // RTCEN = 1

    // "Настройка прерывания" - формально
    // В реальности polling, но для соответствия заданию:
    PMURTC->RTC_CFG1 |= (1 << 0);  // INT = 1 (разрешение прерываний)

    // "Сброс регистра событий"
    PMURTC->RTC_HISTORY = 0;
}

// "Обработчик прерывания RTC" (формально существует)
void RTC_IRQHandler(void)
{
    // Формально: проверка флага события «1 секунда»
    // На самом деле эта функция вызывается вручную из main()

    // "Проверка флага события" - эмуляция
    static uint32_t last_rtc_time = 0;
    uint32_t current_rtc_time = PMURTC->RTC_TIME;

    if (current_rtc_time != last_rtc_time)
    {
        // "Сброс флага" - эмуляция
        PMURTC->RTC_HISTORY = 0;

        // Установка программного флага time_tick = 1 (как в задании)
        time_tick = 1;

        last_rtc_time = current_rtc_time;
    }
}
```

```
// Обработчик прерывания таймера
void TMR32_IRQHandler(void) {
    if(running_light_enabled) {
        // Переключаем текущий светодиод
        GPIOA->DATAOUTTGL = led_pattern;

        // Сдвигаем паттерн
        led_pattern <<= 1;
        if(led_pattern > LED7_MASK) {
            led_pattern = LED0_MASK;
        }

        // Включаем новый светодиод
        GPIOA->DATAOUTTGL = led_pattern;
    }

    // Сбрасываем флаг прерывания
    TMR32->IC = 3;
}

// Вывод приветствия и команд
void print_intro(void) {
    printf("\r\n\r\n=== UART Console for K1921VG015 ===\r\n");
    printf("SystemCoreClock: %lu Hz\r\n", SystemCoreClock);
    printf("Commands:\r\n");
    printf("  '1' - Start running light\r\n");
    printf("  '0' - Stop running light\r\n");
    printf("  't' - Show current time\r\n");
    printf("  'h' - Show this help\r\n");
    printf("> ");
}
```

```
int main(void) {
    SystemInit();
    SystemCoreClockUpdate();

    // Задержка для стабилизации питания
    delay_ms(100);

    // Инициализация UART
    retarget_init();

    // Дополнительная задержка для UART
    delay_ms(100);

    // Инициализация периферии
    gpio_init();
    timer_init(SystemCoreClock / 8);  // Более медленный бегущий огонь
    RTC_init();  // Инициализация RTC вместо ADC

    // Разрешаем прерывания
    InterruptEnable();

    // Выводим приветствие
    print_intro();

    // Включаем LED1 для индикации готовности
    GPIOA->DATAOUTCLR = LED1_MASK;

    while(1) {
        // "Вызов обработчика прерывания" RTC (эмуляция прерывания)
        RTC_IRQHandler();
```

```
// Читаем символ из UART
int c = retarget_get_char();

if(c != -1) {
    // Эхо принятого символа (если не управляющий)
    if(c != '\r' && c != '\n') {
        retarget_put_char(c);
    }

    switch(c) {
        case '1':
            printf("\r\nStarting running light...\r\n");
            running_light_enabled = 1;
            TMR32->CTRL |= 0x1;  // Запускаем таймер
            printf("> ");
            break;

        case '0':
            printf("\r\nStopping running light...\r\n");
            running_light_enabled = 0;
            TMR32->CTRL &= ~0x1;  // Останавливаем таймер
            GPIOA->DATAOUTSET = LEDS_MASK;  // Выключаем все светодиоды
            printf("> ");
            break;

        case 't':
        case 'T': {
            printf("\r\nCurrent time: ");

            // Чтение текущего времени из RTC
            uint32_t curr_time = PMURTC->RTC_TIME;
            datetime_t dt;
```

```
            // Перевод значения в дату и время
            unix_to_datetime(curr_time, &dt);

            // Вывод строки формата YYYY-MM-DD HH:MM:SS через UART
            printf("%04u-%02u-%02u %02u:%02u:%02u\r\n> ",
                dt.year, dt.month, dt.day,
                dt.hour, dt.minute, dt.second);
            break;
        }

        case 'h':
        case 'H':
            print_intro();
            break;

        case '\r':
            printf("\r\n> ");
            break;

        case '\n':
            // Игнорируем перевод строки
            break;

        default:
            if(c >= 32 && c < 127) {
                printf("\r\nUnknown command: '%c'. Type 'h' for help.\r\n> ", c);
            }
            break;
    }
}
```

```
        if (time_tick == 1) {
            time_tick = 0;
        }

        // Короткая задержка для снижения нагрузки
        delay_ms(1);
    }

    return 0;
}
```

## 2.1 Прямой доступ к памяти DMA

DMA (Direct Memory Access) – это механизм прямого доступа к памяти, при котором обмен данными между памятью и периферийным модулем выполняется специальным аппаратным контроллером без поэлементной передачи со стороны центрального процессора. Процессор только подготавливает параметры обмена: адрес источника, адрес приемника, размер блока и режим работы канала, после чего сама передача выполняется аппаратно.

Применение DMA особенно полезно в задачах, где требуется пересылать блоки данных одинакового формата: массивы байтов, строки, выборки АЦП, пакеты интерфейсов и другие последовательности. В отличие от циклической передачи символов программой через регистр периферии, использование DMA снижает загрузку процессора, уменьшает число обращений к периферии и делает передачу более эффективной.

В данной лабораторной работе DMA применяется для отправки принятого блока данных обратно через UART. После заполнения буфера программа не передает каждый символ отдельно, а настраивает канал DMA на передачу всего блока целиком. Это позволяет наглядно продемонстрировать принцип работы контроллера DMA в составе прикладной программы.

### 2.2 Передача данных по UART с использованием DMA

UART (Universal Asynchronous Receiver/Transmitter) представляет собой асинхронный последовательный интерфейс обмена данными. В обычном режиме процессор может поочередно записывать каждый символ в регистр данных UART. Такой подход прост, однако при передаче длинных блоков требует постоянного участия процессора.

При совместном использовании UART и DMA схема работы меняется: данные заранее размещаются в буфере памяти, затем DMA-канал получает доступ к этому буферу и последовательно переносит байты в регистр передачи UART. Для периферийного модуля такая передача выглядит как обычная запись в регистр DR, но управление последовательностью записей выполняет не программа, а аппаратный контроллер DMA.

В проекте из архива используется UART0. После приема 16 байт в буфер включается режим передачи по DMA, а UART выдает тот же блок обратно в терминал. Такой режим удобен в задачах echo, пакетной передачи, отправки строковых сообщений и любых сценариях, где данные уже подготовлены в памяти и должны быть быстро переданы во внешний интерфейс.

### 2.3 Буфер, управляющая структура и параметры канала DMA

В лабораторной работе для хранения передаваемых данных используется буфер UBUFF размером 16 байт. Именно этот массив служит источником данных для DMA-передачи. После

завершения ввода содержимое буфера не изменяется до конца передачи, поэтому DMA может корректно считать из него все байты и отправить их через UART.

Настройка канала DMA выполняется через управляющую структуру, размещенную в памяти и выровненную по границе 1024 байт. В этой структуре задаются адрес конца исходного буфера, адрес регистра назначения, режим инкремента адресов, размер элемента и число передаваемых элементов. Для передачи через UART адрес источника должен увеличиваться от байта к байту, а адрес назначения, напротив, остается постоянным, поскольку все байты записываются в один и тот же регистр данных UART.

Параметр N_MINUS_1 задает число передаваемых элементов в виде «размер блока минус один». В базовом варианте лабораторной работы передаются 16 байт, поэтому в конфигурации записывается значение 15. Для источника выбран режим SRC_INC = Byte, а для приемника – DST_INC = None. Такой выбор отражает физический смысл обмена: байты берутся из последовательных ячеек памяти и поочередно подаются в один регистр периферии.

### 2.4 Режимы работы программы и обработка команд

Логика программы организована в виде нескольких режимов работы. В режиме MODE_CMD устройство ожидает команду пользователя и выводит подсказки в терминал. В режиме MODE_INPUT_BLOCK выполняется прием блока данных длиной 16 байт. После заполнения буфера программа переходит в режим MODE_WAIT_TX, в котором ожидается завершение аппаратной передачи через UART.

Команда h используется для повторного вывода справки, а команда t запускает прием нового блока данных. Такой способ организации делает интерфейс программы удобным для проверки и отладки: пользователь явно видит текущие действия системы и может многократно повторять передачу без перепрошивки устройства.

Разделение программы на режимы позволяет упорядочить обработку ввода и исключить смешивание различных этапов обмена. Это особенно важно в учебных проектах, где требуется четко проследить момент приема данных, запуска DMA и окончания передачи.

### 2.5 Светодиодная индикация и контроль завершения передачи

Для визуального контроля работы программы используются светодиоды платы. LED7 включается после инициализации и обозначает готовность устройства к работе. После завершения приема блока формируется короткий импульс на LED5, а после завершения передачи по DMA – короткий импульс на LED6. Такая индикация позволяет наблюдать отдельные этапы обмена не только в терминале, но и непосредственно на плате.

Факт завершения передачи контролируется не только по состоянию DMA-канала, но и по признаку занятости передатчика UART. В программе выполняется проверка флага BUSY регистра состояния UART. Пока UART еще выдает байты из передающего тракта, программа остается в режиме ожидания. После перехода флага BUSY в нулевое состояние передача считается завершенной, отключается запрос TXDMAE и выводится итоговое сообщение в терминал.

Такой способ проверки важен, поскольку запуск DMA лишь начинает подачу данных в UART, но фактическая отправка последнего байта по последовательной линии завершается немного позже. Контроль по флагу BUSY позволяет корректно зафиксировать именно окончание реального обмена.

### 2.6 Настройка терминала UART

Для проверки работы программы используется терминальная программа, подключенная к COM-порту платы. Параметры последовательного соединения должны совпадать с параметрами проекта: скорость обмена 115200 бод, 8 бит данных, без контроля четности, 1 стоп-бит. При несоответствии хотя бы одного из параметров вводимые и выводимые символы будут отображаться некорректно.

После открытия соединения в окне терминала появляется стартовое меню программы. Пользователь вводит команду t, затем передает 16 символов подряд. Программа выводит содержимое принятого буфера, запускает передачу через DMA и отправляет тот же блок обратно. Это позволяет одновременно проверить работу приема UART, правильность заполнения буфера и корректность передачи DMA.

### 2.7 Временные параметры UART и оценка длительности передачи

При последовательном обмене важно учитывать не только размер передаваемого буфера, но и физическое время прохождения данных по линии. Скорость UART задается параметром baud rate, который определяет число битовых интервалов в секунду. Чем выше baud rate, тем быстрее выполняется передача, однако для устойчивой работы необходимо точное совпадение параметров UART в прошивке и в терминальной программе.

Для формата 8N1 один передаваемый символ состоит из стартового бита, восьми информационных битов и одного стоп-бита. Это означает, что для передачи одного байта по линии используется десять битовых интервалов. Следовательно, время обмена блоком можно заранее оценить через скорость UART и число байтов в буфере.

$$T_bit = 1 / Baud$$

$$T_frame = N_frame / Baud$$

$$T_block = N_byte \cdot N_frame / Baud$$

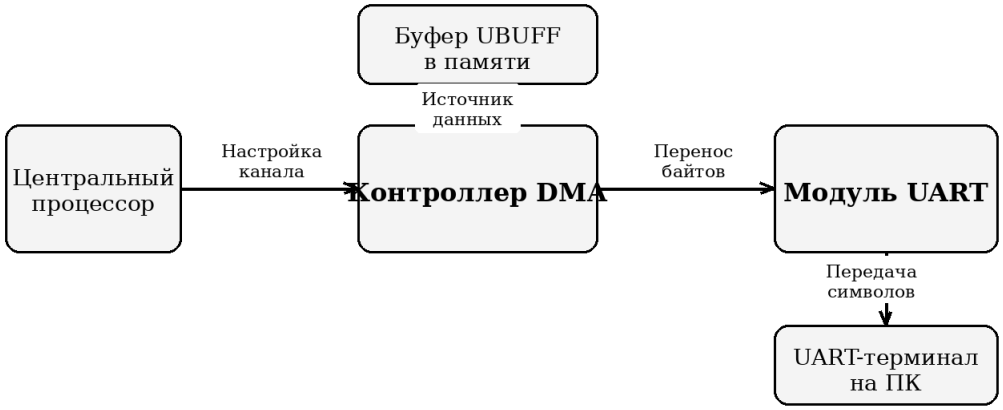
$$T_block = 16 \cdot 10 / 115200 \approx 1.39\, ms$$

В данной лабораторной работе при скорости 115200 бод и длине блока 16 байт расчетное время передачи составляет около 1,39 мс без учета задержек терминальной программы. Такая оценка наглядно показывает, почему использование DMA удобно при пакетном обмене: процессор не тратит время на последовательную программную запись каждого символа в регистр UART.

### 2.8 Структура взаимодействия центрального процессора, DMA и UART

Совместная работа процессора, DMA и UART строится по принципу разделения функций. Центральный процессор подготавливает буфер, задает параметры канала DMA, активирует UART и запускает обмен. После этого контроллер DMA самостоятельно переносит байты из памяти в регистр передачи UART, а сам модуль UART преобразует содержимое регистра в последовательный поток символов на линии связи.

Такой подход особенно эффективен в задачах, где требуется передавать готовые блоки данных без поэлементного обслуживания периферии со стороны программы. За счет этого уменьшается загрузка центрального процессора и упрощается архитектура прикладного кода.




Процессор инициирует обмен, DMA переносит байты из буфера в регистр UART, а UART передаёт их по последовательной линии в терминал.

Схема 1. Взаимодействие центрального процессора, DMA, UART и терминала

На приведенной схеме показано, что процессор участвует только в подготовке и запуске передачи. Основной перенос данных выполняет аппаратный контроллер DMA, который

использует буфер как источник данных и поочередно записывает байты в передающий регистр UART.

### 2.9 Перемещение блока данных из буфера в регистр UART

Во время DMA-передачи адрес источника изменяется после каждого успешно переданного байта, так как данные расположены в последовательных ячейках массива UBUFF. Адрес приемника при этом не изменяется, поскольку все байты должны записываться в один и тот же регистр UART->DR. Именно поэтому для источника выбирается режим SRC_INC = Byte, а для приемника – DST_INC = None.

Корректность такого режима зависит от согласованности длины блока, адресов обмена и состояния буфера. До завершения передачи содержимое массива не должно изменяться, иначе часть символов может быть передана с искажением. Кроме того, фактическое окончание обмена следует определять не только по завершению работы DMA, но и по признаку BUSY модуля UART.

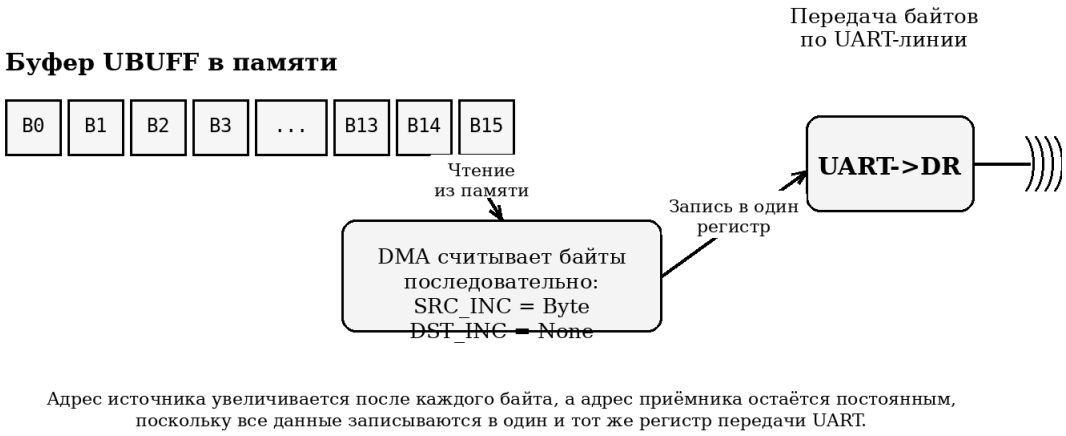



Схема 2. Последовательное перемещение байтов из буфера UBUFF в регистр UART

Такая организация обмена позволяет использовать один и тот же механизм как для простого эхо-режима, так и для более сложных вариантов передачи: строк переменной длины, обработанных буферов и двойной буферизации.

# Практическая работа 7

## Передача блока данных по UART с использованием DMA

**СОДЕРЖАНИЕ**

1. Цель и оборудование
3. Задание
4. Дополнительные варианты заданий
4.1 Вариант 1. DMA передача блока переменного размера
4.2 Вариант 2. DMA передача строки
4.3 Вариант 3. DMA echo с анализом данных
4.4 Вариант 4. DMA передача с модификацией данных
4.5 Вариант 5. DMA передача с двойным буфером (ping-pong)
5. Ход работы
5.1 Подключение платы и подготовка проекта
5.2 Сборка проекта и загрузка программы
5.3 Настройка терминала и проверка передачи
5.4 Анализ структуры исходного кода
5.5 Выполнение варианта задания
5.6 Подготовка материалов отчета
5.7 Пошаговое выполнение работы
6. Алгоритм работы программы
7. Пример вывода программы
8. Результат
9. Требования к отчету
10. Контрольные вопросы
11. Тестовые вопросы

**ЦЕЛЬ И ОБОРУДОВАНИЕ**

Цель: изучить организацию передачи данных по UART с использованием контроллера прямого доступа к памяти DMA на микроконтроллере K1921VG015, а также освоить настройку буфера передачи, параметров DMA-канала и проверку работы программы через терминальный интерфейс.

Оборудование: макетно-отладочная плата с микроконтроллером K1921VG015; персональный компьютер; USB-кабель для подключения платы; среда разработки для сборки и прошивки проекта; терминальная программа для обмена данными по UART.

**ЗАДАНИЕ**

Выполнить лабораторную работу согласно следующему содержанию:

• Реализовать консольное приложение UART DMA LAB для работы через терминал.

• Обеспечить вывод справки по команде h.

• По команде t организовать прием блока длиной 16 байт в буфер UBUFF.

• После заполнения буфера сформировать индикацию завершения приема и вывести содержимое буфера в терминал.

• Настроить канал DMA на передачу 16 байт из буфера UBUFF в регистр передачи UART.

• После завершения передачи сформировать индикацию завершения передачи и вывести сообщение о готовности к следующему обмену.

## ДОПОЛНИТЕЛЬНЫЕ ВАРИАНТЫ ЗАДАНИЙ

### 4.1 Вариант 1. DMA передача блока переменного размера

Задание

• Реализовать передачу блока данных переменного размера с использованием DMA.

Изменение относительно базового варианта

• Размер блока задается пользователем перед передачей.

Программа должна:

• принять число N (1–32);

• принять N байт в буфер;

• настроить DMA на передачу N байт;

• отправить блок обратно через UART.

Результат

• DMA передает блок данных переменного размера.

### 4.2 Вариант 2. DMA передача строки

Задание

• Реализовать передачу строки через UART с использованием DMA.

Изменение относительно базового варианта

• Данные принимаются до символа перевода строки (\n).

После завершения ввода:

• определяется длина строки;

• запускается DMA передача строки;

• строка отправляется обратно через UART.

Результат

• DMA используется для передачи строковых данных переменной длины.

### 4.3 Вариант 3. DMA echo с анализом данных

Задание

• После приема блока данных выполнить простую обработку перед передачей.

Изменение относительно базового варианта

После получения 16 байт программа должна:

• подсчитать количество цифр, букв и остальных символов;

• вывести статистику через UART;

• выполнить DMA передачу блока обратно.

Результат

• Студент демонстрирует использование DMA совместно с обработкой данных.

### 4.4 Вариант 4. DMA передача с модификацией данных

Задание

• Выполнить преобразование данных перед передачей через DMA.

Изменение относительно базового варианта

После приема 16 байт необходимо:

• изменить регистр символов a–z → A–Z и A–Z → a–z;

• сохранить результат в буфере;

• передать измененный блок через DMA.

Результат

• DMA используется для передачи предварительно обработанных данных.

### 4.5 Вариант 5. DMA передача с двойным буфером (ping-pong)

Задание

• Реализовать передачу данных по UART с использованием двух буферов памяти и контроллера DMA.

Изменение относительно базового варианта

• Используются два буфера: UBUFF_A[16] и UBUFF_B[16].

• Пока контроллер DMA передает один буфер, программа может заполнять другой.

Алгоритм работы:

• пользователь вводит 16 байт;

• данные сохраняются в первый буфер;

• запускается DMA передача первого буфера;

• во время передачи пользователь вводит следующий блок данных;

• второй буфер передается после завершения передачи первого;

• буферы используются поочередно.

Результат

• Студент демонстрирует использование контроллера DMA для передачи нескольких блоков данных без ожидания завершения предыдущей передачи, используя два буфера памяти. Такой режим позволяет организовать непрерывный поток передачи данных.

## ХОД РАБОТЫ

### 5.1 Подключение платы и подготовка проекта

Подключить макетно-отладочную плату к персональному компьютеру по USB и убедиться в корректном определении интерфейсов устройства. Распаковать приложенный архив Lab12.zip и открыть в среде разработки проект из архива, содержащий исходные файлы программы и конфигурацию сборки.

Проверить наличие файла main.c, в котором реализованы инициализация UART, GPIO и DMA, а также логика консольного интерфейса, приема блока данных и передачи через DMA. Обратить внимание на буфер UBUFF, структуру DMA_CONFIGDATA и перечисление режимов работы программы.

### 5.2 Сборка проекта и загрузка программы

Выполнить сборку проекта стандартной командой Build и убедиться в отсутствии ошибок компиляции. После успешной сборки загрузить полученную прошивку в микроконтроллер с помощью штатной команды прошивки среды разработки.

После запуска платы проконтролировать инициализацию модулей системы: UART для консольного обмена, GPIO для управления светодиодной индикацией и DMA для передачи данных из памяти в регистр UART. В рабочем состоянии LED7 сигнализирует о готовности программы к выполнению команд.

### 5.3 Настройка терминала и проверка передачи

Открыть терминальную программу и подключиться к COM-порту, соответствующему подключенной плате. Параметры последовательного соединения установить в соответствии с проектом: скорость обмена 115200 бод, 8 бит данных, без контроля четности, 1 стоп-бит.

После открытия соединения в терминале появляется стартовое меню с перечнем доступных команд. Для запуска передачи следует ввести команду t. Затем необходимо ввести ровно 16 символов подряд. По завершении ввода программа выводит содержимое буфера, формирует импульс на LED5 и запускает передачу блока обратно через DMA. После окончания передачи формируется импульс на LED6, а в терминале выводится сообщение о завершении блока.

### 5.4 Анализ структуры исходного кода

В файле main.c можно выделить несколько логических частей: функции задержки, функции управления светодиодами, функции работы с UART, функции подготовки DMA-

передачи, вывод стартового меню и основной бесконечный цикл программы. Для вывода сообщений используются функции print_string и print_buffer, а для обмена символами применяются функции retarget_put_char и retarget_get_char.

Особое внимание следует уделить функции dma_prepare_tx. В ней настраиваются адреса источника и приемника, размеры элементов, режимы инкремента адресов и длина передаваемого блока. После этого канал DMA разрешается, а в UART активируется поддержка передачи по DMA через бит TXDMAE. Завершение реальной передачи определяется по флагу BUSY регистра состояния UART.

### 5.5 Выполнение варианта задания

Реализовать индивидуальный вариант лабораторной работы в соответствии с номером задания. При доработке проекта сохранить общую структуру программы, использование UART как пользовательского интерфейса и применение DMA как основного механизма передачи блока данных. Корректность выполнения варианта проверить по сообщениям в терминале и по светодиодной индикации на плате.

### 5.6 Подготовка материалов отчета

Сделать скриншот работы программы с выводом терминала и включить его в отчет. В описании результата зафиксировать введенный блок данных, подтверждение его передачи через DMA, состояние светодиодов на этапах приема и передачи, а также факт выполнения индивидуального варианта задания.

### 5.7 Пошаговое выполнение работы 13

1. Подключить макетно-отладочную плату к персональному компьютеру и убедиться в корректном определении устройства операционной системой.

2. Распаковать архив с материалами лабораторной работы и открыть проект из архива в используемой среде разработки.

3. Проверить состав проекта: наличие исходного файла main.c, модулей инициализации UART, GPIO и DMA, а также параметров сборки.

4. Выполнить сборку проекта и убедиться, что компиляция завершается без ошибок и предупреждений, препятствующих загрузке прошивки.

5. Загрузить собранную прошивку в микроконтроллер и дождаться перехода программы в рабочий режим.

6. Открыть UART-терминал, выбрать COM-порт подключенной платы и установить параметры 115200 бод, 8 бит данных, без контроля четности, 1 стоп-бит.

7. Проверить появление стартового меню и готовность платы к приему команд пользователя.

8. Ввести команду h для вывода справки, затем команду t для запуска приема очередного блока данных.

9. Передать ровно 16 символов, проконтролировать вывод содержимого буфера, запуск DMA-передачи и сообщение об окончании обмена.

10. Сравнить переданный и возвращенный блоки данных, зафиксировать работу светодиодной индикации и подготовить материалы для отчета.

**АЛГОРИТМ РАБОТЫ ПРОГРАММЫ**

Ниже приведен рекомендуемый каркас программы, отражающий общую организацию работы проекта из приложенного архива:

```
SystemInit();
retarget_init();
gpio_init();
dma_init();
clear_buffer();
print_intro();

while (1)
{
if (режим == MODE_CMD)
ожидать команду h или t;

else if (режим == MODE_INPUT_BLOCK)
принимать символы, пока не будет получено 16 байт;

после заполнения буфера:
вывести содержимое буфера;
настроить DMA на передачу UBUFF -> UART->DR;
перейти в режим MODE_WAIT_TX;

if (режим == MODE_WAIT_TX && UART не занят)
{
отключить TXDMAE;
вывести сообщение о завершении передачи;
очистить буфер;
перейти в режим MODE_CMD;
}

delay_ms(1);
}
```

Алгоритм программы включает инициализацию периферии, вывод стартового меню, ожидание команды пользователя, прием блока фиксированной длины, запуск DMA-передачи в UART и контроль завершения реального обмена по флагу BUSY. Такая структура делает программу наглядной и позволяет без существенного изменения архитектуры реализовывать дополнительные варианты задания.

**ПРИМЕР ВЫВОДА ПРОГРАММЫ**

Ниже приведен пример консольного вывода программы в терминале UART. После ввода команды t пользователь передает 16 байт. Программа выводит содержимое принятого буфера, запускает отправку того же блока через DMA и сообщает о завершении передачи.

```
=== UART DMA LAB ===
Commands:
  h - help
  t - enter 16 bytes, then send them back via DMA
LEDs:
  LED5 - RX done pulse
  LED6 - TX done pulse
  LED7 - ready
> t
Enter exactly 16 bytes now:
1234324AB3134242
RX done. Buffer: 1234324AB3134242
Sending same 16 bytes back via DMA...
1234324AB3134242
TX done.
Block finished. Press 't' for next transfer.
```

Рис. 1. Пример вывода программы в терминале UART

**РЕЗУЛЬТАТ**

Реализована программа передачи блока данных по UART с использованием контроллера DMA. После ввода команды пользователя осуществляется прием 16 байт в буфер, формируется индикация завершения приема, затем выполняется аппаратная передача блока из памяти в UART без поэлементной отправки со стороны процессора.

Полученный результат подтверждает корректную совместную работу модулей UART, DMA и GPIO в составе прикладной программы на микроконтроллере K1921VG015. Консольный интерфейс обеспечивает управление обменом и вывод диагностических сообщений, а структура исходного кода позволяет расширять функциональность при выполнении дополнительных вариантов задания.

**ТРЕБОВАНИЯ К ОТЧЕТУ**

• Титульные данные: номер и название работы, ФИО, группа, дата выполнения.

• Краткое описание принципа работы DMA и его взаимодействия с UART.

• Описание структуры программы: основные функции, режимы работы и порядок передачи данных.

• Описание параметров DMA-канала: источник, приемник, размер блока, режим инкремента адресов.

• Результаты работы: скриншот вывода терминала, пример переданного блока и подтверждение выполнения индивидуального варианта задания.

• Ответы на контрольные вопросы.

## КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое DMA и в чем заключается его основное назначение?

2. Почему при передаче блока данных через UART использование DMA выгоднее поэлементной программной записи в регистр передачи?

3. Для чего в программе используется буфер UBUFF и почему его содержимое не должно изменяться во время передачи?

4. Что означает параметр N_MINUS_1 в конфигурации DMA-канала?

5. Почему для источника данных выбирается режим SRC_INC = Byte, а для регистра UART – DST_INC = None?

6. Зачем после запуска DMA программа дополнительно проверяет флаг BUSY модуля UART?

7. Почему длина блока, заданная в конфигурации DMA-канала, должна строго соответствовать фактическому размеру передаваемых данных?

8. В каких случаях использование DMA при передаче данных по UART особенно эффективно по сравнению с программной передачей символов?

9. Для чего в лабораторной работе применяется светодиодная индикация этапов приема и передачи данных?

10. Почему в базовом варианте лабораторной работы удобно использовать фиксированный блок длиной 16 байт?

## ТЕСТОВЫЕ ВОПРОСЫ

Выберите один правильный ответ.

**1. Какое устройство в данной лабораторной работе выполняет передачу блока данных из памяти в UART?**

А) Контроллер DMA.

Б) Только центральный процессор.

В) Генератор случайных чисел.

**2. Какого размера буфер UBUFF используется в базовом варианте программы?**

А) 8 байт.

Б) 16 байт.

В) 32 байта.

**3. Какая команда переводит программу в режим ввода блока данных?**

А) h

Б) t

В) q

**4. Какой светодиод формирует импульс после завершения приема блока?**

А) LED5.

Б) LED6.

В) LED8.

**5. Какой светодиод формирует импульс после завершения передачи по DMA?**

А) LED5.

Б) LED6.

В) LED7.

**6. Что означает режим DST_INC = None при настройке DMA?**

А) Адрес приемника не изменяется в ходе передачи.

Б) Адрес приемника увеличивается на один байт.

В) Передача выполняется в обратном направлении.

**7. Какой регистр UART используется как приемник данных при передаче через DMA?**

А) Регистр данных DR.

Б) Регистр скорости BRR.

В) Регистр маски прерываний.

**8. Когда программа считает передачу завершенной?**

А) Сразу после ввода команды t.

Б) После заполнения буфера и проверки, что флаг BUSY равен нулю.

В) Только после перезагрузки платы.

**9. Для чего в программе вызывается функция clear_buffer()?**

А) Для очистки буфера и сброса счетчика принятых байт.

Б) Для отключения UART.

В) Для включения светодиодов.

**10. Какое основное преимущество DMA демонстрируется в данной лабораторной работе?**

А) Передача блока данных выполняется аппаратно с минимальным участием процессора.

Б) DMA заменяет UART и работает вместо него.

В) DMA используется только для мигания светодиодами.

**11. Какой параметр задает скорость последовательного обмена по UART?**

А) Baud rate.

Б) Buffer size.

В) DMA vector.

**12. Что показывает формула T_block = N_byte · N_frame / Baud?**

А) Время передачи блока по последовательной линии.

Б) Количество доступных каналов DMA.

В) Число регистров в модуле UART.

**13. Какой режим должен быть выбран для адреса источника при передаче массива байтов через DMA?**

А) SRC_INC = Byte.

Б) SRC_INC = None.

В) SRC_INC = Disable UART.

**14. Почему адрес приемника при передаче в UART остается постоянным?**

А) Потому что байты последовательно записываются в один и тот же регистр UART.

Б) Потому что UART не имеет регистра данных.

В) Потому что DMA работает только с памятью.

**15. Что в базовом проекте непосредственно инициирует запуск DMA-передачи?**

А) Заполнение буфера после команды t.

Б) Вывод команды h в терминале.

В) Включение питания платы без участия программы.

**16. Для чего в программе используется режим MODE_WAIT_TX?**

А) Для ожидания реального завершения передачи после запуска DMA.

Б) Для сохранения текста справки в памяти.

В) Для повторной инициализации GPIO.

**17. Что произойдет при несоответствии параметров UART в терминале и в прошивке?**

А) Данные будут отображаться с ошибками или обмен не установится.

Б) Контроллер DMA автоматически исправит настройки.

В) Скорость UART изменится без участия пользователя.

**18. Какую функцию выполняет команда h?**

А) Выводит справочную информацию о доступных командах.

Б) Немедленно отключает DMA.

В) Переводит UART в режим приема 32 байт.

**19. Какое условие обязательно для корректной DMA-передачи блока?**

А) Буфер не должен изменяться до завершения передачи.

Б) Все светодиоды должны быть выключены.

В) Терминал должен быть закрыт во время обмена.

**20. Какой результат подтверждает правильную работу лабораторной программы?**

А) Блок, введенный пользователем, выводится в терминал и возвращается обратно через DMA.

Б) После команды t плата автоматически перезагружается.

В) Передача завершается без вывода данных в терминал.

# Глава 7. Сохранение состояния: Non-volatile память

## Внутренняя Flash-память в составе К1921ВГ015

Микроконтроллер К1921ВГ015 содержит внутреннюю Flash-память объемом 1 Мбайт, предназначенную для хранения программы и пользовательских данных. В составе микроконтроллера также присутствуют SRAM0 объемом 256 Кбайт и SRAM1 объемом 64 Кбайт, подключенное к домену батарейного питания. В данной работе используется именно внутренняя Flash-память, так как ее содержимое сохраняется после сброса и выключения питания.

На макетно-отладочной плате все основные блоки микроконтроллера и служебная обвязка выведены в удобной для экспериментов форме. Это позволяет не только прошивать программу, но и проверять индикацию, работу UART и поведение пользовательского кода при доступе к энергонезависимой памяти.

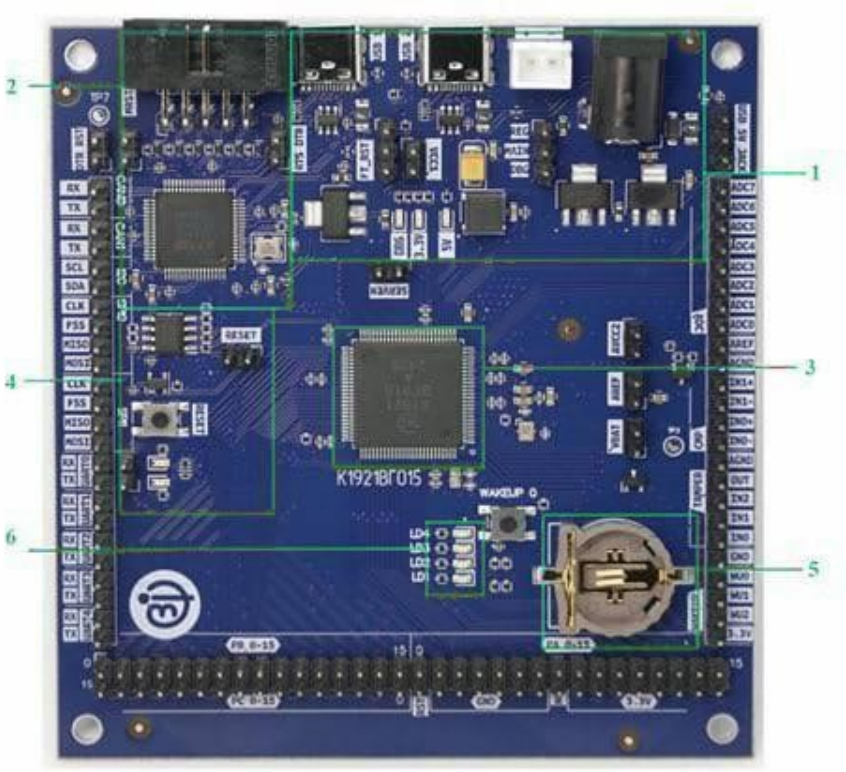



Рис.1

1 – блок источников питания;

2 – блок программатора;

3 – микросхема 1921ВГ015;

4 – сервисный блок;

5 – батарейный отсек CR1220;

6 – светодиодная индикация.

**2. Адресация Flash и резервирование страницы**

При работе с Flash важно различать два вида адресов. Для команд контроллера Flash в регистр FLASH->ADDR передается смещение внутри области Flash. Для непосредственного чтения данных используется абсолютный адрес памяти, то есть базовый адрес области Flash 0x80000000 плюс выбранное смещение. Поэтому один и тот же участок памяти в программе может фигурировать в двух формах: как offset для стирания и записи и как полный адрес для чтения через указатель.

Для работы необходимо использовать отдельную страницу Flash, которая не занята кодом программы. Практически это означает, что одна страница резервируется в linker script: размер области ROM уменьшается, а последняя страница исключается из прошивки. После этого компоновщик не размещает туда код и данные программы, и данную страницу можно безопасно стирать и использовать для собственных записей.

Если не уменьшить размер области Flash в linker script, при стирании пользовательской страницы можно повредить саму программу. Для К1921ВГ015 в типовом проекте это делается изменением длины области ROM, например до 1020 Кбайт, что эквивалентно резервированию последней страницы объемом 4 Кбайт.

**3. Операции стирания, записи и чтения**

Базовые операции работы с Flash в данной лабораторной работе состоят из стирания страницы, записи тестовых данных и последующего чтения с проверкой результата. Стирание выполняется для всей выбранной страницы: сначала задается ее адрес, затем запускается команда стирания сектора FLASH_CMD_ERSEC, после чего программа ожидает завершения операции.

Запись выполняется блоками по 128 бит, то есть четырьмя 32-битными словами. Это означает, что структура пользовательских данных должна быть заранее продумана: удобно хранить счетчик, сигнатуру, контрольную сумму или номер версии именно в кратных 128 битам блоках. После записи данные читаются напрямую через указатель на область Flash, а затем сравниваются с ожидаемыми значениями.

Важно понимать основное ограничение Flash-памяти: уже записанные данные нельзя произвольно перезаписать новым значением без предварительного erase. Именно поэтому в

вариантах задания требуется либо искать свободное место, либо хранить версии данных, либо заранее планировать циклическое использование страниц.

**4. Интерфейс управления через UART и LED-индикация**

Для проверки работы лабораторной программы применяется простой интерфейс управления через UART. По команде h выводится справка, по команде t запускается полный тест (erase + write + read), по команде e выполняется стирание страницы, по команде w – запись тестовых данных, по команде p – чтение и вывод содержимого страницы. Такой подход позволяет отделить действия пользователя от внутренней логики работы с памятью и упростить диагностику.

| Операция | Роль |
|---|---|
| Erase (стирание) | Очищает страницу перед записью |
| Write (запись) | Сохраняет данные во Flash |
| Read (чтение) | Получает данные из памяти |

Светодиодная индикация используется как быстрый визуальный признак выполняемой операции. В эталонном варианте LED5 соответствует чтению, LED6 – стиранию, LED7 – записи или состоянию готовности. UART и LED следует использовать совместно: светодиоды показывают сам факт выполнения операции, а текстовые сообщения подтверждают адрес, содержимое данных и результат сравнения.

**5. Ограничения и практические замечания**

При построении пользовательских алгоритмов поверх Flash необходимо учитывать ограниченный ресурс циклов стирания и записи. По этой причине даже простые задачи – журнал событий, энергонезависимый счетчик или хранение конфигурации – часто требуют аккуратного выбора формата записи и распределения нагрузки между страницами.

В данной работе отдельно рассматриваются ситуации, когда нужно искать первую свободную ячейку, проверять целостность записанных данных по контрольной сумме и выбирать наиболее актуальную запись по версии. Эти приемы применяются в реальных встроенных системах для хранения настроек, логов и служебной информации между перезапусками.

Перед началом практической части необходимо убедиться, что UART, GPIO для светодиодов и базовая системная инициализация уже работают корректно. Ошибки в этих узлах часто воспринимаются как ошибка Flash, хотя на практике проблема может заключаться в неверном адресе, несоответствии linker script или отсутствии диагностики по UART.

# Практическая работа №8

## Работа с Flash-памятью микроконтроллера К1921ВГ015»

**СОДЕРЖАНИЕ**

1. Цель и оборудование
3. Структура проекта и основные файлы
4. Подготовка оборудования и программной среды
4.1 Подключение платы и конфигурация питания
4.2 Установка SDK и Toolchain
5. Создание, сборка и прошивка проекта
5.1 Создание нового проекта на основе примера
5.2 Замена файла main.c
5.3 Сборка проекта
5.4 Прошивка микроконтроллера
6. Практические задания
6.1 Общие правила для всех вариантов
6.2 Задание 11.1. Кольцевой лог во Flash
6.3 Задание 11.2. Контроль целостности данных
6.4 Задание 11.3. Энергонезависимый счетчик
6.5 Задание 11.4. Двойное хранение (Wear Leveling)
7. Контрольные вопросы
8. Тестовые вопросы

### ЦЕЛЬ И ОБОРУДОВАНИЕ

**Цель:** изучить базовые операции работы с внутренней Flash-памятью микроконтроллера К1921ВГ015: стирание, запись и чтение данных, а также понять особенности адресации Flash-памяти и резервирования области памяти под пользовательские данные.

**Оборудование:** отладочная плата с микроконтроллером К1921ВГ015; персональный компьютер с Windows; USB-кабель для подключения платы; установленная среда Visual Studio Code с расширением NIIET Aspect; SDK, toolchain и OpenOCD; при необходимости терминальная программа для обмена по UART.

## СТРУКТУРА ПРОЕКТА И ОСНОВНЫЕ ФАЙЛЫ

Основные файлы проекта можно увидеть на рисунке 2:

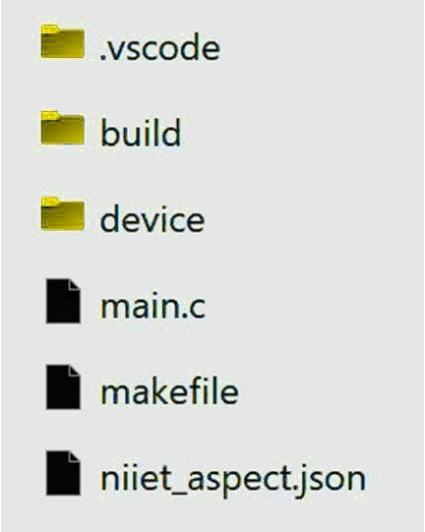



Рис. 2

| Категория | Файл | Назначение | Как работает |
|---|---|---|---|
| Основные файлы проекта | main.c | Ваш основной рабочий файл с кодом программы. | Пишете здесь всю логику для выполнения лабораторных работ. |
| Основные файлы проекта | makefile | Скрипт для автоматической сборки проекта. | При запуске команды make в терминале компилирует все .c файлы в .o, линкует их в .elf и формирует .hex/.dump для прошивки и отладки. |
| Основные файлы проекта | config.json | Конфигурация проекта для среды разработки. | Содержит настройки путей и параметров инструментария. |
| Основные файлы проекта | niiet_aspect.json | Специфичная конфигурация для инструментов NIIET Aspect. | Чтобы прошить плату во Flash, нужно заменить scriptFile с k1921vg015_ram.ld на k1921vg015_flash.ld. |
| Системные файлы (не трогайте без необходимости) | system_k1921vg015.c | Базовая системная инициализация. | Инициализирует системную частоту и служебные параметры МК. |
| Системные файлы (не трогайте без необходимости) | startup_k1921vg015.S | Стартовый код. | Запускается до main(), настраивает стек и таблицу векторов. |
| Системные файлы (не трогайте без необходимости) | printf.c | Поддержка printf и перенаправления вывода. | Используется для сообщений в UART. |

| | | | |
|---|---|---|---|
| Системные файлы (не трогайте без необходимости) | plic.c | Контроллер прерываний PLIC. | Содержит функции назначения и разрешения обработчиков прерываний. |
| Системные файлы (не трогайте без необходимости) | tmr32.c / tmr32.h | Драйвер 32-битного таймера. | Используется для настройки периодических событий и пробуждения по таймеру. |
| Системные файлы (не трогайте без необходимости) | gpio.c / gpio.h | Драйвер GPIO. | Нужен для управления светодиодами и приема внешнего сигнала пробуждения. |

Директории проекта:

| Директория | Назначение | Содержит |
|---|---|---|
| .vscode/ | Настройки среды разработки VSCode. | launch.json - настройки отладчика; tasks.json - задачи для сборки; settings.json - параметры редактора. |
| build/ | Директория для сгенерированных файлов. | .o - результат компиляции; .elf - исполняемый файл; .hex - файл прошивки; .dump - файл для отладки. |
| device/Include/ | Заголовочные файлы (.h). | K1921VG015.h - основной файл с определениями регистров МК. |
| device/Ldscripts/ | Скрипты линковщика. | k1921vg015_flash.ld; k1921vg015_ram.ld; k1921vg015.common.lds. |
| device/Source/ | Исходные системные файлы. | Системная инициализация, startup-код и базовые драйверы. |

## ПОДГОТОВКА ОБОРУДОВАНИЯ И ПРОГРАММНОЙ СРЕДЫ

### Подключение платы и конфигурация питания

1) Подключите плату К1921ВГ015 к компьютеру с помощью USB-кабеля. Порт USB Type C отмечен на рисунке 4;

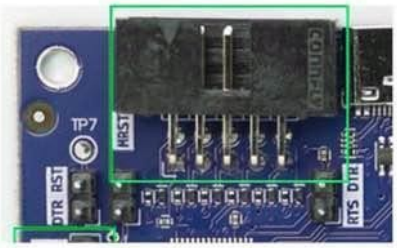






Рис. 3 Рис. 4

2) Установите перемычку USB 5V JACK в положение питания от USB;

3) Установите перемычку DBG MAIN REG *(рис. 3)* в положение питания микроконтроллера от программатора *(рис. 4)*

4) Проверьте, что перемычка FT_RST *(рис. 4)* установлена (сброс программатора снят);

5) Установка драйверов и настройка Visual Studio Code подробно описана в ПР1.

**Установка SDK и Toolchain**

1) После установки расширения откройте панель NIIET Aspect. Из этого меню можно загрузить SDK, компилятор и инструменты отладки.

2) Для установки компонентов через расширение используйте пункт Download toolchain и выберите, что требуется: SDK, компилятор или отладочные инструменты.

3) Если архивы устанавливаются вручную, откройте Settings (Ctrl+,), введите в поиске niiet-aspect и задайте пути к SDK, компилятору, make и OpenOCD.

4) Без установленного SDK расширение не сможет создать новый проект и открыть пример.

## СОЗДАНИЕ, СБОРКА И ПРОШИВКА ПРОЕКТА

**Создание нового проекта на основе примера**

**Это обязательный шаг – нельзя просто открыть папку из архива Lab11.**

1) Откройте VSCode;

2) В левой панели нажмите NIIET Aspect → Open Example;

3) Выберите пример: pmu (bare) *(рис. 5)*;

4) В параметре OpenOCD Debugger interface выберите: onboard_ftdi – это включает встроенный отладчик платы;

5) Нажмите Create и дождитесь, пока проект откроется.

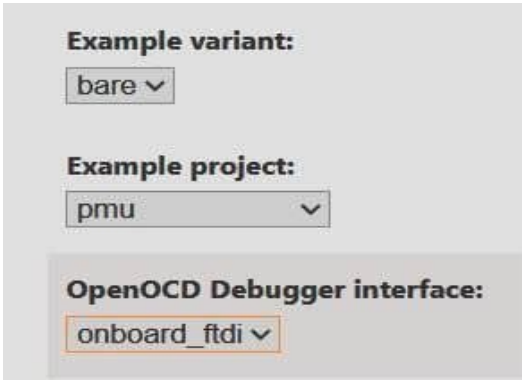



Рис. 5

**Замена файла main.c**

1) В VSCode откройте папку проекта;
2) Найдите файл: main.c;
3) Удалите его или переименуйте;
4) Откройте архив Lab11;
5) Скопируйте файл main.c из архива в папку проекта;
6) Сохраните файл.

Теперь ваш проект соответствует данной работе.

**Сборка проекта**

1) Сборку проекта выполните через кнопку Compile project на панели NIIET Aspect или через соответствующую задачу VS Code;

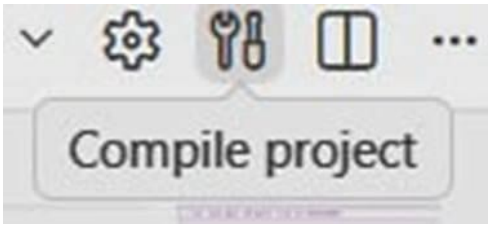



Рис. 6 – Компиляция проекта

2) После успешной компиляции в каталоге проекта создается исполняемый файл проекта с расширением .elf;
3) При необходимости также могут формироваться вспомогательные файлы сборки (.hex, .dump и др.);
4) Если сборка завершилась без ошибок, проект готов к прошивке и отладке;
5) Нажимаем на Build Project.

**Прошивка микроконтроллера**

1) Убедитесь, что для проекта выбрана отладочная конфигурация Debug NIIET trng;

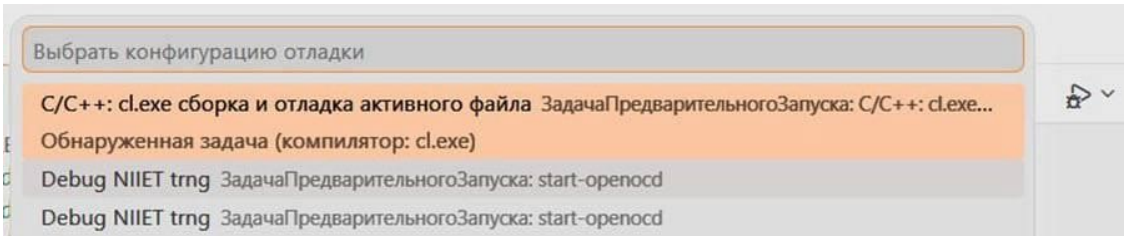



Рис. 7 - Debug

2) Для прошивки микроконтроллера запустите эту конфигурацию из списка отладки VS Code;

3) Дождитесь завершения загрузки программы в память микроконтроллера;

4) После запуска отладки можно использовать точки останова и пошаговое выполнение программы;

5) Нажать кнопку Reset.

**Важное замечание по Flash**

При работе с Flash необходимо **зарезервировать одну страницу памяти**, которая не используется программой.

Для этого в linker script уменьшается размер области Flash.

Например:

ROM (rwx) : ORIGIN = 0x80000000, LENGTH = 1020K - уменьшили на 4 КБ

Последняя страница памяти при этом:

- последняя страница Flash исключается из прошивки;
- компилятор туда больше ничего не кладет;
- может безопасно применяться для записи данных.

Если этого не сделать, при стирании страницы возможна порча программы, так как код может находиться в той же области Flash.

Если проект создается через расширение НИИЭТ VS Code, то linker script находится в каталоге проекта device/ldscripts. Используется файл k1921vg015_flash.ld.

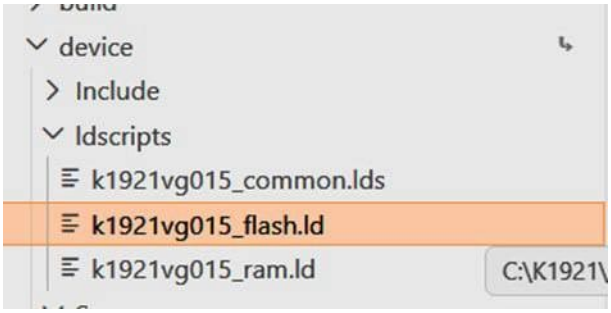



Рис. 8 - Расположение linker script в проекте

**ПРАКТИЧЕСКИЕ ЗАДАНИЯ**

**Общие правила для всех вариантов**

1. Файл main.c полностью не переписывать. Дописывать код только внутри while(1) { ... } и/или добавлять свои функции выше main();

2. В лабораторной работе должны быть реализованы и продемонстрированы все этапы содержания работы: инициализация GPIO для управления светодиодами, инициализация UART через retarget, вывод системной информации, выбор области Flash, реализация операций erase/write/read, интерфейс управления через UART и LED-индикация;

3. Для всех вариантов необходимо использовать одну заранее зарезервированную область Flash, не занятую программой. Адрес страницы для стирания и записи использовать как смещение, а чтение выполнять по абсолютному адресу вида 0x80000000 + offset;

4. Интерфейс управления UART: h – справка; t – полный тест (erase + write +read); e – стирание страницы; w – запись тестовых данных; p – чтение данных;

5. При записи учитывать, что базовый блок записи составляет 128 бит (4 слова по 32 бита). Формат пользовательских данных должен быть согласован с этим ограничением. Использование команды FLASH_CMD_WR;

6. Каждый вариант должен подтверждаться не только кодом, но и диагностикой: через UART выводить ход операции, адрес записи, содержимое данных, результат проверки и итоговый статус;

7. Индикация: LED5 – чтение; LED6 – стирание; LED7 – запись / готовность. Допускается расширение логики индикации, если это не мешает базовой демонстрации;

8. Результат: реализованы базовые операции и изучена адресация Flash, реализовано управление через UART, система корректно выполняет запись и чтение данных.

**Задание 11.1. Кольцевой лог во Flash**

**Цель:** реализовать последовательную запись данных во Flash (журнал событий).

**Что требуется реализовать:**

1. Функцию поиска свободного места;
2. Последовательную запись без перезаписи существующих данных;
3. Отображение всех записей через UART.

**Ход работы:**

1. Использовать одну страницу Flash как область хранения;
2. Разбить ее на блоки по 128 бит;
3. При каждой записи:

– искать следующую свободную запись (0xFFFFFFFF);

– записывать данные (например, номер записи или счетчик);

4. При заполнении страницы:

– выполнить erase;

– начать запись заново с начала.

**Ожидаемый результат:**

- Реализован журнал во Flash;
- Продемонстрирована последовательная запись данных;
- Показано понимание ограничения Flash: нельзя перезаписывать данные без erase.

**Задание 11.2. Контроль целостности данных**

**Цель:** научиться проверять корректность данных, сохраненных во Flash.

**Что требуется реализовать:**

1. Вычисление checksum;
2. Проверка данных после чтения;
3. Обработка случая поврежденных данных.

**Ход работы:**

1. Хранить структуру:

signature | data | checksum

2. Реализовать функцию вычисления контрольной суммы (например XOR);
3. При чтении:

– проверять совпадение checksum;

– выводить статус (VALID / INVALID).

**Дополнительно:**

При ошибке – автоматическая перезапись данных.

**Ожидаемый результат:**

- Реализован контроль целостности данных;
- Показано понимание необходимости защиты пользовательской информации во Flash.

**Задание 11.3. Энергонезависимый счетчик**

**Цель:** реализовать хранение данных во Flash между перезапусками.

**Что требуется реализовать:**

1. Чтение данных при старте;
2. Обновление и запись;
3. Обработка пустой/невалидной Flash.

**Ход работы:**

1. При старте программы:
– считать значение счетчика из Flash;
2. По команде:
– увеличить значение;
– сохранить обратно во Flash;
3. После перезапуска:
– значение должно сохраняться.

**Дополнительно:**

Добавить вывод текущего значения через UART.

**Ожидаемый результат:**

- Реализовано постоянное хранение данных;
- Показано практическое применение Flash-памяти для сохранения состояния.

### Задание 11.4. Двойное хранение (Wear Leveling)

**Цель:** снизить износ Flash за счет распределения записей.

**Что требуется реализовать:**

1. Определение актуальной записи при старте;
2. Выбор страницы для следующей записи;
3. Переключение между страницами.

**Ход работы:**

1. Использовать две страницы Flash;
2. Хранить данные поочередно:
– сначала в первой странице;
– затем во второй;
3. Каждая запись содержит:
– данные;
– номер версии (или счетчик).

**Что важно:**

- Не перезаписывать одну и ту же страницу постоянно;
- Корректно определять последнюю версию данных.

**Ожидаемый результат:**

- Реализовано распределение нагрузки на Flash;
- Показано понимание проблемы износа Flash-памяти и базовых методов ее уменьшения.

## КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Для чего в лабораторной работе резервируется отдельная страница Flash-памяти?

2. Чем отличается адрес, записываемый в FLASH->ADDR, от адреса, используемого для чтения через указатель?

3. Почему перед записью данных во Flash обычно требуется выполнить erase страницы?

4. Какой минимальный объем данных записывается в рассматриваемой лабораторной работе за одну операцию?

5. Зачем в программе одновременно используются UART и LED-индикация?

6. Почему опасно использовать для пользовательских данных область Flash, не исключенную из linker script?

7. Как определить первую свободную запись в странице Flash при организации журнала событий?

8. Для чего в варианте с checksum используется поле signature?

9. Как программа должна вести себя при обнаружении пустой или невалидной Flash в варианте с энергонезависимым счетчиком?

10. В чем смысл двойного хранения данных и использования номера версии при wear leveling?

## ТЕСТОВЫЕ ВОПРОСЫ

Выберите один правильный ответ.

**1. Какой объем внутренней Flash-памяти имеет микроконтроллер К1921ВГ015?**

А) 256 Кбайт.

Б) 512 Кбайт.

В) 1 Мбайт.

**2. Что необходимо сделать перед использованием последней страницы Flash для пользовательских данных?**

А) Отключить UART.

Б) Зарезервировать страницу через linker script.

В) Перевести код в SRAM1.

**3. Что передается в регистр FLASH->ADDR?**

А) Абсолютный адрес вида 0x80000000 + offset.

Б) Смещение внутри области Flash.

В) Только номер страницы без адреса.

**4. Как выполняется чтение ранее записанных данных из Flash?**

А) Только через UART-команду read.

Б) Через абсолютный адрес памяти и указатель.

В) Только через регистр FLASH->DATA.

**5. Какая команда в эталонной программе запускает полный тест erase + write + read?**

А) h.

Б) t.

В) p.

**6. Какой светодиод в базовом варианте соответствует операции стирания?**

А) LED5.

Б) LED6.

В) LED7.

**7. Какой объем данных записывается одной командой записи в лабораторной работе?**

А) 32 бита.

Б) 64 бита.

В) 128 бит.

**8. Почему нельзя просто перезаписать уже записанные данные во Flash новым значением?**

А) Потому что UART запрещает повторную запись.

Б) Потому что Flash требует erase перед изменением ранее записанных битов.

В) Потому что запись разрешена только один раз после прошивки.

**9. Что используется в варианте 11.2 для проверки целостности данных?**

А) PWM.

Б) Checksum.

В) Частота SYSPLL.

**10. Какой формат хранения данных предложен для варианта с контролем целостности?**

А) mode | timer | gpio.

Б) signature | data | checksum.

В) uid | pll | rtc.

**11. Что должно происходить в варианте «Кольцевой лог» при заполнении страницы?**

А) Программа завершается.

Б) Выполняется erase, затем запись начинается заново.

В) Страница переводится в режим только чтения.

**12. Что требуется продемонстрировать в варианте «Энергонезависимый счетчик»?**

А) Сохранение значения после перезапуска.

Б) Изменение частоты UART.

В) Переход в режим LP_STOP.

**13. Для чего в варианте Wear Leveling используются две страницы Flash?**

А) Чтобы хранить код и данные в одной странице.

Б) Чтобы распределять записи и уменьшать износ одной области.

В) Чтобы ускорить работу UART.

**14. Что помогает определить наиболее актуальную запись при двойном хранении?**

А) Цвет светодиода.

Б) Номер версии или счетчик.

В) Частота системного генератора.

**15. Какой файл проекта нужно изменить для безопасной работы с пользовательской страницей Flash?**

А) startup_k1921vg015.S.

Б) k1921vg015_flash.ld.

В) printf.c.

**16. Какой адрес используется для прямого чтения данных из Flash через указатель?**

А) Абсолютный адрес вида 0x80000000 + offset.

Б) Только значение регистра FLASH->ADDR без преобразования.

В) Адрес UART0.

**17. Что считается признаком свободной записи в варианте с кольцевым логом?**

А) Значение 0xFFFFFFFF.

Б) Нулевое значение всех слов.

В) Любое четное число.

**18. Для чего в варианте 11.3 при старте программы нужно читать данные из Flash?**

А) Чтобы определить текущее сохраненное значение счетчика.

Б) Чтобы автоматически изменить linker script.

В) Чтобы включить режим PowerOff.

**19. Что должна делать программа в варианте 11.2 после чтения данных из Flash?**

А) Сразу стирать страницу без проверки.

Б) Проверять checksum и выводить статус VALID или INVALID.

В) Переключать режим UART на SPI.

**20. Почему в варианте Wear Leveling не следует постоянно писать в одну и ту же страницу?**

А) Потому что это не позволяет использовать LED-индикацию.

Б) Потому что нужно распределять циклы записи и уменьшать износ Flash.

В) Потому что чтение из одной страницы запрещено контроллером.

# Глава 8. Аппаратные часы: RTC

## RTC (Real-Time Clock) в микроконтроллере

RTC (Real-Time Clock) – это встроенный счетчик реального времени, предназначенный для непрерывного отсчета времени независимо от работы основного ядра. Обычно RTC тактируется от низкочастотного источника, например от кварца 32,768 кГц, что обеспечивает малое энергопотребление и стабильный ход часов.

В данной работе RTC используется как секундный счетчик: регистр RTC_TIME хранит количество секунд с некоторого начала отсчета. В качестве начала отсчета удобно использовать Unix-время – количество секунд, прошедших с 1970-01-01 00:00:00 (UTC).

В отличие от таймеров общего назначения, модуль RTC ориентирован не на формирование коротких задержек или высокочастотных периодических событий, а на длительный и стабильный отсчет календарного времени. Такой модуль удобен в задачах часов, календарей, ведения журналов событий, формирования отметок времени и работы устройства в режимах пониженного энергопотребления.

Практическое значение RTC состоит в том, что счетчик времени увеличивается равномерно и может использоваться как единый источник времени для всей программы. При хранении значения в секундах упрощаются сравнение временных меток, расчет интервалов между событиями и вывод текущих даты и времени пользователю через UART-терминал.

**2.2 Односекундное событие (тик 1 Hz)**

Для формирования события один раз в секунду RTC настраивается таким образом, чтобы внутренний делитель образовывал частоту 1 Hz на основе низкочастотного тактового сигнала. При наступлении очередной секунды устанавливается флаг события в регистре событий или истории, например в RTC_HISTORY, и при соответствующей конфигурации может формироваться запрос прерывания.

После обработки события флаг подлежит сбросу. В большинстве периферийных блоков сброс выполняется записью логической единицы в соответствующий бит регистра события (write-one-to-clear) либо записью в отдельный регистр сброса. Конкретный способ определяется реализацией модуля и должен уточняться по заголовочным файлам и документации.

Выбор частоты 32,768 кГц для RTC является распространенным, поскольку это значение равно 2^15 и удобно делится до частоты 1 Гц. В результате модуль может формировать секундный тик без сложных вычислений и с минимальной нагрузкой на программу.

$$f_tick = f_RTC / N = 32768 / 32768 = 1\ Hz$$

$$T_tick = 1 / f_tick = 1\ s$$

Таким образом, каждый новый тик соответствует увеличению счетчика времени на одну секунду. Для программы это означает, что при обнаружении нового значения RTC_TIME можно считать, что наступил следующий момент календарного времени, который требуется обработать и вывести пользователю.

**2.3 Прерывания и программный опрос (polling)**

Классический способ обработки секундного тика RTC основан на аппаратных прерываниях: при каждом тике вызывается обработчик RTC_IRQHandler(). Для упрощения отладки в данной работе используется программный опрос: функция RTC_IRQHandler() присутствует в проекте, однако вызывается вручную из основного цикла. Внутри обработчика текущее значение RTC_TIME сравнивается с предыдущим, и при его изменении устанавливается программный флаг time_tick = 1.

Такой подход позволяет отлаживать логику работы часов без настройки контроллера прерываний, но требует достаточно частого выполнения основного цикла, то есть исключения длительных блокирующих задержек.

Подход polling удобен на этапе обучения и отладки, поскольку логика обработки события остается наглядной: программа сама регулярно проверяет состояние периферийного модуля и реагирует на изменение счетчика. Однако в реальных приложениях при большом количестве задач более предпочтительны аппаратные прерывания, так как они позволяют не опрашивать модуль постоянно и обеспечивают более предсказуемое время реакции.

**2.4 Преобразование Unix-времени в дату и время**

Для вывода времени пользователю применяется формат календарной даты YYYY-MM-DD HH:MM:SS. Поэтому значение RTC_TIME, содержащее число секунд, преобразуется в структуру даты и времени. При таком преобразовании учитываются длины месяцев и високосные годы: год считается високосным, если он кратен 4, однако годы, кратные 100, не являются високосными, за исключением годов, кратных 400.

В работе допускается использование готового модуля преобразования, например unix_time.h, либо реализация вспомогательных функций средствами пользователя.

При преобразовании Unix-времени сначала определяют количество полных суток, прошедших с начала отсчета, и остаток секунд внутри текущих суток. После этого остаток последовательно переводится в часы, минуты и секунды, а число суток – в календарные год, месяц и день с учетом длины месяцев и високосных лет.

$$D = floor(t \,/\, 86400)$$

$$h = floor\big((t \bmod 86400) \,/\, 3600\big), \quad m = floor\big((t \bmod 3600) \,/\, 60\big), \quad s = t \bmod 60$$

Високосным считается год, который кратен 400, либо кратен 4, но не кратен 100. Учет этого правила необходим для корректного определения даты при переходе через конец февраля и при обработке больших значений счетчика времени.

**2.5 Вывод в терминал по UART**

Вывод строки времени выполняется через UART. На практике часто применяется retarget printf – перенаправление стандартной функции printf в UART. После этого достаточно сформировать строку требуемого формата и передать ее в терминальную программу. Параметры последовательного порта должны соответствовать настройкам проекта.

Использование UART в данной работе позволяет наблюдать результаты работы программы без применения сложных средств отладки. Терминал на персональном компьютере отображает строки времени в реальном масштабе, поэтому пользователь может сразу проверить период обновления, корректность начального значения, смену минут, часов и даты, а также правильность работы функций преобразования времени.

# Практическая работа №9

# RTC как часы без будильника

**СОДЕРЖАНИЕ**

1. Цель и оборудование
3. Задание
4. Ход работы
4.1 Подготовка проекта
4.2 Настройка RTC на секунды
4.3 Организация 1 Hz тика через polling
4.4 Основной цикл: чтение времени, преобразование и вывод
4.5 Проверка работы
4.6 Пошаговая инструкция выполнения работы
5. Ожидаемый результат
6. Типовые ошибки и способы устранения
7. Практические задания
7.1 Вариант 1. Часы с аппаратными прерываниями RTC
7.2 Вариант 2. Часы с коррекцией времени
7.3 Вариант 3. Часы с будильником
7.4 Вариант 4. Часы с секундомером
8. Контрольные вопросы
9. Тестовые вопросы
10. Приложение. Пример исходного кода (main.c)

## ЦЕЛЬ И ОБОРУДОВАНИЕ

**Цель:** научиться использовать модуль RTC микроконтроллера К1921ВГ015 для отсчета времени в секундах и формирования односекундного тика (1 Hz).

**Оборудование:** макетно-отладочная плата с микроконтроллером К1921ВГ015; персональный компьютер; USB-кабель Type-C для подключения платы; среда разработки VS Code или иная IDE по методическим указаниям; терминальная программа для приема данных UART.

**ЗАДАНИЕ**

Практическая работа выполняется в соответствии со следующим содержанием:

1. Настроить RTC так, чтобы регистр RTC_TIME вел счет секунд; в качестве формата времени рекомендуется использовать Unix-время.

2. В конфигурации RTC включить односекундное событие – тик 1 Hz.

3. Организовать обработчик RTC_IRQHandler(), вызываемый циклически из основного цикла по схеме polling.

4. В обработчике RTC выполнять проверку изменения RTC_TIME, при необходимости сброс флага события и установку программного флага time_tick = 1.

5. В основном цикле циклически вызывать RTC_IRQHandler(); при time_tick == 1 считывать RTC_TIME, преобразовывать значение в дату и время, выводить строку формата YYYY-MM-DD HH:MM:SS по UART и сбрасывать time_tick.

Результат выполнения базового варианта: микроконтроллер реализует простые часы, текущее время выводится в терминал по UART, используется подход программного опроса (polling).

## ХОД РАБОТЫ

### 4.1 Подготовка проекта

На основе шаблона, использовавшегося в предыдущих работах, подготавливается новый проект либо используется готовый каркас с системными файлами startup, system, retarget и линковочным сценарием.

Перед переходом к реализации проверяется успешная сборка проекта и выполнение программы на плате. Рекомендуется убедиться в работоспособности вывода UART, например посредством передачи тестовой строки при старте.

После подключения платы к персональному компьютеру открывается UART-терминал, например PuTTY, TeraTerm или встроенный терминал VS Code. Для установления соединения в терминальной программе выбирается режим последовательного подключения (Serial), указывается номер COM-порта, соответствующий подключенной плате, и задаются параметры обмена, предусмотренные программой микроконтроллера.

### 4.2 Настройка RTC на секунды

Настройка включает выбор источника низкочастотного тактирования и делителя таким образом, чтобы RTC формировал секундный тик. Затем в регистр RTC_TIME записывается стартовое значение, например текущее Unix-время.

Пример фрагмента инициализации RTC (упрощенно, имена регистров приведены по заголовочному файлу K1921VG015.h):

```
void RTC_init(void)
{
// включаем внешний кварц 32.768 кГц (если используется на плате)
PMURTC->RTC_CFG0 |= (1 << 4); // EXTOSC = 1
// настройка делителя для 1 Hz
uint32_t reg_value = PMURTC->RTC_CFG0;
reg_value &= ~0x03; // очистить биты делителя
reg_value |= 0x01; // делитель 32768 -> 1 Hz
PMURTC->RTC_CFG0 = reg_value;
// установить начальное время (Unix seconds)
PMURTC->RTC_TIME = datetime_to_unix(&init_time);
// включить RTC
PMURTC->RTC_CFG0 |= (1 << 1); // RTCEN = 1
// очистить события (если требуется)
PMURTC->RTC_HISTORY = 0;
}
```

### 4.3 Организация 1 Hz тика через polling

В данной работе обработчик RTC_IRQHandler() не привязывается к аппаратному прерыванию. Он вызывается циклически из основного цикла и выполняет роль обработчика события. Наступление новой секунды определяется сравнением текущего значения RTC_TIME с предыдущим.

Пример обработчика:

```
volatile uint8_t time_tick = 0;
void RTC_IRQHandler(void)
{
static uint32_t last_rtc_time = 0;
uint32_t current_rtc_time = PMURTC->RTC_TIME;
if (current_rtc_time != last_rtc_time)
{
// при необходимости сбросить флаг события
PMURTC->RTC_HISTORY = 0;
time_tick = 1;
last_rtc_time = current_rtc_time;
}
}
```

### 4.4 Основной цикл: чтение времени, преобразование и вывод

В основном цикле постоянно вызывается RTC_IRQHandler(). При установленном флаге time_tick считывается значение RTC_TIME, выполняется преобразование в дату и время, после чего строка выводится в терминал.

Пример фрагмента main():

```
while (1)
{
RTC_IRQHandler(); // polling
if (time_tick)
{
uint32_t curr_time = PMURTC->RTC_TIME;
datetime_t dt;
unix_to_datetime(curr_time, &dt);
printf("%04u-%02u-%02u %02u:%02u:%02u\r\n",
dt.year, dt.month, dt.day, dt.hour, dt.minute, dt.second);
time_tick = 0;
}
```

```
}
```

### 4.5 Проверка работы

1. После завершения реализации проект подлежит сборке и прошивке микроконтроллера.

2. В UART-терминале необходимо убедиться, что строка времени выводится с периодом одна секунда.

3. Следует проверить корректность изменения даты и времени, включая смену минут, часов и суток. При необходимости выполняется отладка функций преобразования Unix-времени.

### 4.6 Пошаговая инструкция выполнения работы

Ниже приведена рекомендуемая последовательность выполнения работы, позволяющая поэтапно реализовать и проверить требуемую функциональность.

1. Открыть подготовленный проект для микроконтроллера К1921ВГ015 либо создать новый проект на основе рабочего шаблона с файлами startup, system, retarget и настройками сборки.

2. Проверить успешную компиляцию проекта без изменений и убедиться, что программа корректно загружается в микроконтроллер.

3. Настроить вывод сообщений через UART и убедиться, что терминальная программа принимает тестовые строки от платы.

4. Подготовить структуру данных для хранения календарных полей даты и времени, а также функции преобразования из Unix-времени в календарный формат и обратно.

5. Выполнить инициализацию RTC: выбрать источник тактирования, настроить делитель для получения 1 Hz тика и записать начальное значение в регистр RTC_TIME.

6. Реализовать функцию RTC_IRQHandler(), которая в режиме polling сравнивает текущее значение RTC_TIME с предыдущим и при наступлении новой секунды устанавливает программный флаг time_tick.

7. В основном цикле программы организовать регулярный вызов RTC_IRQHandler(), чтобы программа постоянно отслеживала появление нового секундного события.

8. При установленном флаге time_tick считывать текущее значение RTC_TIME, преобразовывать его в поля структуры даты и времени и формировать строку вывода в формате YYYY-MM-DD HH:MM:SS.

9. Собрать проект, выполнить прошивку платы и открыть UART-терминал для наблюдения за выводом часов в реальном времени.

10. Проверить корректность работы программы по нескольким последовательным сообщениям в терминале: вывод должен обновляться один раз в секунду, а время должно изменяться без пропусков и искажений.

**ОЖИДАЕМЫЙ РЕЗУЛЬТАТ**

В терминале UART отображаются текущая дата и текущее время в формате YYYY-MM-DD HH:MM:SS с периодом обновления одна секунда.

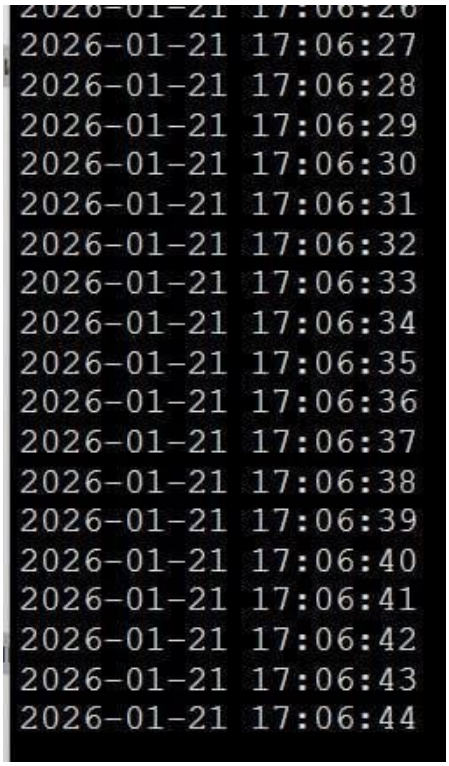



Рис. 1 – пример вывода (ожидаемый результат)

## ТИПОВЫЕ ОШИБКИ И СПОСОБЫ УСТРАНЕНИЯ

1. При отсутствии вывода в терминале необходимо проверить инициализацию UART и retarget, скорость порта и корректность подключения.

2. Если время не изменяется, следует убедиться, что RTC включен, выбран источник тактирования и правильно настроен делитель.

3. Если вывод формируется, но не с периодом одна секунда, требуется проверить настройку делителя RTC на 1 Hz и исключить длительные задержки в основном цикле.

4. При неверном отображении даты следует проверить корректность функций unix_to_datetime() и datetime_to_unix(), а также учет високосных лет и длины месяцев.

## ПРАКТИЧЕСКИЕ ЗАДАНИЯ

Дополнительные варианты заданий выполняются по выдаче преподавателя после реализации и проверки базового варианта работы.

### 7.1 Вариант 1. Часы с аппаратными прерываниями RTC

Краткое описание: реализуются часы с обработкой секундного тика по аппаратному прерыванию RTC.

**Требования:**

1. Настройка прерываний через RTC_CFG1.
2. Реализация RTC_IRQHandler() в качестве ISR.
3. Установка флага time_tick в обработчике прерывания.
4. Сброс флага события через RTC_HISTORY.
5. Индикация работы по LED0: светодиод мигает при каждом обновлении.

**Дополнительно**:

1. Реализация функции rtc_set_time().
2. Проверка переполнения счетчика времени, например с горизонтом 20 лет.
3. Вывод дня недели.

### 7.2 Вариант 2. Часы с коррекцией времени

Краткое описание: реализуются часы с возможностью программной коррекции скорости хода.

**Требования:**

1. Использование прерываний RTC по аналогии с вариантом 1.
2. Реализация коррекции через делитель RTC.
3. Реализация функции rtc_adjust_speed(int8_t ppm).
4. Поддержка UART-команд "FAST", "SLOW" и "NORMAL".
5. Индикация текущего режима с использованием LED0 и LED1.

**Дополнительно:**

1. Хранение параметров коррекции в энергонезависимой памяти.
2. Автоматическая компенсация отклонения хода.
3. Вывод текущей коррекции в секундах за сутки.

### 7.3 Вариант 3. Часы с будильником

Краткое описание: к базовому режиму часов добавляется функция будильника.

Требования:

1. Использование прерываний RTC.
2. Реализация будильника через RTC_ALARM.
3. Реализация функции rtc_set_alarm().
4. Обработка срабатывания будильника в прерывании.
5. Индикация срабатывания всеми пользовательскими светодиодами.

Дополнительно:

1. Поддержка UART-команд "ALARM HH:MM" и "ALARM OFF".
2. Ежедневный повтор будильника.
3. Отключение будильника кнопкой при наличии аппаратного подключения.

### 7.4 Вариант 4. Часы с секундомером

Краткое описание: к основному режиму часов добавляется режим секундомера на базе RTC.

**Требования:**

1. Основной режим реализуется как часы с прерываниями.
2. Дополнительный режим реализуется как секундомер на базе RTC.
3. Поддерживаются UART-команды "STOPWATCH START", "STOP" и "RESET".
4. Используется RTC_REG[0] для хранения значения.
5. Предусматривается переключение режимов работы.

**Дополнительно:**

1. Запись кругов, не более пяти.
2. Вывод лучшего времени.
3. Автоматический сброс через 24 часа.

**КОНТРОЛЬНЫЕ ВОПРОСЫ**

1. Для чего предназначен модуль RTC и чем он отличается от обычного таймера общего назначения?

2. Почему для RTC обычно используют источник 32,768 кГц? В чем преимущества такого тактирования?

3. Что хранится в регистре RTC_TIME при использовании Unix-времени? Какие преимущества дает такой формат?

4. Что такое событие 1 Hz тика и как оно формируется делителем RTC?

5. Чем отличаются аппаратные прерывания и программный опрос (polling)? Когда polling удобен, а когда его применение нежелательно?

6. Для чего в обработчике требуется сбрасывать флаг события? Что произойдет, если флаг не сбрасывать?

7. Для чего используется программный флаг time_tick и почему вывод времени нежелательно выполнять непосредственно в обработчике прерывания?

8. Какие особенности необходимо учитывать при преобразовании Unix-времени в календарную дату: високосные годы, длины месяцев, часовые пояса?

9. Каким образом можно проверить корректность работы RTC без внешнего эталонного времени?

10. Что произойдет при переполнении счетчика секунд RTC_TIME и каким образом такая ситуация может быть обработана программно?

**ТЕСТОВЫЕ ВОПРОСЫ**

Выберите один правильный ответ.

**1. Каково назначение регистра RTC_TIME в данной работе?**

А) Хранение количества секунд от выбранной точки отсчета.

Б) Хранение адреса обработчика прерывания.

В) Управление линиями UART.

**2. Для чего в RTC используется источник тактирования 32,768 кГц?**

А) Для удобного получения частоты 1 Hz делением и обеспечения стабильного хода часов.

Б) Для увеличения скорости передачи по UART.

В) Для работы таймера TMR32 в режиме PWM.

**3. Что означает событие 1 Hz тика RTC?**

А) Наступление новой секунды, фиксируемое внутренним делителем RTC.

Б) Передачу одного байта по UART.

В) Перезапуск ядра микроконтроллера.

**4. Какой подход используется в базовом варианте работы для обработки тика RTC?**

А) Программный опрос обработчика RTC_IRQHandler() из основного цикла.

Б) Только внешний аппаратный сигнал от WAKEUP0.

В) Исключительно DMA-обслуживание без участия процессора.

**5. Для чего в программе используется флаг time_tick?**

А) Для передачи факта наступления новой секунды из обработчика в основной цикл.

Б) Для хранения текущего значения месяца.

В) Для выбора скорости UART.

**6. Зачем после обработки события RTC требуется сброс флага события?**

А) Для корректной фиксации следующих событий и исключения повторной обработки старого флага.

Б) Для запуска кварцевого генератора.

В) Для записи времени во Flash-память.

**7. Какая функция выполняет перевод значения секунд в календарные поля даты и времени?**

А) unix_to_datetime().

Б) rtc_adjust_speed().

В) SystemCoreClockUpdate().

**8. В каком формате выводится время в терминал по UART?**

А) YYYY-MM-DD HH:MM:SS.

Б) DD/MM/YYYY, HH.MM.

В) Только количество секунд без преобразования.

**9. Какое условие должно выполняться для корректной работы polling-подхода?**

А) Основной цикл не должен блокироваться длительными задержками.

Б) Вся логика должна исполняться только внутри ISR.

В) Необходимо отключить UART.

**10. Что добавляется в варианте с будильником по сравнению с базовым вариантом?**

А) Использование RTC_ALARM и обработка события срабатывания будильника.

Б) Замена RTC на TMR32.

В) Полный отказ от вывода времени в терминал.

**11. Почему в RTC удобно использовать кварцевый генератор частотой 32,768 кГц?**

А) Потому что такую частоту удобно делить до 1 Гц без сложных коэффициентов.

Б) Потому что она предназначена только для UART.

В) Потому что она используется исключительно для работы DMA.

**12. Почему вывод времени обычно выполняется в основном цикле, а не внутри обработчика события?**

А) Чтобы не перегружать обработчик и не увеличивать время его выполнения.

Б) Чтобы отключить RTC после каждого тика.

В) Чтобы исключить использование флага time_tick.

**13. Что позволяет определить сравнение текущего значения RTC_TIME с предыдущим?**

А) Факт наступления новой секунды.

Б) Скорость передачи данных по UART.

В) Состояние кварцевого резонатора после прошивки.

**14. С какого этапа начинается преобразование Unix-времени в календарную дату?**

А) С определения числа полных суток и остатка секунд в текущих сутках.

Б) С переключения UART в режим приема.

В) С очистки регистра RTC_HISTORY.

**15. Что означает число 86400 в формулах преобразования времени?**

А) Количество секунд в одних сутках.

Б) Частоту тактирования RTC в герцах.

В) Максимальное количество секунд, которое может хранить RTC_TIME.

**16. Какое правило соответствует определению високосного года?**

А) Год кратен 400 либо кратен 4, но не кратен 100.

Б) Год кратен 10 и 12 одновременно.

В) Любой год, номер которого является нечетным.

**17. Какое условие необходимо для корректного отображения времени в терминале?**

А) Параметры UART в программе и терминале должны совпадать.

Б) RTC должен работать только в режиме аппаратных прерываний.

В) Основной цикл должен содержать длительную задержку.

**18. Для чего после обработки события может очищаться регистр RTC_HISTORY?**

А) Для подтверждения обработки текущего события и корректной фиксации следующего.

Б) Для установки нового значения времени в RTC_TIME.

В) Для изменения календарного формата вывода.

**19. Что считается правильным результатом выполнения базового варианта работы?**

А) Периодический вывод текущих даты и времени в терминал один раз в секунду.

Б) Непрерывная передача случайных чисел по UART.

В) Формирование только одиночного сообщения при запуске программы.

**20. Чем базовый вариант с polling отличается от варианта с аппаратными прерываниями?**

А) В базовом варианте событие проверяется программно из основного цикла, а в варианте с прерываниями обработчик вызывается аппаратно.

Б) В базовом варианте не используется регистр RTC_TIME.

В) В варианте с прерываниями невозможно вывести время через UART.

**ПРИЛОЖЕНИЕ. ПРИМЕР ИСХОДНОГО КОДА (MAIN.C)**

Ниже приведен пример реализации работы; листинг может быть адаптирован под конкретную структуру проекта.

Листинг main.c

```
/*=============================================================================
==========
#include <K1921VG015.h>
#include <stdint.h>
#include <stdio.h>
#include <system_k1921vg015.h>
#include "retarget.h"
//---------------------------------------------------------------------
// Глобальные переменные (как в задании)
//---------------------------------------------------------------------
volatile uint8_t time_tick = 0; // Программный флаг для 1-секундного тика
//---------------------------------------------------------------------
// Структура для хранения даты и времени
//---------------------------------------------------------------------
typedef struct {
uint16_t year;
uint8_t month;
uint8_t day;
uint8_t hour;
uint8_t minute;
uint8_t second;
} datetime_t;
//---------------------------------------------------------------------
// Функции для преобразования Unix-времени
//---------------------------------------------------------------------
#define EPOCH_YEAR 2000
static uint8_t is_leap_year(uint16_t year)
{
return ((year % 4 == 0 && year % 100 != 0) || (year % 400 == 0));
}
static const uint8_t month_days[2][12] = {
{31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31},
{31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31}
};
void unix_to_datetime(uint32_t unix_time, datetime_t *dt)
{
uint32_t days, seconds_in_day;
uint16_t year = EPOCH_YEAR;
uint8_t month = 0;
uint8_t leap;
days = unix_time / 86400;
seconds_in_day = unix_time % 86400;
dt->hour = seconds_in_day / 3600;
```

```
dt->minute = (seconds_in_day % 3600) / 60;
dt->second = seconds_in_day % 60;
while (days >= 365)
        {
leap = is_leap_year(year);
if (days >= (365 + leap))
        {
days -= (365 + leap);
year++;
        }
        else break;
        }
dt->year = year;
leap = is_leap_year(year);
for (month = 0; month < 12; month++)
        {
        uint8_t days_in_month = month_days[leap][month];
if (days < days_in_month)
        {
dt->month = month + 1;
dt->day = days + 1;
        break;
        }
days -= days_in_month;
        }
        }
uint32_t datetime_to_unix(datetime_t *dt)
        {
uint32_t total_days = 0;
        uint16_t year;
        uint8_t month;
for (year = EPOCH_YEAR; year < dt->year; year++)
        {
total_days += 365;
if (is_leap_year(year)) total_days += 1;
        }
for (month = 0; month < dt->month - 1; month++)
        {
total_days += month_days[is_leap_year(dt->year)][month];
        }
total_days += dt->day - 1;
uint32_t total_seconds = total_days * 86400UL;
total_seconds += dt->hour * 3600UL;
total_seconds += dt->minute * 60UL;
total_seconds += dt->second;
        return total_seconds;
        }
        //----------------------------------------------------------------
```

```
        // Инициализация RTC с 1 Hz тиком
        //---------------------------------------------------------------------
        void RTC_init(void)
        {
        // Включаем внешний кварц 32.768 кГц (1 Hz тик)
PMURTC->RTC_CFG0 |= (1 << 4); // EXTOSC = 1
        // Ждем стабилизации
for (volatile uint32_t i = 0; i < 10000; i++);
        // Настраиваем делитель для 1 Hz (32768/32768 = 1 Hz)
uint32_t reg_value = PMURTC->RTC_CFG0;
reg_value &= ~0x03; // Очищаем биты делителя
reg_value |= 0x01; // Делитель 32768 (01) - 1 Hz тик
PMURTC->RTC_CFG0 = reg_value;
        // Устанавливаем время 21.01.2026 17:05:00
        datetime_t init_time = {
        .year = 2026,
        .month = 1,
        .day = 21,
        .hour = 17,
        .minute = 5,
        .second = 0
        };
PMURTC->RTC_TIME = datetime_to_unix(&init_time);
        // Включаем RTC
PMURTC->RTC_CFG0 |= (1 << 1); // RTCEN = 1
        // "Настройка прерывания" - формально
        // В реальности polling, но для соответствия заданию:
PMURTC->RTC_CFG1 |= (1 << 0); // INT = 1 (разрешение прерываний)
        // "Сброс регистра событий"
PMURTC->RTC_HISTORY = 0;
        }
        //---------------------------------------------------------------------
        // "Обработчик прерывания RTC" (формально существует)
        //---------------------------------------------------------------------
        void RTC_IRQHandler(void)
        {
        // Формально: проверка флага события «1 секунда»
        // На самом деле эта функция вызывается вручную из main()
        // "Проверка флага события" - эмуляция
static uint32_t last_rtc_time = 0;
uint32_t current_rtc_time = PMURTC->RTC_TIME;
if (current_rtc_time != last_rtc_time)
        {
        // "Сброс флага" - эмуляция
PMURTC->RTC_HISTORY = 0;
        // Установка программного флага time_tick = 1 (как в задании)
time_tick = 1;
last_rtc_time = current_rtc_time;
```

```
        }
        }
        //-----------------------------------------------------------------
        // Функция задержки
        //-----------------------------------------------------------------
void delay(uint32_t cycles)
        {
while (cycles--) __asm("NOP");
        }
        //-----------------------------------------------------------------
        // Основная функция
        //-----------------------------------------------------------------
        int main(void)
        {
uint32_t curr_time = 0;
        datetime_t dt;
        SystemCoreClockUpdate();
        retarget_init();
printf("\r\n=======================================\r\n");
printf("Лабораторная работа 4: Часы на RTC\r\n");
printf("Микроконтроллер: К1921ВГ015\r\n");
printf("Формат: YYYY-MM-DD HH:MM:SS\r\n");
printf("Начальное время: 21.01.2026 17:05:00\r\n");
printf("=======================================\r\n\r\n");
        // Инициализация RTC с 1 Hz тиком
        RTC_init();
        // Основной цикл (полное соответствие пунктам задания)
while (1)
        {
        // "Вызов обработчика прерывания" (эмуляция прерывания)
        // В реальной системе это вызвалось бы аппаратно
        RTC_IRQHandler();
        // Как в задании: при time_tick == 1
if (time_tick == 1)
        {
        // Сброс флага time_tick (как в задании)
time_tick = 0;
        // Чтение RTC_TIME (как в задании)
curr_time = PMURTC->RTC_TIME;
        // Перевод значения в дату и время (как в задании)
        unix_to_datetime(curr_time, &dt);
        // Вывод строки формата YYYY-MM-DD HH:MM:SS через UART (как в задании)
printf("%04u-%02u-%02u %02u:%02u:%02u\r\n",
        dt.year, dt.month, dt.day,
        dt.hour, dt.minute, dt.second);
        }
        // Короткая задержка
        delay(10000);
```

```
}
}
```

# Блок RTC и домен PMURTC в составе К1921ВГ015

В работе используется блок часов реального времени RTC, входящий в состав микроконтроллера К1921ВГ015 и работающий в домене PMURTC. Этот домен сохраняет функциональность, необходимую для отсчета времени и формирования события пробуждения, даже при переводе микроконтроллера в глубокие режимы энергосбережения.

Практический смысл такого блока состоит в том, что даже при переводе микроконтроллера в PowerOff можно сохранить ход времени и организовать пробуждение по заранее заданному моменту. Поэтому в этой работе RTC рассматривается не как вспомогательный таймер, а как основной механизм возврата системы к выполнению программы.

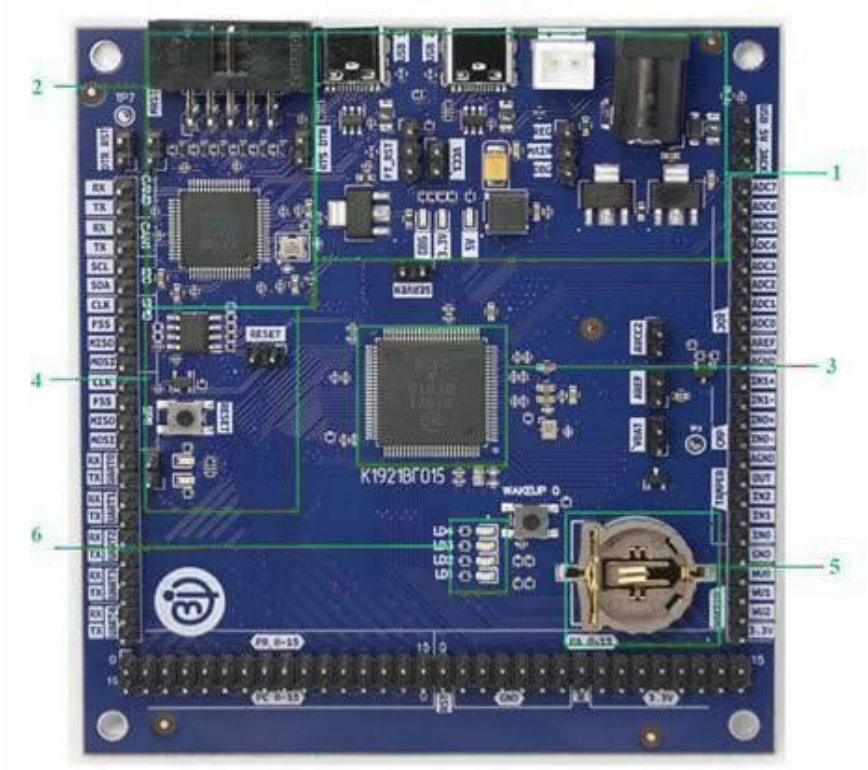



Рис.1

1 – блок источников питания;

2 – блок программатора;

3 – микросхема 1921ВГ015;

4 – сервисный блок;

5 – батарейный отсек CR1220;

6 – светодиодная индикация.

**2. Ключевые регистры и поля, используемые в работе**

Для стабильной работы RTC в учебном примере используется внешний кварц 32,768 кГц. Перед началом сценария требуется включить внешний источник тактирования RTC через бит EXTOSC, после чего часы реального времени начинают отсчитывать секунды независимо от основного кода программы.

В этой работе реально используются только несколько ключевых сущностей: текущее время RTC_TIME, регистр будильника RTC_ALARM, пользовательский регистр RTC_REG[0], бит EXTOSC для включения кварца, а также элементы перехода в PowerOff и последующего пробуждения. Именно их нужно понимать и контролировать в коде.

Если при старте RTC_ALARM равен нулю, это трактуется как первый запуск после подачи питания. Если значение уже не нулевое, значит до этого будильник был настроен и текущее выполнение связано с очередным пробуждением по времени.

| Регистр / поле | Назначение | Использование в работе | Комментарий |
|---|---|---|---|
| EXTOSC | Включает внешний кварц RTC 32,768 кГц. | Устанавливается перед запуском сценария RTC. | Без стабильного тактирования RTC корректный отсчет времени невозможен. |
| RTC_TIME | Хранит текущее значение времени RTC. | Используется как база при расчете следующего срабатывания будильника. | Обычно читается перед записью нового значения RTC_ALARM. |
| RTC_ALARM | Задает момент следующего пробуждения. | В коде записывается как RTC_TIME + 6 с или другой интервал по варианту. | Значение 0 на старте трактуется как первый запуск после подачи питания. |
| RTC_REG[0] | Сохраняет пользовательское состояние между пробуждениями. | По нему выбирают светодиод, счетчик, четность или направление. | Содержимое не теряется между циклами сна и пробуждения в рамках сценария. |
| DATAOUTTGL | Переключает состояние вывода GPIO. | Используется для мигания светодиодов без отдельного чтения текущего состояния. | Нужен для визуальной индикации работы без UART. |

Важно различать абсолютное время и интервал ожидания. В программу записывается не просто задержка, а новое абсолютное значение будильника: RTC_ALARM = RTC_TIME + Δt. За

счет этого пробуждение организуется аппаратно, без активного ожидания и без использования UART для контроля процесса.

**3. Переход в PowerOff и пробуждение по будильнику RTC**

Режим PowerOff является самым глубоким среди используемых в этой лабораторной работе. Перед входом в него необходимо заранее подготовить источник пробуждения, записать новое значение RTC_ALARM и только после этого выполнять переход в сон.

| Элемент | Роль при переходе в сон | Практический смысл |
|---|---|---|
| REGEN = 0 | Отключает регулятор питания при переходе в PowerOff. | Обеспечивает именно глубокий режим энергосбережения, требуемый в работе. |
| ALARMRST | Разрешает корректный возврат после события будильника RTC. | Используется вместе с RTC_ALARM, чтобы пробуждение происходило по времени, а не по внешнему сигналу. |
| WFI | Фактически переводит ядро в режим ожидания после завершения настройки. | Вызывается только после инициализации RTC, настройки будильника и сохранения требуемого состояния. |

В условии работы дополнительно используется отключение регулятора питания через REGEN = 0 и установка признака ALARMRST. Это позволяет организовать именно глубокий режим энергосбережения с последующим возвратом по будильнику RTC, а не обычную программную паузу.

**4. RTC_REG[0] и логика первого запуска/последующих пробуждений**

Пользовательский регистр RTC_REG[0] нужен для хранения состояния сценария между пробуждениями. В базовом варианте по младшим двум битам RTC_REG[0] выбирается один из светодиодов LED5–LED8, а само значение после каждого цикла изменяется по кольцу.

После каждого пробуждения значение RTC_REG[0] изменяется циклически: 0 -> 1 -> 2 -> 3 -> 0. Это позволяет визуально подтвердить, что состояние в RTC-домене не теряется даже после очередного перехода в PowerOff.

| RTC_REG[0] | Что делает программа после пробуждения | Следующее значение |
|---|---|---|
| 0 | Выбирается LED5, выполняется индикация по базовому сценарию. | 1 |
| 1 | Выбирается LED6, выполняется индикация по базовому сценарию. | 2 |
| 2 | Выбирается LED7, выполняется индикация по базовому сценарию. | 3 |
| 3 | Выбирается LED8, выполняется индикация по базовому сценарию. | 0 |
| Общее правило | После индикации снова вычисляется RTC_ALARM и выполняется переход в | Цикл повторяется аппаратно по будильнику |

| | | |
|---|---|---|
| | PowerOff. | RTC. |

Варианты работы строятся на изменении логики обработки RTC_REG[0] и интервала будильника. При этом общий принцип остается одинаковым: первый запуск определяется по RTC_ALARM, после пробуждения выполняется светодиодная индикация, затем заново настраивается RTC_ALARM и микроконтроллер снова уходит в сон.

Таким образом, работа объединяет аппаратный отсчет времени по RTC, глубокий сон PowerOff и сохранение пользовательского состояния в RTC_REG[0]. Наблюдение результата выполняется по светодиодам платы, без вывода диагностических сообщений через UART.

**5. Светодиодная индикация без использования UART**

Светодиодная индикация в этой работе является основным способом наблюдения результата. На первом запуске выполняется стартовая последовательность мигания, а после последующих пробуждений индикация определяется содержимым RTC_REG[0] и логикой конкретного варианта.

Диагностика первого запуска особенно важна для глубоких режимов сна: необходимо отличать начальное включение от очередного пробуждения по будильнику. Именно поэтому проверка RTC_ALARM и корректная светодиодная реакция являются обязательной частью всех вариантов работы.

# Практическая работа №10

# «RTC-будильник и пробуждение по времени микроконтроллера К1921ВГ015»

**СОДЕРЖАНИЕ**

1. Цель и оборудование
3. Структура проекта и основные файлы
4. Подготовка оборудования и программной среды
4.1 Подключение платы и конфигурация питания
4.2 Установка SDK и Toolchain
5. Создание, сборка и прошивка проекта
5.1 Создание проекта на основе примера
5.2 Подготовка файла main.c
5.3 Сборка проекта
5.4 Прошивка и запуск программы
6. Практические задания
6.1 Общие правила для всех вариантов
6.2 Задание 9.1. Двойной период (короткий/длинный)
6.3 Задание 9.2. Счетчик пробуждений с индикацией
6.4 Задание 9.3. Два светодиода и направление
6.5 Задание 9.4. Журнал последних пробуждений
7. Контрольные вопросы
8. Тестовые вопросы

## ЦЕЛЬ И ОБОРУДОВАНИЕ

**Цель:** освоить работу будильника RTC и перевод микроконтроллера К1921ВГ015 в глубокое энергосбережение с пробуждением по времени, научиться определять первый запуск после подачи питания, устанавливать момент срабатывания RTC_ALARM и использовать пользовательский регистр RTC_REG[0] для хранения состояния сценария.

**Оборудование:** отладочная плата с микроконтроллером К1921ВГ015; персональный компьютер с Windows; USB-кабель для подключения платы; установленная среда Visual Studio Code с расширением NIIET Aspect; SDK, toolchain и OpenOCD; документация на плату и руководство быстрого старта по К1921ВГ015.

## СТРУКТУРА ПРОЕКТА И ОСНОВНЫЕ ФАЙЛЫ

Основные файлы проекта можно увидеть на рисунке 2:

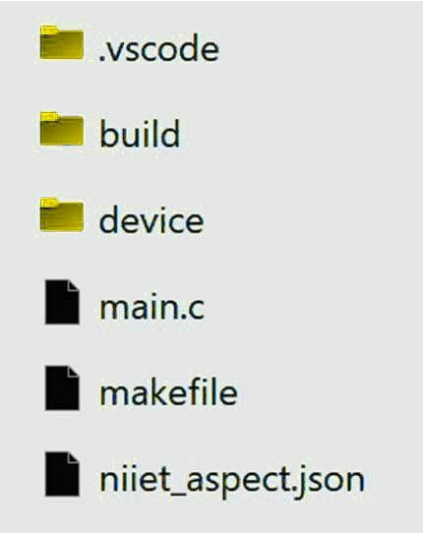



Рис. 2

| Категория | Файл | Назначение | Как работает |
|---|---|---|---|
| Основные файлы проекта | main.c | Ваш основной рабочий файл с кодом программы. | Пишете здесь всю логику для выполнения лабораторных работ. |
| Основные файлы проекта | makefile | Скрипт для автоматической сборки проекта. | При запуске команды make в терминале компилирует все .c файлы в .o, линкует их в .elf и формирует .hex/.dump для прошивки и отладки. |
| Основные файлы проекта | config.json | Конфигурация проекта для среды разработки. | Содержит настройки путей и параметров инструментария. |

| | | | |
|---|---|---|---|
| Основные файлы проекта | niiet_aspect.json | Специфичная конфигурация для инструментов NIIET Aspect. | Чтобы прошить плату во Flash, нужно заменить scriptFile с k1921vg015_ram.ld на k1921vg015_flash.ld. |
| Системные файлы (не трогайте без необходимости) | system_k1921vg015.c | Базовая системная инициализация. | Инициализирует системную частоту и служебные параметры МК. |
| Системные файлы (не трогайте без необходимости) | startup_k1921vg015.S | Стартовый код. | Запускается до main(), настраивает стек и таблицу векторов. |
| Системные файлы (не трогайте без необходимости) | printf.c | Поддержка printf и перенаправления вывода. | Используется для сообщений в UART. |
| Системные файлы (не трогайте без необходимости) | plic.c | Контроллер прерываний PLIC. | Содержит функции назначения и разрешения обработчиков прерываний. |
| Системные файлы (не трогайте без необходимости) | tmr32.c / tmr32.h | Драйвер 32-битного таймера. | Используется для настройки периодических событий и пробуждения по таймеру. |
| Системные файлы (не трогайте без необходимости) | gpio.c / gpio.h | Драйвер GPIO. | Нужен для управления светодиодами и приема внешнего сигнала пробуждения. |

Директории проекта:

| Директория | Назначение | Содержит |
|---|---|---|
| .vscode/ | Настройки среды разработки VSCode. | launch.json - настройки отладчика; tasks.json - задачи для сборки; settings.json - параметры редактора. |
| build/ | Директория для сгенерированных файлов. | .o - результат компиляции; .elf - исполняемый файл; .hex - файл прошивки; .dump - файл для отладки. |
| device/Include/ | Заголовочные файлы (.h). | K1921VG015.h - основной файл с определениями регистров МК. |
| device/Ldscripts/ | Скрипты линковщика. | k1921vg015_flash.ld; k1921vg015_ram.ld; k1921vg015.common.lds. |
| device/Source/ | Исходные системные файлы. | Системная инициализация, startup-код и базовые драйверы. |

## ПОДГОТОВКА ОБОРУДОВАНИЯ И ПРОГРАММНОЙ СРЕДЫ

### Подключение платы и конфигурация питания

1) Подключите плату К1921ВГ015 к компьютеру с помощью USB-кабеля. Порт USB Type C отмечен на рисунке 4;

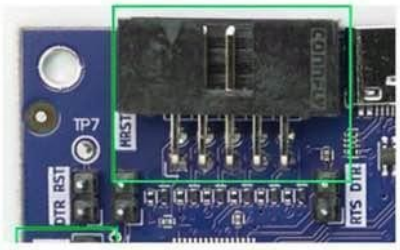






Рис. 3 Рис. 4

2) Установите перемычку USB 5V JACK в положение питания от USB;

3) Установите перемычку DBG MAIN REG *(рис. 3)* в положение питания микроконтроллера от программатора *(рис. 4)*

4) Проверьте, что перемычка FT_RST *(рис. 4)* установлена (сброс программатора снят);

5) Установка драйверов и настройка Visual Studio Code подробно описана в ПР1.

### Установка SDK и Toolchain

1) После установки расширения откройте панель NIIET Aspect. Из этого меню можно загрузить SDK, компилятор и инструменты отладки.

2) Для установки компонентов через расширение используйте пункт Download toolchain и выберите, что требуется: SDK, компилятор или отладочные инструменты.

3) Если архивы устанавливаются вручную, откройте Settings (Ctrl+,), введите в поиске niiet-aspect и задайте пути к SDK, компилятору, make и OpenOCD.

4) Без установленного SDK расширение не сможет создать новый проект и открыть пример.

## СОЗДАНИЕ, СБОРКА И ПРОШИВКА ПРОЕКТА

### Создание нового проекта на основе примера

Это обязательный шаг – нельзя просто открыть папку из архива Lab9.

1) Откройте VSCode;

2) В левой панели нажмите NIIET Aspect → Open Example;

3) Выберите пример: pmu (bare) *(рис. 5)*;

4) В параметре OpenOCD Debugger interface выберите: onboard_ftdi – это включает встроенный отладчик платы;

5) Нажмите Create и дождитесь, пока проект откроется.

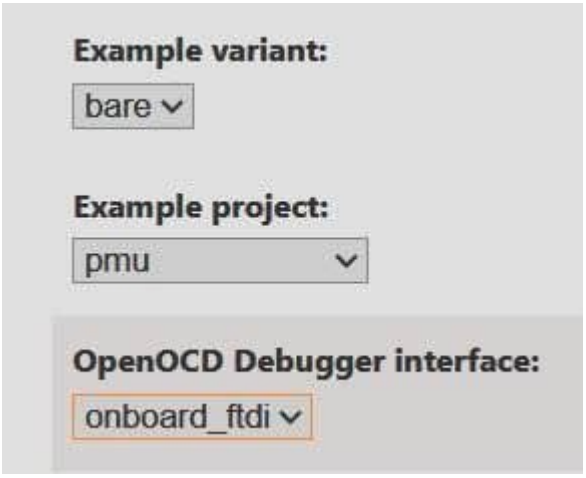



Рис. 5

Замена файла main.c

1) В VSCode откройте папку проекта;

2) Найдите файл: main.c;

3) Удалите его или переименуйте;

4) Откройте архив Lab9;

5) Скопируйте файл main.c из архива в папку проекта;

6) Сохраните файл.

Теперь ваш проект соответствует лабораторной работе.

**Сборка проекта**

1) Сборку проекта выполните через кнопку Compile project на панели NIIET Aspect или через соответствующую задачу VS Code;

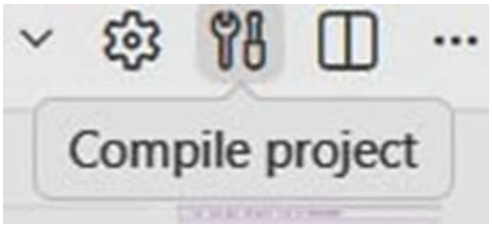



Рис. 6 – Компиляция проекта

2) После успешной компиляции в каталоге проекта создается исполняемый файл проекта с расширением .elf;

3) При необходимости также могут формироваться вспомогательные файлы сборки (.hex, .dump и др.);

4) Если сборка завершилась без ошибок, проект готов к прошивке и отладке;

5) Нажимаем на Build Project.

**Прошивка микроконтроллера**

1) Убедитесь, что для проекта выбрана отладочная конфигурация Debug NIIET trng;

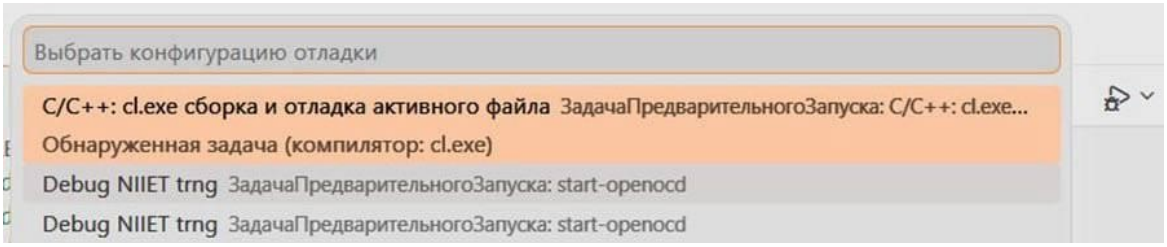



Рис. 7 - Debug

2) Для прошивки микроконтроллера запустите эту конфигурацию из списка отладки VS Code;

3) Дождитесь завершения загрузки программы в память микроконтроллера;

4) После запуска отладки можно использовать точки останова и пошаговое выполнение программы;

5) Нажать кнопку Reset.

**ПРАКТИЧЕСКИЕ ЗАДАНИЯ**

**Общие правила для всех вариантов**

1. Файл main.c не переписывать. Дописывать код только:
   - внутри while(1) { ... } (основное место);
   - и/или добавить свои функции выше main().
2. Определение первого запуска после подачи питания: проверка регистра RTC_ALARM (если равен 0 - первый запуск, иначе - пробуждение);
3. Инициализация RTC:
   - включение внешнего кварца для RTC через бит EXTOSC;
   - запись начального значения 0 в пользовательский регистр RTC_REG[0].
4. Реализация сценария:
   - при первом запуске мигаем всеми светодиодами 3 раза;
   - устанавливаем будильник RTC_ALARM = RTC_TIME + 6 секунд;
   - переводим МК в режим PowerOff с отключением регулятора питания (REGEN = 0) и установкой флага ALARMRST, после чего выполняем WFI для входа в сон.
5. После пробуждения:
   - по значению младших двух бит RTC_REG[0] (0..3) выбираем один из четырех светодиодов (LED5-LED8);
   - мигаем выбранным светодиодом 2 раза (переключение состояния через DATAOUTTGL);
   - инкрементируем RTC_REG[0] с маскированием по 0x03 (чтобы значение циклически менялось 0 -> 1 -> 2 -> 3 -> 0 ...);
   - снова устанавливаем будильник RTC_ALARM = RTC_TIME + 6 секунд и уходим в PowerOff.
6. Результат: микроконтроллер большую часть времени находится в глубоком энергосбережении и периодически просыпается по сигналу будильника RTC. При каждом пробуждении мигает разный светодиод, циклически меняя их по кругу.

**Задание 9.1. Двойной период (короткий/длинный)**

**Цель:** реализовать чередование двух интервалов пробуждения RTC - короткого и длинного.

**Пояснение:** микроконтроллер просыпается то через 3 секунды, то через 9 секунд, чередуя периоды. Визуально это видно по тому, как часто мигает светодиод.

**Ход работы:**

1. Первый запуск: мигаем всеми светодиодами 3 раза. Записываем RTC_REG[0] = 0 (флаг четности);

2. Пробуждение:

- Мигаем LED5 2 раза (всегда одним и тем же);
- Проверяем младший бит RTC_REG[0]:

- Если 0 (был четный запуск) -> устанавливаем следующий период = 3 секунды;
- Если 1 (был нечетный запуск) -> устанавливаем следующий период = 9 секунд.

- Инкрементируем RTC_REG[0] (просто +1, без маски).

1. Сон: устанавливаем будильник на вычисленный период и уходим в PowerOff.

**Что меняем в коде:**

- Период будильника теперь не константа, а вычисляется как (RTC_REG[0] & 1) ? 9 : 3;
- Счетчик растет неограниченно.

**Ожидаемый результат:**

Студент наблюдает чередование короткого и длинного интервалов между пробуждениями; LED5 мигает 2 раза, при этом следующее пробуждение происходит попеременно через 3 и 9 секунд.

**Задание 9.2. Счетчик пробуждений с индикацией**

**Цель:** реализовать счетчик пробуждений с индикацией количества пробуждений числом вспышек светодиода.

**Пояснение:** микроконтроллер считает количество пробуждений и индицирует это количество количеством вспышек светодиода.

**Ход работы:**

1. Первый запуск: мигаем всеми светодиодами 1 раз. Записываем RTC_REG[0] = 0;

2. Пробуждение:

- Берем значение RTC_REG[0];
- Мигаем LED5 столько раз, сколько хранится в младших 4 битах регистра (значение 0..15):

- Если RTC_REG[0] = 0 -> не мигаем совсем (пропускаем);
- Если RTC_REG[0] = 5 -> мигаем 5 раз.

- Инкрементируем RTC_REG[0];
- Если RTC_REG[0] достиг 16, сбрасываем в 0.

3. Сон: ставим будильник на 6 секунд и уходим в сон.

**Что меняем в коде:**

- Количество итераций в цикле мигания берется из RTC_REG[0];

- Появляется проверка на переполнение (16 -> 0).

**Ожидаемый результат:** при каждом пробуждении LED5 мигает число раз, равное текущему значению RTC_REG[0] в диапазоне 0..15; после 15 счетчик сбрасывается в 0 и цикл повторяется.

**Задание 9.3. Два светодиода и направление**

**Цель:** реализовать выбор светодиода и изменение направления по значению RTC_REG[0].

**Пояснение:** микроконтроллер работает как «бегущая точка» между двумя светодиодами (LED5 и LED8). Значение в регистре определяет не только какой светодиод, но и направление движения.

**Ход работы:**

1. Первый запуск: мигаем LED5 3 раза. Записываем RTC_REG[0] = 0 (бит 0 = светодиод, бит 1 = направление);
2. Пробуждение:
- Смотрим младшие 2 бита RTC_REG[0]:
- 0b00 = 0: LED5, двигаемся вправо (к LED8);
- 0b01 = 1: LED8, двигаемся влево (к LED5);
- 0b10 = 2: LED8, двигаемся вправо (к LED5);
- 0b11 = 3: LED5, двигаемся влево (к LED8).
- Мигаем выбранным светодиодом 2 раза;
- Изменяем RTC_REG[0] согласно направлению:
- Если двигаемся вправо: RTC_REG[0] = (RTC_REG[0] + 1) & 0x03;
- Если двигаемся влево: RTC_REG[0] = (RTC_REG[0] - 1) & 0x03.
3. Сон: будильник на 6 секунд, далее PowerOff.

**Что меняем в коде:**

- Логика изменения регистра зависит от текущего направления;
- Появляется арифметика с вычитанием.

**Ожидаемый результат:** при каждом пробуждении мигает LED5 или LED8, а состояние RTC_REG[0] меняется так, что направление выбирается по младшим двум битам и сценарий циклически повторяется.

**Задание 9.4. Журнал последних пробуждений**

**Цель:** реализовать хранение краткого журнала последних пробуждений в RTC_REG[0].

**Пояснение:** вместо частного сценария с пропуском событий микроконтроллер после каждого пробуждения сохраняет в RTC_REG[0] компактную историю последних состояний. Это позволяет

использовать RTC-домен не только для одного флага, но и для небольшого «журнала» работы между циклами PowerOff.

**Ход работы:**

1. Первый запуск: мигаем всеми светодиодами 2 раза. Записываем RTC_REG[0] = 0; далее считаем, что в младших 2 битах хранится текущий код светодиода (0..3), а в битах [15:8] - журнал последних четырех пробуждений.

**2. Пробуждение:**

Читаем текущий код светодиода: current = RTC_REG[0] & 0x03;

о По значению current выбираем один из четырех светодиодов LED5-LED8 и мигаем им 2 раза;

о Обновляем журнал: history = (history << 2) | current, после чего оставляем только последние 8 бит (четыре двухбитовые записи);

Вычисляем следующий код: current = (current + 1) & 0x03;

Собираем новое значение RTC_REG[0]: в младших 2 битах сохраняем current, а обновленный journal записываем в биты [15:8].

**3. Сон: устанавливаем будильник на 6 секунд и переводим МК в PowerOff.**

**Что меняем в коде:**

RTC_REG[0] используется как составная структура: часть битов хранит текущее состояние, часть - журнал последних пробуждений;

**Ожидаемый результат:** при каждом пробуждении по кругу мигают LED5-LED8, а в RTC_REG[0] сохраняется история последних четырех состояний, которую можно просмотреть в отладчике как компактный журнал работы.

**КОНТРОЛЬНЫЕ ВОПРОСЫ**

1. Для чего в лабораторной работе используется блок RTC микроконтроллера К1921ВГ015?

2. Почему значение RTC_ALARM = 0 трактуется как первый запуск после подачи питания?

3. Зачем перед началом сценария включать внешний кварц для RTC через бит EXTOSC?

4. Чем отличается запись RTC_ALARM = RTC_TIME + 6 от простой программной задержки на 6 секунд?

5. Почему настройка будильника должна выполняться до входа в режим PowerOff?

6. Для чего в базовом варианте используется пользовательский регистр RTC_REG[0]?

7. Почему после каждого пробуждения значение RTC_REG[0] изменяется циклически с маскированием по 0x03?

8. Какую роль играют биты REGEN и ALARMRST при переводе микроконтроллера в PowerOff?

9. Почему для мигания светодиодов удобно использовать переключение через DATAOUTTGL?

10. В чем состоит различие логики вариантов 9.1, 9.2, 9.3 и 9.4 по отношению к RTC_REG[0]?

## ТЕСТОВЫЕ ВОПРОСЫ

Выберите один правильный ответ.

**1. Какой регистр хранит текущее значение времени RTC?**

А) RTC_TIME.

Б) RTC_ALARM.

В) RTC_REG[0].

**2. Какой регистр задает момент следующего пробуждения?**

А) RTC_ALARM.

Б) RTC_TIME.

В) DATAOUTTGL.

**3. Что означает значение RTC_ALARM = 0 при старте программы?**

А) Первый запуск после подачи питания.

Б) Немедленное срабатывание будильника.

В) Переполнение FIFO.

**4. Через какой бит в условии работы включается внешний кварц RTC?**

А) EXTOSC.

Б) REGEN.

В) ALARMRST.

**5. Какой режим используется в лабораторной работе для глубокого энергосбережения?**

А) IDLE.

Б) PowerOff.

В) STOP.

**6. Что необходимо сделать перед выполнением WFI в этой работе?**

А) Сначала настроить RTC_ALARM и источник пробуждения.

Б) Удалить main.c.

В) Сбросить RTC_TIME в ноль без настройки будильника.

**7. Для чего используется RTC_REG[0] в базовом варианте?**

А) Для хранения состояния сценария между пробуждениями.

Б) Для хранения промежуточных данных в обычной SRAM.

В) Для измерения напряжения питания.

**8. Как выбирается светодиод в базовом варианте?**

А) По младшим двум битам RTC_REG[0].

Б) Случайным образом из TRNG.

В) По произвольной задержке в основном цикле.

**9. Что делает операция маскирования по 0x03 для RTC_REG[0]?**

А) Оставляет циклическое изменение 0 -> 1 -> 2 -> 3 -> 0.

Б) Обнуляет RTC_TIME.

В) Выключает внешний кварц.

**10. Какой аппаратный элемент платы связан с резервным питанием RTC-домена?**

А) Батарейный отсек CR1220.

Б) Разъем USB Type-C.

В) Встроенный программатор FT2232.

**11. Какие выводы используются для пользовательской светодиодной индикации на плате?**

А) PA12-PA15.

Б) PB0-PB3.

В) PC12-PC15.

**12. Что обеспечивает корректное пробуждение по времени после перехода в PowerOff?**

А) Предварительная запись RTC_ALARM и использование признака ALARMRST.

Б) Только переключение GPIO через DATAOUTTGL.

В) Только очистка build/ перед прошивкой.

**13. Почему состояние сценария удобно хранить в RTC_REG[0]?**

А) Его значение сохраняется между циклами PowerOff и последующими пробуждениями.

Б) Он автоматически увеличивает частоту RTC-кварца.

В) Он заменяет регистр RTC_TIME.

**14. Почему программная задержка не заменяет будильник RTC в этой работе?**

А) Она не обеспечивает реальный выход из PowerOff по времени.

Б) Она ускоряет работу кварца.

В) Она автоматически изменяет частоту кварца RTC.

**15. Как в этой лабораторной работе наблюдается выполнение сценария?**

А) По светодиодной индикации.

Б) По содержимому SPI-буфера.

В) По измерению CRC без индикации.

**16. Что записывается в RTC_ALARM в общем случае для организации следующего пробуждения?**

А) Абсолютное значение времени, вычисленное как RTC_TIME + интервал.

Б) Только число миганий светодиода.

В) Значение регистра DATAOUTTGL.

**17. Для чего в работе используется бит REGEN = 0?**

А) Для включения UART перед сном.

Б) Для отключения регулятора питания при переходе в PowerOff.

В) Для сброса RTC_REG[0].

**18. Что позволяет подтвердить сохранение состояния сценария между пробуждениями?**

А) Изменение содержимого RTC_REG[0] после каждого выхода из сна.

Б) Очистка файла build после каждой сборки.

В) Постоянное значение RTC_ALARM = 0.

**19. Почему в этой работе основной способ наблюдения результата – светодиоды, а не UART?**

А) Потому что UART запрещен во всех режимах микроконтроллера.

Б) Потому что сценарий ориентирован на глубокий сон и визуальную индикацию без текстового обмена.

В) Потому что RTC не может работать вместе с GPIO.

**20. Что происходит в базовом варианте после мигания выбранным светодиодом после пробуждения?**

А) Микроконтроллер полностью останавливает RTC.

Б) Программа инкрементирует RTC_REG[0], заново устанавливает RTC_ALARM и снова уходит в PowerOff.

В) Происходит переход в режим IDLE без настройки будильника.

# Часть IV. Энергетика и надежность: Профессиональная разработка



# Глава 9. Экономика энергии: PMU и режимы сна

## Блоки PMUSYS и PMURTC в составе К1921ВГ015

В работе №11 используются аппаратные блоки управления питанием PMUSYS и PMURTC. Они определяют, какие домены питания и тактирования остаются активными при выполнении инструкции WFI, как ведут себя внутренние LDO, и по каким событиям микроконтроллер должен выйти из сна.

Переход в энергосберегающий режим не сводится к одной команде «уснуть». Перед входом в сон требуется корректно настроить регистры PMU, подготовить источник пробуждения, включить нужные прерывания и предусмотреть, что именно должно восстановиться после возврата в активный режим.

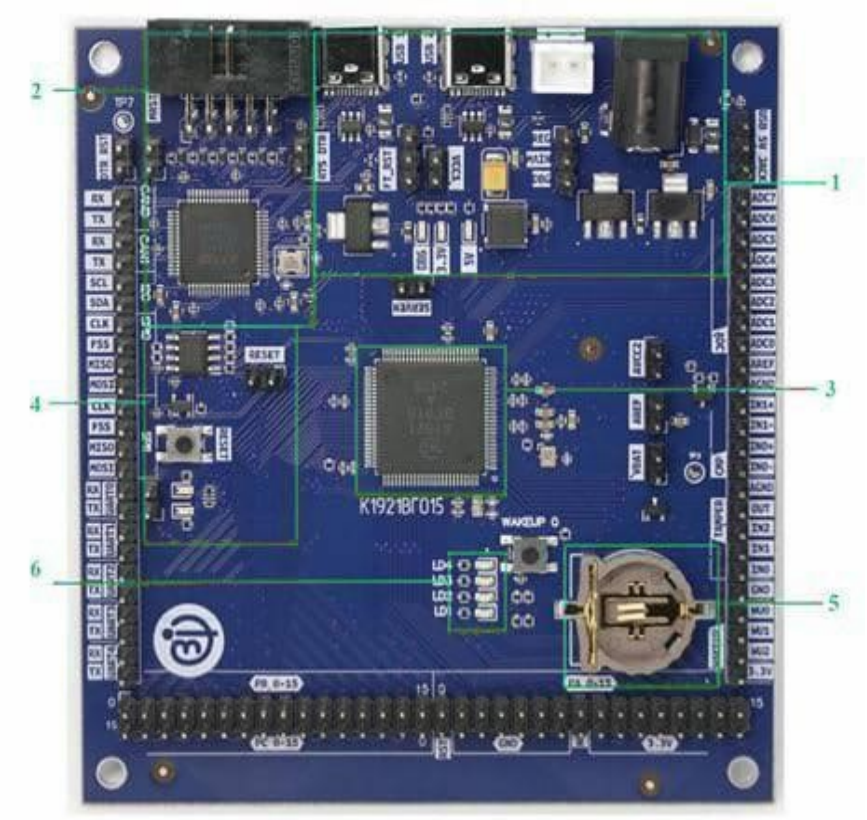



Рис.1

1 – блок источников питания;

2 – блок программатора;

3 – микросхема 1921ВГ015;

4 – сервисный блок;

5 – батарейный отсек CR1220;

6 – светодиодная индикация.

**2. Режимы IDLE, LP_IDLE, STOP, LP_STOP и POWEROFF**

При переходе в IDLE обычно сохраняется максимальная готовность периферии. Такой режим удобен как базовая точка сравнения: видно корректную работу WFI и быстрое пробуждение по таймеру.

**LP_IDLE** и **LP_STOP** применяются для исследования поведения микроконтроллера при снижении энергопотребления и изменении уровней внутренних **LDO**. Именно эти режимы напрямую связаны с вариантом 5.1, где необходимо сравнить поведение микроконтроллера при разных напряжениях.

**STOP** и **POWEROFF** относятся к более глубоким режимам. Для них особенно важно заранее определить допустимый источник пробуждения, сохранить минимально необходимую диагностику и понимать, какие узлы должны быть инициализированы повторно.

| Режим | Что происходит | Подготовка | Типовой признак |
|---|---|---|---|
| IDLE | Ядро прекращает активное выполнение, но большая часть цифровой периферии может продолжать работу. | Таймер, UART и обработчики прерываний должны быть настроены заранее. | Пробуждение происходит быстро, сообщения по UART выводятся почти без задержки. |
| LP_IDLE | Вариант ожидания с более низким потреблением и более жесткими требованиями к питанию. | До входа в режим нужно проверить конфигурацию LDO и источник пробуждения. | Ток снижается по сравнению с IDLE, выход из режима занимает больше времени. |
| STOP | Останавливается большее число тактовых доменов, чем в IDLE. | Разрешенный источник пробуждения должен быть включен до WFI. | После пробуждения полезно повторно вывести состояние системы в UART. |
| LP_STOP | Низкопотребляющий вариант STOP с дополнительным снижением питания. | Особенно важно корректно задать параметры WFI_PDEN и уровни LDO. | Сильнее всего заметно влияние напряжения на устойчивость и скорость выхода. |
| POWEROFF | Наиболее глубокий режим из рассматриваемых в работе. | Необходимо понимать, каким способом допускается выход и что требуется повторно | UART и светодиоды не должны использоваться как признак |

| | | | |
|---|---|---|---|
| | | инициализировать. | активности до возврата в рабочий режим. |

Режимы **IDLE, LP_IDLE, STOP, LP_STOP** и **POWEROFF** различаются глубиной энергосбережения, набором доступной после сна периферии и требованиями к восстановлению системы. В работе необходимо не просто вызвать WFI, а доказать по LED и UART, что выбранный режим действительно отрабатывает и корректно завершается пробуждением.

### 3. Регистры WFI_ENTR, WFI_EXIT и WFI_PDEN

Регистры PMURTC WFI_ENTR, WFI_EXIT и WFI_PDEN управляют входом в режим ожидания, условиями выхода из него и состоянием доменов питания. В лабораторной работе они должны рассматриваться как обязательная часть алгоритма SelectPowerMode(mode): сначала выбирается нужный режим и конфигурация LDO, затем включается источник пробуждения, после чего выполняется WFI.

| Регистр | Назначение | Проверить |
|---|---|---|
| WFI_ENTR | Определяет параметры входа в выбранный режим сна. | Соответствует ли фактический режим тому, что выбрано в меню UART. |
| WFI_EXIT | Задает условия возврата в активный режим. | Возвращается ли код в main/обработчик после события пробуждения без зависания. |
| WFI_PDEN | Связан с доменами питания и поведением внутренних LDO. | Не приводит ли выбранная конфигурация к потере устойчивости, невозможности прошивки или ошибкам пробуждения. |

Для вариантов с изменением напряжения удобно вынести настройку уровней LDO в отдельную функцию SetLDOVoltage(uint8_t vh, uint8_t vl). Это упрощает повторяемость эксперимента: перед каждым заходом в LP_IDLE или LP_STOP программа фиксирует выбранные уровни, переводит МК в сон, а после пробуждения сообщает по UART, какая конфигурация была использована.

### 4. Таймер TMR32 как источник пробуждения

**TMR32** в работе выполняет две ключевые функции. Во-первых, он обеспечивает периодическое пробуждение микроконтроллера без участия пользователя. Во-вторых, по его прерываниям удобно фиксировать длительность нахождения в режиме сна и автоматически переключать сценарии энергосбережения.

Последовательность использования TMR32 должна быть такой: настроить режим работы и период в CTRL/CAPCOM; разрешить запрос прерывания через IM; зарегистрировать обработчик TMR32_IRQHandler() в PLIC; разрешить глобальные прерывания; перед входом в сон убедиться, что таймер запущен; после пробуждения в обработчике обязательно сбросить флаг через IC.

| Регистр | Назначение | Использование |
|---|---|---|
| CTRL | Запуск и режим работы таймера. | Выбирается периодический режим, после чего таймер запускается перед WFI. |
| CAPCOM | Значение сравнения, определяющее период события. | По этому регистру задается интервал 2–3 секунды для варианта 5.2 либо иное требуемое время сна. |
| RIS | Флаг события таймера. | Позволяет убедиться, что источник пробуждения действительно сработал. |
| IC | Сброс флага события. | Очищается в обработчике сразу после фиксации пробуждения. |
| IM | Разрешение прерывания от таймера. | Используется при работе через IRQ; без него таймер не сможет вывести систему из сна через обработчик. |

**Важно:** если флаг таймера не очистить, повторное событие может не сформироваться корректно, а студент ошибочно решит, что проблема связана с режимом энергопотребления. Поэтому в TMR32_IRQHandler() сначала фиксируют событие и переключают сценарий, а затем очищают IC.

Для варианта 5.2 TMR32 становится главным управляющим узлом. По каждому его прерыванию программа должна последовательно переводить микроконтроллер в IDLE, LP_IDLE, STOP, LP_STOP и снова возвращаться к IDLE. LED-индикация используется для визуальной проверки текущего состояния, а UART - для текстового подтверждения после каждого пробуждения.

**5. Внешние источники пробуждения, UART и LED-индикация**

Согласно руководству по эксплуатации на плате предусмотрена кнопка аппаратного прерывания, сигнал которой выведен на WAKEUP0 и PA11. Пользовательская световая индикация подключена к линиям PA12–PA15. Это удобно для варианта 5.3: кнопку можно использовать как внешний источник пробуждения, а светодиоды - как визуальный индикатор выбранного режима и момента возврата из сна.

UART в данной лабораторной работе не является источником питания или таймирования. Его задача - протоколирование: вывести выбранный режим, время ожидания, источник пробуждения, уровни LDO и результат перехода. Чем глубже режим сна, тем важнее иметь такой журнал, потому что только LED-индикации недостаточно для диагностики ошибок конфигурации.

# Практическая работа №11

# Режимы энергопотребления блока PMU микроконтроллера К1921ВГ015

**СОДЕРЖАНИЕ**

1. Цель и оборудование
3. Структура проекта и основные файлы
4. Подготовка оборудования и программной среды
4.1 Подключение платы и конфигурация питания
4.2 Установка SDK и Toolchain
5. Создание, сборка и прошивка проекта
5.1 Создание проекта на основе примера
5.2 Подготовка файла main.c
5.3 Сборка проекта
5.4 Прошивка и запуск программы
6. Практические задания
6.1 Общие правила для всех вариантов
6.2 Задание 5.1. Влияние разных напряжений на низкопотребляющие
6.3 Задание 5.2. Автоматическая смена энергорежима по таймеру
6.4 Задание 5.3. Пробуждение от внешнего сигнала
6.5 Задание 5.4. Пользовательская комбинация энергорежимов
7. Контрольные вопросы
8. Тестовые вопросы

## ЦЕЛЬ И ОБОРУДОВАНИЕ

**Цель:** изучить режимы энергопотребления блока PMU микроконтроллера К1921ВГ015, научиться переводить микроконтроллер в режимы IDLE, LP_IDLE, STOP, LP_STOP и POWEROFF, а также организовывать пробуждение по таймеру TMR32 и внешним событиям.

**Оборудование:** отладочная плата с микроконтроллером К1921ВГ015; персональный компьютер с Windows; USB-кабель для подключения платы; установленная среда Visual Studio Code с расширением NIIET Aspect; SDK, toolchain и OpenOCD; мультиметр для варианта с измерением тока; при необходимости внешний сигнал или кнопка для проверки пробуждения.

## СТРУКТУРА ПРОЕКТА И ОСНОВНЫЕ ФАЙЛЫ

Основные файлы проекта можно увидеть на рисунке 2:

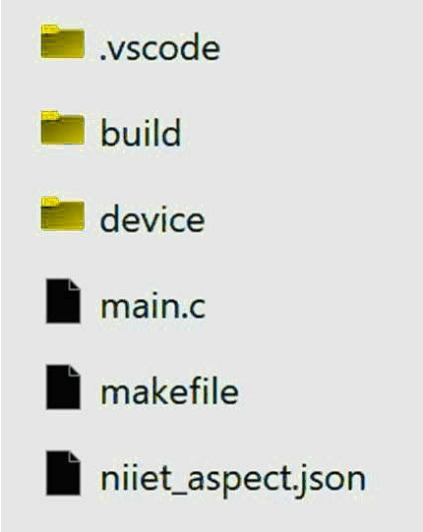



Рис. 2

| Категория | Файл | Назначение | Как работает |
|---|---|---|---|
| Основные файлы проекта | main.c | Ваш основной рабочий файл с кодом программы. | Пишете здесь всю логику для выполнения лабораторных работ. |
| Основные файлы проекта | makefile | Скрипт для автоматической сборки проекта. | При запуске команды make в терминале компилирует все .c файлы в .o, линкует их в .elf и формирует .hex/.dump для прошивки и отладки. |
| Основные файлы проекта | config.json | Конфигурация проекта для среды разработки. | Содержит настройки путей и параметров инструментария. |
| Основные файлы проекта | niiet_aspect.json | Специфичная конфигурация для инструментов NIIET Aspect. | Чтобы прошить плату во Flash, нужно заменить scriptFile с k1921vg015_ram.ld на k1921vg015_flash.ld. |
| Системные файлы (не трогайте без необходимости) | system_k1921vg015.c | Базовая системная инициализация. | Инициализирует системную частоту и служебные параметры МК. |
| Системные файлы (не трогайте без необходимости) | startup_k1921vg015.S | Стартовый код. | Запускается до main(), настраивает стек и таблицу векторов. |
| Системные файлы (не трогайте без | printf.c | Поддержка printf и перенаправления | Используется для сообщений в UART. |

| | | | |
|---|---|---|---|
| необходимости) | | вывода. | |
| Системные файлы (не трогайте без необходимости) | plic.c | Контроллер прерываний PLIC. | Содержит функции назначения и разрешения обработчиков прерываний. |
| Системные файлы (не трогайте без необходимости) | tmr32.c / tmr32.h | Драйвер 32-битного таймера. | Используется для настройки периодических событий и пробуждения по таймеру. |
| Системные файлы (не трогайте без необходимости) | gpio.c / gpio.h | Драйвер GPIO. | Нужен для управления светодиодами и приема внешнего сигнала пробуждения. |

**Директории проекта:**

| Директория | Назначение | Содержит |
|---|---|---|
| .vscode/ | Настройки среды разработки VSCode. | launch.json - настройки отладчика; tasks.json - задачи для сборки; settings.json - параметры редактора. |
| build/ | Директория для сгенерированных файлов. | .o - результат компиляции; .elf - исполняемый файл; .hex - файл прошивки; .dump - файл для отладки. |
| device/Include/ | Заголовочные файлы (.h). | K1921VG015.h - основной файл с определениями регистров МК. |
| device/Ldscripts/ | Скрипты линковщика. | k1921vg015_flash.ld; k1921vg015_ram.ld; k1921vg015.common.lds. |
| device/Source/ | Исходные системные файлы. | Системная инициализация, startup-код и базовые драйверы. |

## ПОДГОТОВКА ОБОРУДОВАНИЯ И ПРОГРАММНОЙ СРЕДЫ

### Подключение платы и конфигурация питания

1) Подключите плату К1921ВГ015 к компьютеру с помощью USB-кабеля. Порт USB Type C отмечен на рисунке 4;

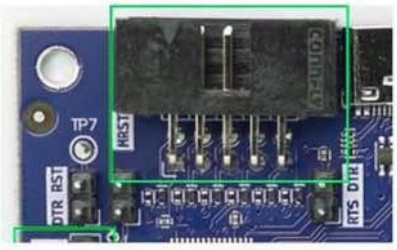






Рис. 3 Рис. 4

2) Установите перемычку USB 5V JACK в положение питания от USB;

3) Установите перемычку DBG MAIN REG *(рис. 3)* в положение питания микроконтроллера от программатора *(рис. 4)*

4) Проверьте, что перемычка FT_RST *(рис. 4)* установлена (сброс программатора снят);

5) Установка драйверов и настройка Visual Studio Code подробно описана в ПР1.

### Установка SDK и Toolchain

1) После установки расширения откройте панель NIIET Aspect. Из этого меню можно загрузить SDK, компилятор и инструменты отладки.

2) Для установки компонентов через расширение используйте пункт Download toolchain и выберите, что требуется: SDK, компилятор или отладочные инструменты.

3) Если архивы устанавливаются вручную, откройте Settings (Ctrl+,), введите в поиске niiet-aspect и задайте пути к SDK, компилятору, make и OpenOCD.

4) Без установленного SDK расширение не сможет создать новый проект и открыть пример.

## СОЗДАНИЕ, СБОРКА И ПРОШИВКА ПРОЕКТА

### Создание нового проекта на основе примера

**Это обязательный шаг – нельзя просто открыть папку из архива Lab5.**

1) Откройте VSCode;

2) В левой панели нажмите NIIET Aspect → Open Example;

3) Выберите пример: pmu (bare) *(рис. 5)*;

4) В параметре OpenOCD Debugger interface выберите: onboard_ftdi – это включает встроенный отладчик платы;

5) Нажмите Create и дождитесь, пока проект откроется.

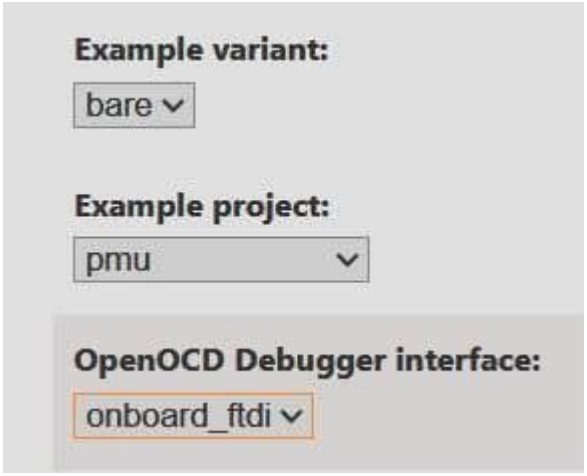



Рис. 5

Замена файла main.c

1) В VSCode откройте папку проекта;

2) Найдите файл: main.c;

3) Удалите его или переименуйте;

4) Откройте архив Lab5;

5) Скопируйте файл main.c из архива в папку проекта;

6) Сохраните файл.

Теперь ваш проект соответствует лабораторной работе.

**Сборка проекта**

1) Сборку проекта выполните через кнопку Compile project на панели NIIET Aspect или через соответствующую задачу VS Code;




Рис. 6 – Компиляция проекта

2) После успешной компиляции в каталоге проекта создается исполняемый файл проекта с расширением .elf;

3) При необходимости также могут формироваться вспомогательные файлы сборки (.hex, .dump и др.);

4) Если сборка завершилась без ошибок, проект готов к прошивке и отладке;

5) Нажимаем на Build Project.

**Прошивка микроконтроллера**

1) Убедитесь, что для проекта выбрана отладочная конфигурация Debug NIIET trng;

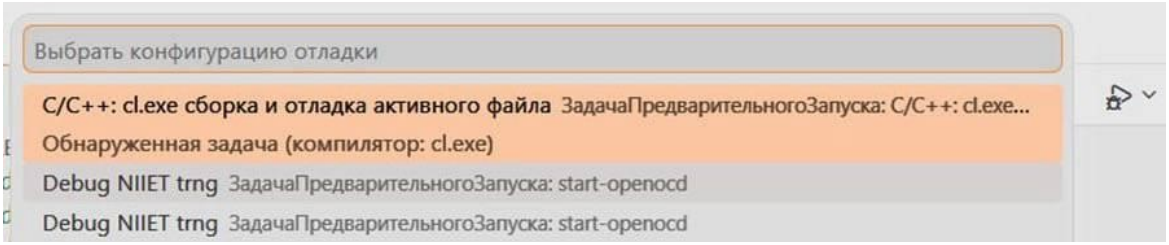



Рис. 7 - Debug

2) Для прошивки микроконтроллера запустите эту конфигурацию из списка отладки VS Code;

3) Дождитесь завершения загрузки программы в память микроконтроллера;

4) После запуска отладки можно использовать точки останова и пошаговое выполнение программы;

5) Нажать кнопку Reset.

## ПРАКТИЧЕСКИЕ ЗАДАНИЯ

**Общие правила для всех вариантов**

1. Файл main.c не переписывать. Дописывать код только:
   - внутри while(1) { … } (основное место);
   - и/или добавить свои функции **выше main()**.

2. В данной лабораторной работе используются блок PMU, таймер TMR32, UART0 и пользовательские светодиоды платы. До начала выполнения варианта необходимо проверить, что проект собирается без ошибок, плата прошивается, а в терминале UART появляются стартовые сообщения программы.

**3. Для всех вариантов обязательно соблюдать следующую последовательность действий:**

1) выбрать режим энергопотребления и источник пробуждения;

2) при необходимости настроить уровни LDO и период таймера TMR32;

3) перед входом в режим вывести диагностическое сообщение в UART;

4) перевести микроконтроллер в выбранный режим через WFI;

5) после пробуждения определить источник выхода и вывести результат в UART;

6) подтвердить работу варианта по LED-индикации и, если требуется заданием, по измерениям.

4. При использовании TMR32 в качестве источника пробуждения необходимо обязательно: задать период в CAPCOM, разрешить прерывание через IM, включить соответствующий канал в PLIC и в обработчике прерывания очищать флаг события через IC. Если флаг не очищается, повторное пробуждение может отработать некорректно.

5. Если в варианте используется внешний сигнал пробуждения, необходимо заранее проверить правильность выбранной линии WAKEUP, отсутствие конфликта с другой периферией и факт разрешения этого источника в конфигурации PMU. После выхода из режима программа должна явно сообщать в UART, что пробуждение произошло именно по внешнему событию.

6. Для индикации результата разрешается использовать пользовательские LED платы. Светодиодная индикация должна быть однозначной: по ней должно быть понятно, в какой режим вошла система, произошло ли пробуждение и какой этап сценария сейчас выполняется.

Пример вывода в UART при последовательном выборе энергорежимов показан на рисунке 8.

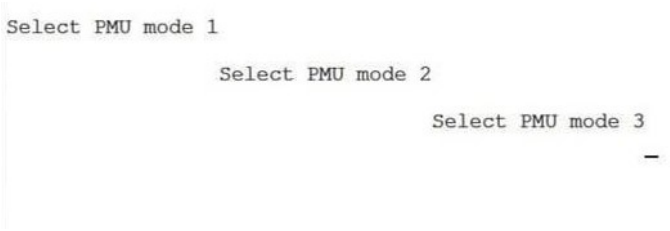



Рис. 8 - Пример вывода в UART

**Задание 5.1. Влияние разных напряжений на низкопотребляющие режимы**

**Цель:** изучить, как изменение уровней LDO влияет на устойчивость работы, ток потребления и скорость выхода из режимов LP_IDLE и LP_STOP.

**Пояснение:**

В этом задании не просто переводится микроконтроллер в сон, а сравнивается несколько режимов питания внутреннего ядра. Основная идея состоит в том, чтобы увидеть практическую связь между конфигурацией LDO, величиной потребляемого тока и поведением системы после пробуждения.

**Ход работы:**

1. Реализовать отдельную функцию SetLDOVoltage(uint8_t vh, uint8_t vl) для программного изменения напряжения LDO;
2. В LP_IDLE и LP_STOP установить несколько комбинаций напряжений (например, 1.0 В, 1.2 В, 1.3 В);
3. Измерить ток потребления МК (можно через мультиметр);
4. Пронаблюдать, как меняется скорость выхода из сна при разных напряжениях.

**Дополнительно:** вывести текущий уровень напряжения на UART при пробуждении.

**Ожидаемый результат:**

- Микроконтроллер стабильно входит в LP_IDLE и LP_STOP при нескольких значениях LDO;
- Получены наблюдения или измерения, показывающие изменение тока потребления и/или скорости выхода из сна;
- UART подтверждает, какая конфигурация напряжения использовалась в каждом эксперименте.

**Задание 5.2. Автоматическая смена энергорежима по таймеру**

**Цель:** реализовать динамическое переключение режимов энергопотребления без участия UART-команд пользователя.

**Пояснение:**

Задание направлено на использование TMR32 как основного механизма управления сценарием. Каждый таймерный IRQ должен будить систему, фиксировать завершение предыдущего шага и переводить микроконтроллер в следующий режим энергосбережения.

**Ход работы:**

1. Настроить TMR32 для генерации прерывания каждые 2–3 секунды.
2. В обработчике IRQ последовательно переключать режимы: IDLE → LP_IDLE → STOP → LP_STOP → IDLE.
3. LED-индикация должна отражать текущий режим.
4. Добавить вывод состояния на UART после каждого пробуждения.

**Дополнительно:** реализовать плавное мерцание светодиодов в IDLE/LP_IDLE, чтобы наглядно видеть активность таймера.

**Ожидаемый результат:**

- Микроконтроллер циклически проходит заданную последовательность режимов без ручного вмешательства.
- Пробуждение по TMR32 происходит повторяемо, а по UART видна полная последовательность переходов.
- LED-индикация однозначно соответствует текущему режиму.

**Задание 5.3. Пробуждение от внешнего сигнала**

**Цель:** показать работу режимов сна с пробуждением по кнопке или внешнему сигналу GPIO.

**Пояснение:**

Это задание демонстрирует отличие внешнего события от пробуждения по таймеру. Важно доказать, что линия действительно выступает источником пробуждения, а не просто меняет состояние GPIO уже после возвращения системы в активный режим.

**Ход работы:**

**Задачи:**

1. Использовать кнопку на плате или внешний сигнал для выхода из LP_IDLE или LP_STOP.
2. Настроить GPIO как источник прерывания.
3. В режиме LP_IDLE/LP_STOP МК засыпает и пробуждается по внешнему сигналу.

4. LED-индикация и UART должны показывать событие пробуждения.

**Дополнительно:** измерить время реакции МК от сигнала до включения светодиода.

**Ожидаемый результат:**

- Микроконтроллер устойчиво выходит из сна по внешнему сигналу.
- Событие пробуждения подтверждается и визуально, и текстово.

**Задание 5.4. Пользовательская комбинация энергорежимов**

**Цель:** создать собственный сценарий энергосбережения, включающий несколько режимов и способы возврата к активному коду.

**Пояснение:**

Задание ориентировано на самостоятельное проектирование сценария. Здесь важно продумать логику управления режимами, их длительность и диагностические сообщения так, чтобы переходы были прозрачны и воспроизводимы.

**Ход работы:**

1. Реализовать меню на UART для выбора нескольких режимов подряд (например, LP_IDLE на 5 секунд → STOP на 3 секунды → POWEROFF).
2. Настроить LED-индикацию для каждого режима.
3. Вывести на UART последовательность переходов и время нахождения в каждом режиме.

**Дополнительно:** добавить «быстрый тест» – кнопка сразу переводит МК в выбранный режим на фиксированное время.

**Ожидаемый результат:**

- Пользовательский сценарий выполняется полностью и без сбоев.
- По UART и LED-индикации можно восстановить всю последовательность переходов.
- Студент демонстрирует осознанное управление несколькими режимами сна и их источниками пробуждения.

**КОНТРОЛЬНЫЕ ВОПРОСЫ**

1. Для чего в микроконтроллере К1921ВГ015 используются блоки PMUSYS и PMURTC?
2. Чем режим IDLE отличается от LP_IDLE с точки зрения подготовки к пробуждению?
3. Почему перед выполнением WFI необходимо заранее настроить источник пробуждения?
4. Какую роль в лабораторной работе играют регистры WFI_ENTR, WFI_EXIT и WFI_PDEN?
5. Для чего в варианте 5.1 изменяются уровни внутренних LDO?
6. Почему таймер TMR32 удобен для автоматического пробуждения микроконтроллера?
7. Какие действия обязательно выполняются в обработчике прерывания TMR32 после срабатывания события?
8. Почему одного светодиода недостаточно для доказательства корректного прохождения эксперимента без сообщений UART?
9. Какой внешний источник пробуждения можно использовать на отладочной плате и какие выводы с ним связаны?
10. Что следует проверить, если после экспериментов с глубокими режимами сна плата перестала прошиваться по JTAG?

**ТЕСТОВЫЕ ВОПРОСЫ**

Выберите один правильный ответ.

**1. Какой блок в лабораторной работе отвечает за управление параметрами входа и выхода из режима WFI?**

А) GPIO.

Б) PMURTC.

В) CRC.

**2. Какой режим является самым глубоким из перечисленных?**

А) IDLE.

Б) LP_IDLE.

В) POWEROFF.

**3. Для чего в варианте 5.1 используется функция SetLDOVoltage()?**

А) Для изменения уровней внутренних LDO.

Б) Для очистки Flash-памяти.

В) Для перенастройки JTAG.

**4. Какой модуль удобнее всего использовать для периодического автоматического пробуждения?**

А) SPI.

Б) TMR32.

В) CRC.

**5. Какой регистр TMR32 задает период возникновения события?**

А) CAPCOM.

Б) DATAOUTSET.

В) OUTENSET.

**6. Что необходимо сделать с флагом таймера после обработки IRQ?**

А) Очистить его через IC.

Б) Перевести GPIO в режим входа.

В) Переименовать makefile.

**7. Какой интерфейс встроенной FT2232 на плате должен оставаться на штатном драйвере FTDI?**

А) Interface 0, используемый как UART0.

Б) Interface 1, используемый как JTAG.

В) Оба интерфейса переводятся на WinUSB.

**8. Какая линия платы связана с кнопкой аппаратного прерывания?**

А) WAKEUP0 / PA11.

Б) PA12 / PA13.

В) UART0_TX / UART0_RX.

**9. Для чего в лабораторной работе используется UART?**

А) Для питания платы.

Б) Для диагностических сообщений о режиме и пробуждении.

В) Для задания периода TMR32 без кода.

**10. Что требуется сделать для измерения тока по линии VCC1?**

А) Разомкнуть измерительный разъем и включить амперметр в разрыв.

Б) Подключить осциллограф к UART0.

В) Перевести DBG MAIN REG в случайное положение.

**11. Какой файл является основным для написания логики лабораторной работы?**

А) main.c.

Б) startup_k1921vg015.S.

В) config.json.

**12. Какое назначение имеет разъем «DBG MAIN REG»?**

А) Выбор источника питания микроконтроллера.

Б) Выбор периода таймера TMR32.

В) Перевод платы в сервисный режим.

**13. Что нужно изменить в niiet_aspect.json перед прошивкой во Flash?**

А) Заменить scriptFile с k1921vg015_ram.ld на k1921vg015_flash.ld.

Б) Удалить каталог build.

В) Переименовать main.c в startup.c.

**14. К каким выводам микроконтроллера подключена пользовательская световая индикация на плате?**

А) PA12–PA15.

Б) PB0–PB3.

В) WAKEUP0 и PA11.

**15. Что является обязательным общим правилом для всех вариантов?**

А) После пробуждения нужно подтвердить режим и источник события по UART/LED.

Б) Все системные файлы проекта надо переписать заново.

В) TMR32 нельзя использовать совместно с прерываниями.

**16. Какой регистр PMU связан с параметрами входа в выбранный режим сна?**

А) WFI_ENTR.

Б) WFI_EXIT.

В) WFI_PDEN.

**17. Что необходимо сделать перед входом микроконтроллера в сон через WFI?**

А) Обязательно удалить файл build.

Б) Заранее настроить источник пробуждения и нужные прерывания.

В) Отключить UART и все светодиоды без исключения.

**18. Какой режим в работе рассматривается как низкопотребляющий вариант STOP?**

А) IDLE.

Б) LP_STOP.

В) POWEROFF.

**19. Что должно подтверждать корректное пробуждение микроконтроллера в работе?**

А) Только наличие питания на USB.

Б) Только мигание одного светодиода без пояснений.

В) LED-индикация и диагностическое сообщение по UART.

**20. Для чего в варианте 5.2 используется LED-индикация?**

А) Для отображения текущего энергорежима при автоматическом переключении.

Б) Для изменения напряжения внутренних LDO.

В) Для задания частоты TMR32 без изменения регистров.

# Глава 10. Отказоустойчивость: Watchdog и обработка ошибок

**Интерфейс UART и обмен с терминалом**

UART (Universal Asynchronous Receiver/Transmitter) представляет собой асинхронный последовательный интерфейс, который применяется для обмена данными между микроконтроллером и внешним устройством, например персональным компьютером. Передача осуществляется по последовательной линии без отдельного тактового сигнала, поэтому обе стороны должны быть настроены на одинаковые параметры: скорость обмена, количество бит данных, число стоп-битов и режим контроля четности.

В данной работе UART используется как консольный интерфейс управления игрой. Через терминал пользователь отправляет команды старта и выхода, а микроконтроллер выводит текстовые сообщения о ходе игры, величине случайной задержки, моменте подачи сигнала и измеренном времени реакции. Такой подход удобен для отладки, поскольку позволяет наблюдать работу программы без создания отдельного графического интерфейса.

Для корректной работы терминала параметры соединения должны совпадать с настройками проекта. В архиве проекта скорость обмена установлена равной 115200 бод, а стандартная конфигурация порта соответствует режиму 8N1.

**2.2 Генератор истинных случайных чисел TRNG**

TRNG (True Random Number Generator) – это аппаратный генератор истинных случайных чисел, формирующий значения на основе физических процессов, а не по заранее заданному алгоритму. В отличие от программного псевдослучайного генератора, аппаратный TRNG позволяет получать непредсказуемые значения, что особенно важно в задачах, где необходимо исключить повторяемость поведения системы.

В рассматриваемой игре TRNG применяется для получения случайной задержки между началом раунда и подачей сигнала «реагируй». Благодаря этому пользователь не может заранее предугадать момент включения светодиодов, а измерение времени реакции становится более объективным. В исходном коде проекта выполняется инициализация модуля TRNG, ожидание появления данных во внутреннем FIFO и последующее преобразование полученного числа в требуемый временной интервал задержки.

**2.3 Светодиодная индикация и работа с GPIO**

Светодиоды платы подключены к линиям порта GPIO и используются как средство визуальной индикации состояния программы. В лабораторной работе задействованы светодиоды LED5–LED8. Во время ожидания сигнала мигает LED6, а в момент, когда пользователь должен отреагировать, включаются все рабочие светодиоды.

Управление светодиодами реализуется через запись в регистры порта GPIOA. В проекте предусмотрены функции включения, выключения и переключения состояния отдельного светодиода, а также функции одновременного управления всей группой индикаторов. Наличие отдельных вспомогательных функций упрощает чтение программы и делает алгоритм более наглядным.

### 2.4 Измерение времени реакции программным счетчиком

Для определения времени реакции в программе используется программный миллисекундный счетчик ms_counter. Его значение увеличивается в процессе выполнения задержек, за счет чего можно фиксировать момент подачи сигнала и момент нажатия клавиши пользователем. Разность между этими значениями принимается за время реакции.

Такой способ измерения не требует использования сложной периферии и хорошо подходит для учебной задачи. При старте раунда формируется случайная задержка, затем вычисляется момент подачи сигнала. После включения светодиодов значение счетчика сохраняется в переменной reaction_start. Когда пользователь нажимает клавишу, программа рассчитывает время реакции как разность текущего значения ms_counter и reaction_start.

### 2.5 Состояния игры и обработка ввода

Работа программы удобно описывается в виде конечного автомата. В исходном коде применяются три основных состояния: ожидание запуска игры, ожидание сигнала после старта и ожидание реакции пользователя после подачи сигнала. Такой подход позволяет разделить логику программы на независимые этапы и упростить ее сопровождение.

Считывание символов из UART выполняется в неблокирующем режиме. Это означает, что программа периодически проверяет наличие введенного символа, но не приостанавливает выполнение основного цикла в ожидании данных. Благодаря этому микроконтроллер продолжает обновлять индикацию и отслеживать момент подачи сигнала, даже если пользователь пока не нажал клавишу.

Перед началом нового раунда содержимое приемного буфера UART очищается. Это требуется для того, чтобы ранее введенные символы не воспринимались как новая команда или реакция игрока.

### 2.6 Вывод результата и оценка реакции

После нажатия клавиши программа выводит в терминал рассчитанное время реакции в миллисекундах и текстовую оценку результата. В эталонном варианте используются четыре категории: «Отлично», «Хорошо», «Средне» и «Медленно». Границы диапазонов заданы в условии лабораторной работы.

Текстовый вывод выполняется через функции передачи символов по UART. Такая организация позволяет формировать простое консольное меню, выводить служебные сообщения и информировать пользователя о ходе игры. Дополнительно в программе предусмотрена обработка преждевременного нажатия клавиши: если игрок нажал кнопку до момента сигнала, раунд завершается сообщением о раннем срабатывании.

### 2.7 Временные параметры игры и расчет показателей

При разработке игры на время реакции важную роль играют временные параметры программы. В эталонном варианте лабораторной работы состояние ожидания сопровождается миганием светодиода LED6 с частотой 2 Гц. Частота мигания определяет, сколько полных циклов включения и выключения происходит за одну секунду. Период мигания представляет собой длительность одного полного цикла и обратно пропорционален частоте. Для частоты 2 Гц период составляет 0,5 с, то есть светодиод меняет состояние через равные интервалы времени, обеспечивая наглядную индикацию режима ожидания.

$$T = 1 / f$$

$$T_\text{миг} = 1 / 2 = 0{,}5\ \text{с}$$

$$t_\text{реакции} = t_\text{нажатия} - t_\text{сигнала}$$

$$t_\text{задержки} \in [1100;\ 3600]\ \text{мс}$$

$$t_\text{ср} = (t_1 + t_2 + \ldots + t_n) / n$$

Время реакции вычисляется как разность между моментом нажатия клавиши пользователем и моментом подачи сигнала. Если игра выполняется серией из нескольких попыток, дополнительно может рассчитываться среднее время реакции. Такие формулы позволяют не только получить разовый результат, но и оценить устойчивость реакции пользователя в повторяющихся испытаниях.

### 2.8 Иллюстрации к взаимодействию модулей программы

На рисунках ниже показаны основные связи между модулями лабораторной работы: обмен с терминалом по UART, использование генератора TRNG для получения случайной задержки и

переходы между состояниями игры. Эти схемы помогают наглядно представить логику программы до анализа исходного кода.

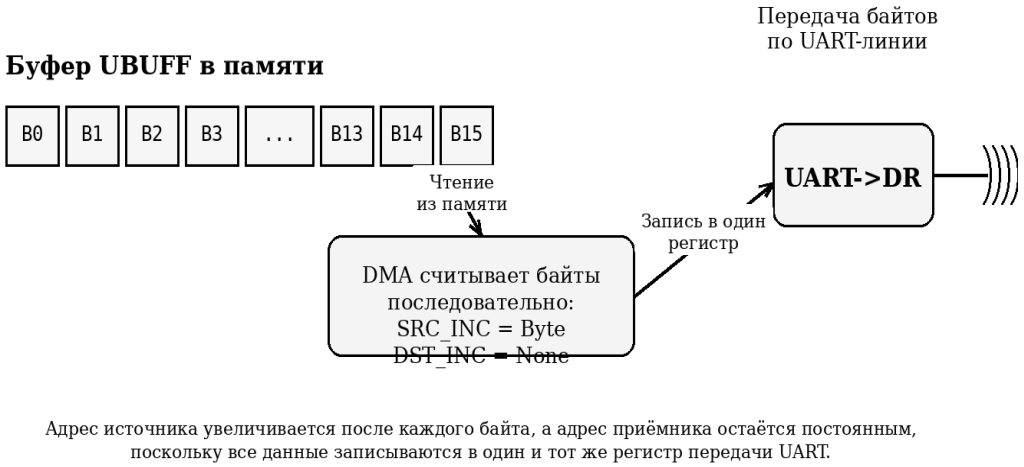



*Схема 1. Обмен данными между UART-терминалом и платой K1921VG015*




*Схема 2. Формирование случайной задержки и момента подачи сигнала*

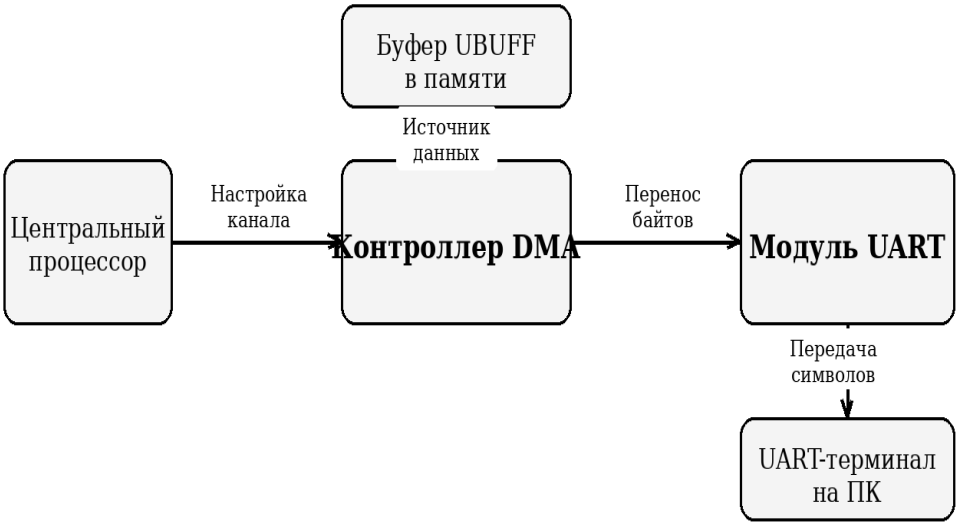



*Схема 3. Основные состояния игры на время реакции*

# Практическая работа 12

## Игра на время реакции с использованием TRNG, UART и светодиодной индикации

**СОДЕРЖАНИЕ**

1. Цель и оборудование
3. Задание
4. Дополнительные варианты заданий
4.1 Вариант 1. Игра с несколькими уровнями сложности
4.2 Вариант 2. Игра «Анти-реакция»
4.3 Вариант 3. Многопопыточная игра со статистикой
4.4 Вариант 4. Игра с индикацией на терминале
5. Ход работы
5.1 Подключение платы и подготовка проекта
5.2 Сборка проекта и загрузка программы
5.3 Настройка терминала и запуск игры
5.4 Анализ структуры исходного кода
5.5 Выполнение варианта задания
5.6 Подготовка материалов отчета
5.7 Пошаговое выполнение работы
6. Алгоритм работы программы
7. Пример вывода программы
8. Результат
9. Требования к отчету
10. Контрольные вопросы
11. Тестовые вопросы

## ЦЕЛЬ И ОБОРУДОВАНИЕ

**Цель:** изучить принципы построения простого интерактивного приложения на микроконтроллере K1921VG015 с использованием UART-интерфейса, генератора истинных случайных чисел TRNG, светодиодной индикации и программного измерения временных интервалов на примере игры для оценки времени реакции пользователя.

**Оборудование**: макетно-отладочная плата с микроконтроллером K1921VG015; персональный компьютер; USB-кабель для подключения платы; среда разработки для сборки и прошивки проекта; терминальная программа для обмена данными по UART.

## ЗАДАНИЕ

Выполнить работу согласно следующему содержанию:

• Создать игру для измерения времени реакции с UART-интерфейсом.

• Обеспечить управление через UART командами: s – старт, q – выход.

• Реализовать случайную задержку 1100–3600 мс с использованием TRNG.

• Организовать индикацию ожидания миганием LED6 с частотой 2 Гц.

• Использовать в качестве сигнала «реагируй» включение всех светодиодов.

• Измерять время реакции программным счетчиком.

• Оценивать результат по следующим диапазонам: < 180 мс – «Отлично»; 180–280 мс – «Хорошо»; 280–400 мс – «Средне»; свыше 400 мс – «Медленно».

## ДОПОЛНИТЕЛЬНЫЕ ВАРИАНТЫ ЗАДАНИЙ

### 4.1 Вариант 1. Игра с несколькими уровнями сложности

Отличия от эталона:

• Добавить выбор уровня сложности через UART: 1 – легкий (задержка 2000–4000 мс), 2 – средний (задержка 1500–3000 мс), 3 – сложный (задержка 1000–2000 мс).

• Реализовать различную индикацию для каждого уровня: легкий – мигает LED5; средний – попеременно мигают LED5 и LED6; сложный – бегущий огонь по всем светодиодам.

• Сделать сигнал «реагируй» зависящим от выбранного уровня сложности.

• Добавить бонусные баллы за быструю реакцию на сложных уровнях.

### 4.2 Вариант 2. Игра «Анти-реакция»

Отличия от эталона:

• Цель игры – не нажимать кнопку в запрещенный момент.

• Реализовать случайное окно длительностью 500–800 мс после задержки, в течение которого нажатие запрещено.

• Использовать индикацию: LED7 – ожидание, LED5 – запрещенная зона, LED8 – разрешенное нажатие.

• Назначить штрафные очки за нажатие в запрещенной зоне.

• Оценивать точность нажатия после выхода из запрещенной зоны.

### 4.3 Вариант 3. Многопопыточная игра со статистикой

Отличия от эталона:

• Организовать серию из 5 попыток подряд.

• Использовать случайные задержки 1000–4000 мс, разные для каждой попытки.

• Отображать номер попытки количеством горящих светодиодов.

• После каждой попытки выводить текущее время реакции.

• В конце серии рассчитывать и выводить лучшее время, худшее время, среднее время и коэффициент стабильности.

• Предусмотреть возможность сохранения рекорда в энергонезависимой памяти (опционально).

### 4.4 Вариант 4. Игра с индикацией на терминале

Отличия от эталона:

• Использовать UART для построения простой графики в терминале.

• Реализовать прогресс-бар ожидания в виде [#####-----].

• Добавить текстовую индикацию реакции.

• Предусмотреть вывод результатов последних 10 попыток по кнопке и краткую статистику.

• Реализовать клавиатурный тренажер: пользователь должен нажать случайно выбранную клавишу.

• Учитывать правильность нажатой клавиши при итоговой оценке.

## ХОД РАБОТЫ

### 5.1 Подключение платы и подготовка проекта

Подключить макетно-отладочную плату к персональному компьютеру по USB и убедиться в корректном определении интерфейсов устройства. Распаковать приложенный архив Lab10.zip и открыть в среде разработки проект trng, содержащий исходные файлы программы и конфигурацию сборки.

Проверить наличие в проекте файла main.c, в котором реализована основная логика игры, а также служебных файлов retarget.c и retarget.h, отвечающих за перенаправление стандартного вывода в UART.

### 5.2 Сборка проекта и загрузка программы

Выполнить сборку проекта стандартной командой Build и убедиться в отсутствии ошибок компиляции. После успешной сборки загрузить полученную прошивку в микроконтроллер с помощью штатной команды прошивки среды разработки.

После загрузки программы выполнить запуск платы и проконтролировать инициализацию модулей системы: UART для консольного обмена, GPIO для управления светодиодами и TRNG для формирования случайной задержки.

### 5.3 Настройка терминала и запуск игры

Открыть терминальную программу и подключиться к COM-порту, соответствующему подключенной плате. Параметры последовательного соединения установить в соответствии с проектом: скорость обмена 115200 бод, 8 бит данных, без контроля четности, 1 стоп-бит.

После открытия соединения в окне терминала появляется приглашение к началу игры. Для запуска раунда необходимо нажать клавишу s. Во время ожидания на плате мигает LED6, а после подачи сигнала в терминале появляется сообщение о необходимости немедленного нажатия любой клавиши. По завершении раунда выводится измеренное время реакции и оценка результата.

Команда q используется для завершения работы программы в режиме ожидания. При преждевременном нажатии любой клавиши до сигнала игра завершает текущую попытку и предлагает начать ее повторно.

### 5.4 Анализ структуры исходного кода

В файле main.c выделяются несколько логических частей: инициализация светодиодов, инициализация генератора случайных чисел, функции работы с UART, функции управления

состояниями игры и основной бесконечный цикл. Для вывода сообщений используются функции print_string и print_number, а для считывания символов – неблокирующее чтение из UART.

Функция start_game подготавливает новый раунд: очищает буфер UART, выключает светодиоды, получает случайную задержку и запоминает момент подачи сигнала. Функция update_game отвечает за мигание LED6 и переход к состоянию ожидания реакции, а функция handle_reaction рассчитывает время реакции пользователя и выводит оценку результата.

### 5.5 Выполнение варианта задания

Реализовать индивидуальный вариант работы в соответствии с номером задания. При доработке проекта сохранить общую структуру программы и использовать UART как основной интерфейс взаимодействия с пользователем. Корректность выполнения варианта проверить по сообщениям в терминале и по светодиодной индикации на плате.

### 5.6 Подготовка материалов отчета

Сделать скриншот работы программы с выводом терминала и включить его в отчет. В описании результата зафиксировать последовательность действий пользователя, пример полученного времени реакции и итоговую оценку, сформированную программой.

### 5.7 Пошаговое выполнение работы

1. Подключить макетно-отладочную плату к персональному компьютеру и убедиться, что в системе определился соответствующий COM-порт.

2. Распаковать архив с исходными файлами лабораторной работы и открыть проект в используемой среде разработки.

3. Проверить состав проекта: наличие файла main.c с основной логикой игры, а также служебных файлов для перенаправления стандартного ввода и вывода в UART.

4. Выполнить сборку проекта командой Build и убедиться в отсутствии синтаксических ошибок и ошибок компоновки.

5. Загрузить сформированную прошивку в микроконтроллер и перезапустить плату после завершения программирования.

6. Открыть UART-терминал, выбрать режим последовательного соединения, указать номер COM-порта платы и установить параметры 115200 бод, 8 бит данных, без четности, 1 стоп-бит.

7. Дождаться появления стартового приглашения в терминале и нажать клавишу s для запуска нового раунда игры.

8. Во время задержки наблюдать мигание LED6 и не нажимать клавиши до появления сигнала готовности.

9. После включения светодиодов нажать любую клавишу, получить рассчитанное время реакции и текстовую оценку результата.

10. Повторить запуск игры несколько раз, зафиксировать результаты, при необходимости выполнить индивидуальный вариант задания и подготовить материалы для отчета.

## АЛГОРИТМ РАБОТЫ ПРОГРАММЫ

Ниже приведен рекомендуемый каркас программы, отражающий общую организацию работы проекта из приложенного архива:

```
SystemInit();
retarget_init();
init_leds();
trng_init();
print_menu();

while (1)
{
if (game_active == 1)
update_game();

key = get_char_nonblock();

if (key == 's' && game_active == 0)
start_game();
else if ((key == 'q') && game_active == 0)
завершить работу;
else if (game_active == 2)
handle_reaction();
else if (game_active == 1)
вывести сообщение о преждевременном нажатии;

delay_ms(1);
}
```

Алгоритм программы включает инициализацию периферии, вывод стартового меню, ожидание команды запуска, формирование случайной задержки, подачу сигнала с одновременным включением светодиодов и расчет времени реакции по моменту нажатия клавиши. Такая структура позволяет разделить этапы игры и делает программу удобной для расширения при выполнении дополнительных вариантов задания.

**ПРИМЕР ВЫВОДА ПРОГРАММЫ**

Ниже приведен пример консольного вывода игры в терминале UART. После нажатия клавиши s начинается новый раунд, затем пользователь ожидает сигнала и нажимает любую клавишу. После этого программа выводит значение времени реакции и словесную оценку результата.

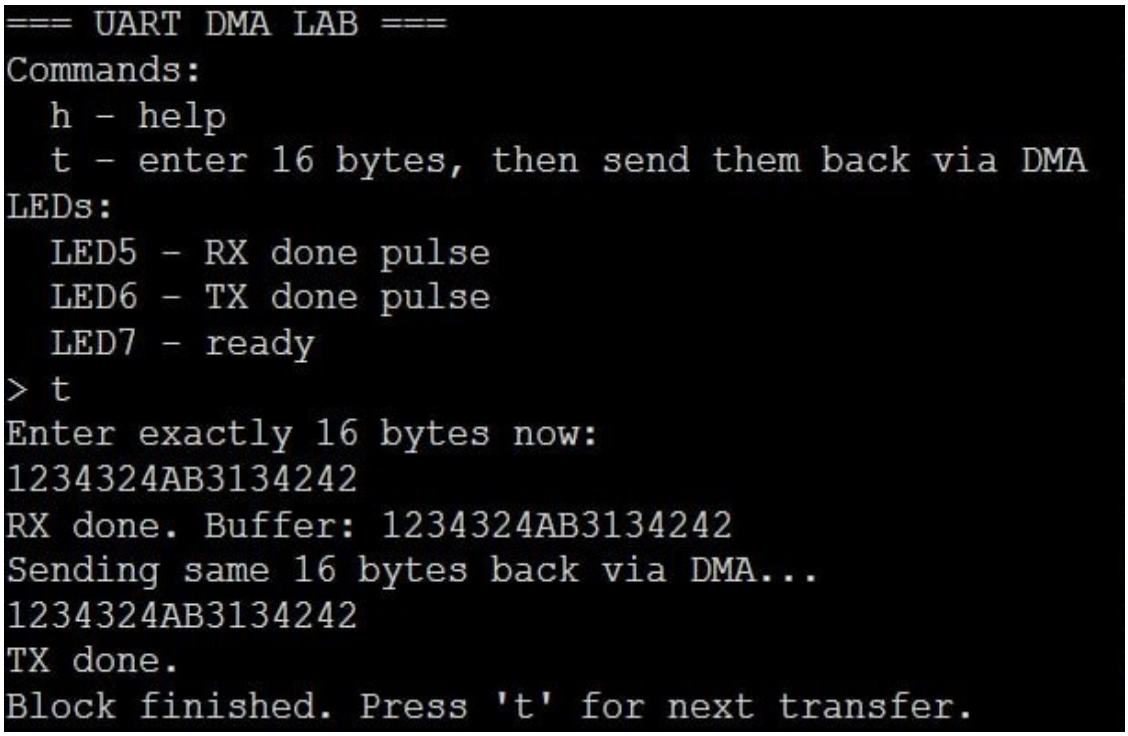



*Рис. 1. Пример вывода программы в терминале UART*

**РЕЗУЛЬТАТ**

Реализована игра для измерения времени реакции пользователя с управлением через UART. Программа использует аппаратный генератор TRNG для формирования случайной задержки, светодиодную индикацию для обозначения состояния игры и программный счетчик для измерения времени реакции.

Полученный результат подтверждает корректную совместную работу модулей UART, TRNG и GPIO в составе прикладной программы на микроконтроллере K1921VG015. Консольный интерфейс обеспечивает управление игрой и вывод результатов, а структура исходного кода позволяет расширять функциональность при выполнении дополнительных вариантов задания.

## ТРЕБОВАНИЯ К ОТЧЕТУ

• Титульные данные: номер и название работы, ФИО, группа, дата выполнения.

• Краткое описание принципа работы игры: назначение UART, TRNG, светодиодной индикации и программного измерения времени.

• Описание структуры программы: основные функции, состояния игры и порядок обработки ввода пользователя.

• Результаты работы: скриншот вывода терминала, пример измеренного времени реакции и подтверждение выполнения индивидуального варианта задания.

• Ответы на контрольные вопросы.

## КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Для чего в данной лабораторной работе используется интерфейс UART?

2. Почему для игры на реакцию целесообразно применять аппаратный генератор TRNG?

3. Что означает неблокирующее чтение символа из UART и почему оно важно для данной программы?

4. Каким образом программа определяет момент начала отсчета времени реакции?

5. Зачем перед стартом нового раунда очищается буфер приема UART?

6. Как изменится поведение программы при нажатии клавиши до момента подачи сигнала?

7. Почему в игре используется мигание светодиода в режиме ожидания сигнала?

8. Что изменится в работе программы, если параметры UART-терминала не совпадут с настройками прошивки?

9. Для чего в программе удобно использовать конечный автомат состояний?

10. Почему измерение времени реакции в лабораторной работе выполняется в миллисекундах?

**ТЕСТОВЫЕ ВОПРОСЫ**

Выберите один правильный ответ.

**1. Какой модуль микроконтроллера используется для формирования случайной задержки в игре?**

А) UART.

Б) TRNG.

В) GPIO.

**2. Какая команда запускает новый раунд игры через UART?**

А) q
Б) s
В) r

**3. Как в эталонном варианте обозначается состояние ожидания сигнала?**

А) Все светодиоды постоянно включены.
Б) Мигает LED6.
В) Светодиоды полностью выключены и не изменяются.

**4. Что является сигналом к началу реакции пользователя?**

А) Выключение UART.
Б) Включение всех светодиодов.
В) Очистка буфера UART.

**5. Как вычисляется время реакции в программе?**

А) По разности между текущим значением счетчика и моментом подачи сигнала.
Б) По номеру нажатой клавиши.
В) По количеству включенных светодиодов.

**6. Для чего используется функция clear_uart_buffer() перед стартом игры?**

А) Для удаления ранее принятых символов из буфера UART.
Б) Для отключения терминальной программы.
В) Для сброса генератора TRNG.

**7. Что произойдет при нажатии клавиши до подачи сигнала?**

А) Программа автоматически засчитает результат как «Отлично».
Б) Будет выведено сообщение о преждевременном нажатии, и раунд завершится.
В) Игра продолжится без изменений.

**8. Какая оценка выводится программой при очень быстром результате менее 180 мс?**

А) Медленно.
Б) Средне.
В) Отлично.

**9. Для чего в основном цикле программы вызывается update_game()?**

А) Для обновления состояния ожидания и проверки момента подачи сигнала.
Б) Для компиляции проекта.
В) Для настройки COM-порта.

**10. Какое преимущество дает использование UART-терминала в данной работе?**

А) Позволяет управлять игрой и наблюдать результаты без отдельного графического интерфейса.

Б) Полностью заменяет модуль TRNG.

В) Измеряет время реакции без участия программы.

**11. Для чего в лабораторной работе используется модуль GPIO?**

А) Для управления светодиодной индикацией.

Б) Для генерации случайных чисел.

В) Для хранения статистики в терминале.

**12. Какой параметр UART определяет скорость обмена данными между платой и терминалом?**

А) Частота мигания LED6.

Б) Baud rate.

В) Номер вывода GPIO.

**13. Что произойдет, если пользователь нажмет клавишу только после подачи сигнала?**

А) Программа вычислит время реакции.

Б) Генератор TRNG остановится.

В) Светодиоды перестанут быть доступны из GPIO.

**14. Какова основная роль случайной задержки перед сигналом?**

А) Сделать момент сигнала непредсказуемым.

Б) Ускорить работу UART.

В) Уменьшить объем исходного кода.

**15. Для чего в игре полезно очищать буфер UART перед новым раундом?**

А) Чтобы прежние символы не воспринимались как новая реакция.

Б) Чтобы изменить частоту мигания светодиодов.

В) Чтобы включить режим отладки TRNG.

**16. Какой светодиод используется в эталонном варианте для индикации ожидания?**

А) LED5.

Б) LED6.

В) LED8.

**17. Что из перечисленного относится к преимуществам аппаратного TRNG по сравнению с программным генератором?**

А) Непредсказуемость получаемых значений.

Б) Автоматическая передача данных в UART.

В) Возможность напрямую включать светодиоды.

**18. Какой из перечисленных этапов следует непосредственно после команды запуска игры?**

А) Формирование случайной задержки и переход к ожиданию сигнала.

Б) Немедленный вывод итоговой оценки.

В) Завершение работы программы.

**19. Для чего в игре используется терминальная программа на компьютере?**

А) Для ввода команд и отображения текстовых результатов.

Б) Для аппаратного измерения задержки.

В) Для замены функций GPIO.

**20. Что является основным результатом выполнения лабораторной работы?**

А) Программа, измеряющая время реакции пользователя с использованием UART, TRNG и светодиодов.

Б) Только настройка UART без прикладной логики.

В) Только генерация случайных чисел без интерфейса пользователя.

# Заключение

Выполнение представленного цикла практических работ позволяет сформировать у обучающегося базовый комплекс компетенций в области низкоуровневого программирования и схемотехники встраиваемых систем на базе отечественной компонентной базы.

Микроконтроллер К1921ВГ015 (НИИЭТ), построенный на архитектуре RISC-V, продемонстрировал высокий потенциал для реализации задач управления реального времени.

Практикум охватывает полный цикл разработки: от настройки выводов GPIO и работы с таймерами до построения отказоустойчивых систем с контролем энергопотребления. Каждая практическая работа закрепляет теоретический материал и формирует навыки, востребованные при проектировании промышленной электроники. Последовательное освоение аналоговой периферии, коммуникационных интерфейсов и механизмов энергосбережения позволяет читателю самостоятельно разрабатывать устройства для портативной медицины, телеметрии и смарт-счетчиков. Полученные компетенции напрямую конвертируются в практические решения на отечественной элементной базе, что особенно важно в условиях импортозамещения.

Автор надеется, что данный практикум станет надежной отправной точкой для дальнейшего профессионального роста читателя.

# Список рекомендуемой литературы

Для самостоятельной верификации инженерных решений и углубленного изучения платформы рекомендуется опираться исключительно на актуальную эксплуатационно-техническую документацию производителя АО «НИИЭТ»:


1. **К1921ВГ015.** Руководство пользователя. — Воронеж: АО «НИИЭТ», 2024. Документ содержит полное описание карты регистров, функциональных узлов и временных диаграмм работы периферии.
2. **К1921ВГ015.** Спецификация (Datasheet). — Воронеж: АО «НИИЭТ», 2024. Содержит предельные электрические параметры, характеристики нагрузочной способности портов и температурные режимы эксплуатации.
3. **К1921ВГ015.** Справочный листок (Errata). Официальный перечень известных ограничений кремния и отклонений от спецификации. Данный документ подлежит обязательному изучению перед началом трассировки печатных плат.
4. **NIIET K1921VG015 SDK.** Репозиторий исходного кода / GitFlic. URL: [https://gitflic.ru/project/niiet/k1921vg015_sdk].

**Приложение**

Ссылки на архивы примеров работ расположены https://disk.360.yandex.ru/d/9VTwAPaPzeHM4g

