Статус документа: ЧЕРНОВИК

Содержит неизбежные для ранней версии:

- *Опечатки и стилистические погрешности*
- *Фактические неточности*
- *Неполные описания*

Текст находится в процессе доработки.

О чём этот документ?

Это описание **продуманной концепции** экосистемы vm5277. Мы находимся на стадии альфа-реализации, где ядро для AVR уже работает, а поддержка PIC/STM8 и другие компоненты — **архитектурно проработаны и ожидают своей реализации.**

Оглавление для VM5277

- Исходный код компилятора и кодогенераторов (/common)
- Примеры проектов (/examples)
- Исходный код RTOS (/rtos)
- Стандартная библиотека (/runtime)

6. Работа с компилятором

- Основной сценарий использования
- Синтаксис командной строки и параметры
- Описание этапов компиляции
- Форматы входных и выходных файлов

7. Внесение вклада в развитие проекта

- Как сообщить об ошибке (Bug Report)
- Обсуждение проекта (Telegram, Email)
- Путь для разработчиков: сборка из исходных кодов

8. Лицензирование и правовые аспекты

- Лицензия проекта
- Использование сторонних товарных знаков (Java, AVR, PIC, STM8)
- Авторские права

1. Введение в проект VM5277

📌 Статус проекта: Альфа-версия

Данная документация описывает **полноценную и продуманную архитектуру** экосистемы vm5277. На текущий момент реализован и находится в стадии доработки компилятор, ассемблер и RTOS для платформы **AVR**.

Поддержка других платформ (**PIC, STM8**) и многие расширенные компоненты являются **запланированными к реализации**, но их архитектура уже проработана и не содержит концептуальных противоречий.

Данная документация описывает не только текущее состояние, но и архитектурный замысел и дорожную карту развития embedded-инструментария

Вероятно, в ходе развития проекта некоторые архитектурные особенности могут быть незначительно скорректированы для повышения эффективности и удобства.

1. Что такое VM5277?

VM5277 — это универсальный мост между мирами embedded-разработки и прикладной разработки ПО с ООП парадигмой.

Независимо от того, с какой стороны вы пришли в embedded-разработку, VM5277 предлагает унифицированный подход...

Если вы пришли из мира Arduino:
Вы знаете, как просто начать с digitalWrite() и millis(), но столкнулись с ограничениями:

- "Раздутый" код, который не помещается в память
- Привязка к конкретным платформам, архитектурам и платформозависимым библиотекам МК
- Сложности с переносом проекта между разными МК
- Сложность восприятия кода, перенасыщенного заголовочными файлами, дефайнами, указателями, двойными указателями, сложной системой типов и операций с памятью.

Если Вы Java-разработчик:
Вы цените чистый ООП-код, но при взгляде на embedded-мир видите:

- Отсутствие мультиплатформенности в реалиях Java разработки
- Отсутствие единой нормальной стандартной библиотеки, и огромное количество плохо согласованных библиотек
- Аналогично — сложность восприятия кода

Если вы опытный embedded разработчик:

Вероятно Вы согласитесь, что большая часть кода — это бизнес-логика, остальное — критичные ко времени операции. VM5277 позволяет оптимизировать бизнес логику, предоставляя более высокоуровневый, а главное — мультиплатформенный, язык разработки ПО. При этом позволяет использовать нативные вызовы ассемблер функций через объявление нативных методов.

Разделяйте и властвуйте:

- **Верхний уровень на J8B** — скорость разработки и надежность
- **Критичные участки на ассемблере** — максимальная производительность и ответственность

Опытные embedded-разработчики получают:

- **Сокращение времени разработки** бизнес-логики благодаря высокоуровневой абстракции
- **Снижение количества ошибок** благодаря строгой типизации J8B и другим фичам
- **Упрощение поддержки** через читаемую бизнес-логику
- **Сохраняют полный контроль** над критичными участками через нативные методы

С vm5277 вся или большая часть вашего кода будет выглядеть вот так:

```
import rtos.System;
import rtos.RTOSParam;
import rtos.Thread;
import hal.GPIO;

interface ISensor {
	fixed read();
}

class TemperatureSensor implements ISensor {
	private byte sensorPin;
```

```
	public TemperatureSensor(byte pin) {
		this.sensorPin = pin;
	}

	private native short readAnalog();

	public fixed read() {
		short raw = readAnalog();
		return (fixed)raw * 0.1 - 50.0;
	}
}

class TemperatureMonitor {
	private ISensor sensor;

	public TemperatureMonitor(ISensor sensor) {
		this.sensor = sensor;
	}

	public void run() {
		while(true) {
			fixed temp = sensor.read();

			System.outCStr("Temperature: ");
			System.out(temp);
			System.outCStr(" C\n");

			if(temp>30.0) {
				System.outCStr("*** OVERHEAT ALERT ***\n");
			}

			Thread.waitS(3);
		}
	}
}

class Main {
	public static void main() {
		System.setParam(RTOSParam.CORE_FREQ, 16);
		System.setParam(RTOSParam.STDOUT_PORT, GPIO.PB2);

		ISensor sensor = new TemperatureSensor(GPIO.PB7);

		TemperatureMonitor monitor = new TemperatureMonitor(sensor);
		monitor.run();
	}
}
```

Техническая суть: VM5277 — это **компилятор-транслятор**, который превращает ваш Java-подобный код J8B (создан специально для 8-битных микроконтроллеров) в оптимизированный ассемблер для AVR, PIC или STM8. Никаких виртуальных машин для слабого железа, никаких интерпретаторов — только нативная производительность.

Ключевая магия:

- **Единый код** для AVR/PIC/STM8 через унифицированный HAL

- **Нативная компиляция** — производительность, сравнимая с ручным ассемблером
- **Современный ООП-синтаксис,** основанный на композиции и наследовании интерфейсов, который исключает сложности C/C++ (такие как множественное наследование, шаблоны) и навязчивое использование указателей.
- **RTOS из коробки** — многозадачность без костылей в виде знакомого Threads
- **Кросс-платформенные инструменты** — работают на Windows/Linux/macOS

VM5277 — это инструментарий, который значительно снижает трудозатраты в целом. А также снижает порог вхождения в embedded-разработку, делая её более доступной, предсказуемой и эффективной.

Конечно многие заявленные преимущества возможны только при накопленном, богатом и хорошо оттестированным временем кодом, что не может дать новый проект. Однако, даже сейчас вы можете использовать это как инструмент генерирующий не плохой ассемблерный код из простого и понятного кода высокоуровневого языка.

Более того, этот подход дает возможность с меньшими трудозатратами, своими силами реализовать некоторый функционал (например драйвер LCD экрана) чем портирование чужого решения между различными платформами.

2. Важное замечание: Статус "Альфа-версия"

Проект VM5277 находится на стадии **активной альфа-разработки,** это означает:

- **Реализован базовый функционал компилятора** для языка J8B с Java-подобным синтаксисом
- **Работает кодогенерация** в оптимизированный AVR ассемблер для ATmega328
- **Реализован AVR ассемблер** — более гибкий и более функциональный чем, к примеру, Avra
- **Реализована минимальная, базовая архитектура RTOS** с системными вызовами
- **Есть основные языковые конструкции**: классы, методы, циклы, условия, массивы, enum, ООП, и т.д.
- **Добавлены ключевые оптимизации**: свёртка констант, упрощение выражений и т.п.

Однако проекту далеко для production-использования:

- **Всестороннее тестирование отсутствует**— код проверяется в основном визуальной инспекцией сгенерированного ассемблера
- **Текущая реализация сфокусирована на AVR (ATmega328)** — хотя архитектура поддерживает мультиплатформенность, реализации для PIC и STM8 запланированы на будущее
- **Не реализованы некоторые языковые конструкции**: switch-case, исключения
- **Оптимизации реализованы частично** (~50% от запланированного)
- **Сложные функции RTOS не протестированы** на реальном железе

Текущие технические ограничения:

- Кодогенерация (backend) только под AVR ATmega328
- Нет поддержки PIC, STM8 и других архитектур
- Отсутствует система исключений (try-catch)

- Не реализована конструкция switch-case
- Ограниченная диагностика ошибок

Риски при использовании:

- Проект поставляется "как есть" (AS-IS)
- Возможны скрытые ошибки frontend и backend
- Нет гарантий корректности работы RTOS
- Отсутствует документация по многим компонентам (начата реализация)
- **Проект разрабатывается одним человеком** — все зависит от моего времени и возможностей

Однако важно отметить:

- Проекту уже **7 месяцев** - пройден огромный путь от идеи до написанного с нуля работающего компилятора
- Это **мой первый компилятор** - за это время преодолены многочисленные технические сложности
- Я продолжаю активную разработку **прямо сейчас**
- Риск прекращения разработки **крайне низок** — просто исходя из объема проделанной работы и потраченного времени

Честно о человеческом факторе: Да, я один. Но именно это дает проекту преимущество - целостное видение, непротиворечивая архитектура и полное понимание всех компонентов системы. Ваша поддержка и обратная связь помогут ускорить развитие и сделать проект по-настоящему значимым.

План развития (Roadmap) - является ориентировочным и может корректироваться

Ближайшие цели (Q4 2025 - Q1 2026):

- Реализация системы исключений (try, catch, throw)
- Добавление конструкции switch-case
- Завершение реализации недостающих языковых конструкций в целом
- Расширение runtime проверок безопасности (после try-catch)
- Возможно - разработка планировщика задач для AVR
- Создание некоторых базовых драйверов HAL
- Начало разработки документации для разработчиков
- Создание демонстрационных примеров и сравнительных тестов производительности
- Пробная реализация поддержки МК серии ATTiny

Среднесрочные цели (Q2 2026):

- Реализация базового функционала многозадачности (близко к Java Threads)
- Добавление поддержки PIC и STM8 архитектур
- Создание унифицированных кросс-платформенных API
- Улучшение системы оптимизаций

Долгосрочное видение:

- Доработка функционала многозадачности
- Реализация отладчика верхнего уровня
- Реализация библиотек верхнего уровня и драйверов (HAL, RTOS)
- Разработка LSP-сервера для IDE поддержки
- Поддержка 32-битных микроконтроллеров (native)
- Реализация легковесной JVM для мощных устройств
- Создание полноценной экосистемы с отладкой и профилированием

Для разработчиков, желающих экспериментировать с проектом:

- Исходный код доступен в репозитории GitHub
- Примеры проектов находятся в директории /examples/j8b/
- Актуальные новости и изменения в NEWS.md

Проект реализуется одним разработчиком, скорость и качество реализации сильно зависит от участия сообщества.

На текущем этапе мне бы очень помогли Unit тесты семантики, или просто ваша обратная связь по работе frontend компилятора.

3. Системные требования

Поддерживаемые операционные системы

- **Windows** 7 и новее (вероятна работа на более старых с JRE8)
- **Linux** - большинство современных дистрибутивов
- **macOS** - см. ограничения Java 8+

Необходимое программное обеспечение

- *Готовый дистрибутив (exe)***: Собран в Native Image (GraalVM), не требует JRE.**
- *Версия для разработчиков (JAR)*: Требует JRE 8+.

Аппаратные требования

- **Процессор**: Любой современный (примеры собираются за десятки миллисекунд)
- **Память**: 512 МБ ОЗУ (минимально), 1+ ГБ (рекомендуется)
- **Дисковое пространство**: 100 МБ

Технические особенности проекта:

- **Полностью написан на Java** - без сторонних зависимостей
- **Сборка через Maven** - простая установка и обновление
- **Мгновенная компиляция J8B** - примеры собираются за 0.02 секунды (native)
- **Компактные исполняемые файлы** при использовании GraalVM Native Image

- **Консольное приложение** - не требует графического интерфейса
- **Два варианта использования**: готовый exe (не требует JRE) или JAR (требует JRE 8+)

Фактически, vm5277 не предъявляет высоких требований к железу и будет работать на большинстве современных компьютеров, поддерживающих JRE 8+

4. Установка и настройка

Загрузка и установка

Загрузите архив с последней версией проекта с GitHub:
https://github.com/w5277c/vm5277/archive/refs/heads/main.zip

Распакуйте архив:

Windows:
Распакуйте содержимое архива в корень диска C:\. Переименуйте полученную папку vm5277-main в vm5277. Итоговый путь должен быть: C:\vm5277\.

GNU Linux и macOS:
Распакуйте архив в ваш домашний каталог. Переименуйте папку vm5277-main в vm5277. Итоговый путь должен быть: ~/vm5277/.

Настройте переменные окружения:
Эта настройка добавляет пути к исполняемым файлам компилятора и утилит в вашу систему, позволяя запускать их из любого места.

Windows:
Запустите файл path_export.cmd из каталога C:\vm5277\scripts\. Этот скрипт **настроит переменные окружения постоянно** для всех будущих сессий командной строки.

Важно: После выполнения скрипта закройте текущее окно командной строки и откройте новое, чтобы изменения вступили в силу.

GNU Linux и macOS:
Выполните скрипт path_export.sh, а затем обновите текущую оболочку, выполнив source для файла ~/.bashrc

```
cd ~/vm5277
./scripts/path_export.sh
source ~/.bashrc
```

Проверка установки

Убедитесь, что компилятор j8bc доступен в системе. Выполните в командной строке:

```
j8bc --version
```

В результате должна отобразиться версия компилятора, например:

```
j8b compiler version: 0.3.0
```

Тестовая компиляция

Перейдите в корневую директорию проекта (переменная окружения $vm5277 должна указывать на нее после выполнения скриптов).

cd $vm5277

Выполните компиляцию тестового примера helloworld.j8b для микроконтроллера AVR ATmega328P:

j8bc avr:atmega328p examples/j8b/helloworld.j8b

Ожидаемый результат

В случае успешной компиляции вы увидите вывод, аналогичный приведенному ниже. Ключевые индикаторы успеха — сообщение Build SUCCESS и наличие сгенерированных файлов (.asm, .hex) в папке examples/j8b/target/.

```
Parsing /home/kostas/vm5277/examples/j8b/helloworld.j8b ...
Parsing done, time:0.003 s
Semantic...
Semantic done, time:0.001 s
Codegen...
Codegen done, time:0.002 s
Assembling...

--CODE-----------------------------------
 Start  = 0000, End = 020A, Length = 020B
 -----
 Total  :  523 words (1046 bytes) 3%

parsed: 1939 lines
Build SUCCESS, warnings:0
Assembling done, time:0.014 s
Total time:0.021 s
```

5. Обзор структуры проекта

Корневая директория vm5277 содержит следующую структуру, которая отражает основные компоненты платформы: компилятор, среду выполнения, операционную систему реального времени, документацию и примеры.

```
├── bin/                  # Исполняемые файлы и скрипты
│   ├── j8bc              # Основной компилятор
│   ├── jre/              # Среда выполнения (JRE для варианта запуска в JAR формате)
│   └── libs/             # Библиотеки компилятора (jar-файлы)
│
├── common/               # Исходный код компилятора основных компонентов (Maven-проекты)
│   ├── compiler/         # Ядро компилятора - frontend (лексер, парсер, семантический анализ)
│   ├── avr_codegen/      # Кодогенератор для архитектуры AVR - backend
│   ├── stub_codegen/     # Кодогенератор -заглушка для Unit тестов компилятора
│   ├── common_lib/       # Общая библиотека для компонентов компилятора
│   └── avr_assembler/    # Ассемблер для AVR (преобразует asm в hex)
│
├── runtime/              # Библиотеки языка программирования J8B
│   ├── hal/              # Аппаратные абстракции (HAL)
│   ├── lang/             # Базовые классы (Byte, Number, Runtime, ...)
│   └── rtos/             # API для работы с RTOS
│
├── rtos/                 # Реализация RTOS для различных архитектур
│   ├── avr/              # RTOS для микроконтроллеров AVR
│   │   ├── conv/         # Конвертация разных типов данных
│   │   ├── core/         # Ядро ОС
│   │   ├── devices/      # Расширенные определения для МК
│   │   ├── dmem/         # Менеджеры динамической памяти
│   │   ├── io/           # Функции ввода/вывода(GPIO)
```

```
│   │   ├── j8b/           # Функции для J8B компилятора
│   │   ├── math/          # Математические процедуры (деление, умножение, ...)
│   │   ├── stdio/         # Функции стандартного ввода-вывода (UART, вывод чисел, …)
│   │   ├── sys/           # Системные функции RTOS
│   │   └── drivers/       # Драйверы различных устройств
│   ├── pic/               # RTOS для PIC
│   └── stm8/              # RTOS для STM8
│
├── defs/                  # Аппаратные определения
│   └── avr/               # Файлы описания для AVR (инструкции, регистры)
│       ├── atmega328.inc  # Определения для ATmega328
│       └── *.instr        # Наборы инструкций процессора
│
├── examples/              # Примеры программ
│   ├── j8b/               # Примеры на языке J8B
│   │   ├── helloworld.j8b # Простой пример "Hello World"
│   │   ├── gpio.j8b       # Работа с GPIO
│   │   └── target/        # Сгенерированные файлы (asm, hex, lst, ...)
│   └── avr_asm/           # Примеры на чистом ассемблере AVR
│
├── docs/                  # Документация
│
└── scripts/               # Вспомогательные скрипты
    ├── path_export.sh     # Настройка переменных окружения (Linux/macOS)
    └── path_export.cmd    # Настройка переменных окружения (Windows)
```

Ключевые компоненты:

Компилятор - Написан на Java и организован как набор библиотек:

- **compiler** - Ядро с лексическим, синтаксическим и семантическим анализом
- **avr_codegen** - Генератор оптимизированного ассемблерного кода для AVR
- **avr_assembler** - Финальный этап компиляции: преобразование asm в hex-прошивку

Среда выполнения (runtime/) - Библиотеки классов, предоставляющие:

- Базовые типы данных (Byte, Number)
- Аппаратные абстракции (HAL.GPIO)
- API для работы с RTOS (Thread, System)

RTOS (rtos/) - Реализация операционной системы реального времени:

- Написана на оптимизированном ассемблере для каждой архитектуры
- Включает планировщик задач, управление памятью, таймеры, ...
- Предоставляет драйверы для периферии

Инструменты (bin/) - Готовые к использованию утилиты:

- j8bc - основной компилятор из исходного кода в прошивку
- libs/*.jar - библиотеки кодогенерации и ассемблирования

Такой подход обеспечивает модульность платформы - компилятор, RTOS и библиотеки времени выполнения развиваются независимо, но работают согласованно для создания эффективных прошивок для 8-битных микроконтроллеров.

6. Работа с компилятором

Основной сценарий использования

На текущем этапе разработки работа с компилятором осуществляется через командную строку. Типичный workflow выглядит следующим образом:

Текущий процесс разработки:

- **Создание исходного кода**
  - Разработчик создает файлы с расширением .j8b в любом текстовом редакторе
  - Пример: my_project.j8b
- **Компиляция через командную строку**

  j8bc avr:atmega328p my_project.j8b

- **Анализ результатов**
  - Компилятор генерирует отчет о процессе сборки
  - В папке target/ создаются выходные файлы:
    - .asm - ассемблерный код для анализа
    - .hex - прошивка для загрузки в МК
    - .lst - листинг для отладки
    - .map - карта памяти
- **Прошивка микроконтроллера**
  - Использование внешних инструментов (AVRDUDE, STM8Flash и др.)

Пример полного цикла:

```
# Создание и редактирование кода в текстовом редакторе
nano helloworld.j8b
# Компиляция проекта
j8bc avr:atmega328p helloworld.j8b
# Если компиляция успешна - прошивка МК
avrdude -p atmega328p -c arduino -P /dev/ttyUSB0 -b 115200 -U flash:w:helloworld_cseg.hex
```

Перспективы улучшения workflow:

С реализацией **LSP-сервера** (запланировано в roadmap) процесс разработки станет значительно удобнее:

- **Подсветка синтаксиса** в популярных IDE (VS Code, IntelliJ, Kate)
- **Автодополнение** кода и навигация по проекту
- **Встроенная проверка ошибок** в реальном времени
- **Интегрированная компиляция** без переключения в терминал
- **Отладка** с помощью универсального бутлоадера

Несмотря на текущий минимализм интерфейса, компилятор уже обеспечивает полный цикл преобразования высокоуровневого ООП-кода в оптимизированную прошивку для 8-битных микроконтроллеров.

Синтаксис командной строки и параметры

Базовый формат команды

```
j8bc <platform>:<mcu> <input.j8b> [options]
```

Обязательные параметры

- **<platform>:<mcu>** - целевая платформа и микроконтроллер
  - Примеры: avr:atmega328p, stm8:stm8s003f3, pic:16f877a
- **<input.j8b>** - исходный файл на языке J8B

Основные опции компиляции

Управление выводом

- **-o, --output <file>** - указание выходного HEX-файла (по умолчанию: <input>.hex)
- **--dump-ir** - вывод промежуточного представления после фронтенд-обработки
- **-am, --asm-map** - генерация файла карты памяти ассемблера
- **-al, --asm-list** - генерация листинга ассемблера

Оптимизация

- **-p, --opt <level>** - уровень оптимизации высокоуровневого кода:
  - none - без оптимизации
  - front - только фронтенд-оптимизация
  - size - оптимизация по размеру (по умолчанию)
  - speed - оптимизация по скорости выполнения (не реализовано)

Настройки строгости

- **-s, --strict <level>** - уровень обработки неоднозначностей:
  - strong - трактовать как ошибку
  - light - показывать предупреждение (по умолчанию)
  - none - тихий режим

Аппаратные параметры

- **-F, --freq <MHz>** - тактовая частота МК в МГц (также может быть задана в исходном коде)
- **-P, --path <dir>** - пользовательский путь к директории тулкита

Отладка и информация

- **--cg-verbose 0-2** - генерация детальной информации о кодогенерации:
  - 0 - минимальный вывод
  - 1 - обычная детализация
  - 2 - максимальная детализация (пока не используется)

- **-v, --version** - отображение версии компилятора
- **-h, --help** - вывод справки

Примеры использования

Базовая компиляция

j8bc avr:atmega328p main.j8b

Компиляция с оптимизацией по скорости

j8bc avr:atmega328p main.j8b -p speed -F 16

Компиляция с отладочной информацией

j8bc avr:atmega328p main.j8b --dump-ir -am -al --cg-verbose 2

Компиляция со строгими проверками

j8bc avr:atmega328p main.j8b -s strong -p none

Использование кастомного пути к тулките

j8bc avr:atmega328p main.j8b -P /custom/path/to/vm5277

Форматы выходных файлов

При успешной компиляции в папке `target/` генерируются:

- **`.asm`** - ассемблерный код для анализа
- **`.hex`** - прошивка в формате Intel HEX
- **`.lst`** - листинг для отладки (с `-al`)
- **`.map`** - карта памяти (с `-am`)
- **`.j8bir`** - промежуточное представление (с `--dump-ir`)

Описание этапов компиляции

Компилятор J8BC выполняет полный цикл преобразования исходного кода в готовую прошивку, включая встроенный вызов ассемблера. Процесс состоит из следующих этапов:

Инициализация и предупреждения

```
Version: 0.3.0 | License: Apache-2.0
============================================================
WARNING: This project is under active development...
============================================================
```

- Отображение версии компилятора и лицензионной информации
- Вывод предупреждений о статусе разработки

- Рекомендация проверять сгенерированный ассемблерный код

Парсинг (Frontend)

```
Parsing /media/kostas/repos/w5277c/vm5277_local/examples/j8b/enum.j8b ...
Parsing done, time:0.045 s
```

- **Лексический анализ** - разбиение исходного кода на токены
- **Синтаксический анализ** - построение абстрактного синтаксического дерева (AST)
- Проверка базовой синтаксической корректности
- Время выполнения: ~0.045 секунд в примере (JRE режим)

Семантический анализ

```
Semantic...
Semantic done, time:0.012 s
```

- Проверка типов данных и их совместимости
- Валидация областей видимости переменных и методов
- Проверка корректности вызовов методов и доступа к полям классов
- Поиск необъявленных переменных и методов
- Время выполнения: ~0.012 секунд (JRE режим)

Генерация кода (Backend)

```
Codegen...
Codegen done, time:0.015 s
```

- Преобразование AST в промежуточное представление (IR)
- Применение выбранных оптимизаций (-o size/speed/none/front)
- Генерация платформо-специфичного ассемблерного кода для целевого MK
- Распределение регистров и управление памятью
- Время выполнения: ~0.015 секунд (JRE режим)
-

Ассемблирование (встроенное)

```
Assembling...
INF|29:13 /media/.../core.asm ######## LOGGING ENABLED
INF|30:13 /media/.../core.asm ######## IO BAUDRATE:
```

- **Встроенный вызов ассемблера** - преобразование asm в машинный код
- Подключение файлов RTOS и системных библиотек
- Разрешение внешних ссылок и адресов
- Генерация отладочной информации (при включенных флагах)

Анализ результатов сборки

```
--CODE-----------------------------------
```

```
 Start = 0000, End = 0259, Length = 025A
 -----
 Total : 602 words (1204 bytes) 3%
parsed: 2161 lines
Build SUCCESS, warnings:0
Assembling done, time:0.117 s
Total time:0.227 s
```

- Отчет об использовании памяти (адреса, размер кода)
- Статистика по строкам ассемблерного кода
- Итоговый статус сборки (BUILD SUCCESS/FAILURE)
- Количество предупреждений и ошибок
- Время ассемблирования: ~0.117 секунд (JRE режим)
- Общее время компиляции ~0.227 секунд (JRE режим)

Ключевые особенности процесса:

- **Сквозная компиляция** - от исходного кода до готового HEX-файла в одной команде
- **Интегрированный ассемблер** - не требует внешних инструментов для сборки
- **Детальная временная статистика** - позволяет оптимизировать процесс разработки
- **Автоматическое подключение Runtime и RTOS** — компилятору известны пути необходимых ресурсов

Весь процесс полностью автоматизирован и не требует ручного вызова ассемблера или линкера.

Форматы входных и выходных файлов

Входные форматы

Исходный код (.j8b)

```
/*
 * Copyright 2025 konstantin@5277.ru
 */
import rtos.System;
import rtos.RTOSParam;
import hal.GPIO;

class Main {
    public static void main() {
        System.setParam(RTOSParam.CORE_FREQ, 16);
        System.setParam(RTOSParam.STDOUT_PORT, GPIO.PB2);

        cstr text = "Hello world!\r\nПривет!\r\n";
        System.outCStr(text);
        System.stop();
    }
}
```

Особенности:

- Java-подобный синтаксис с упрощениями
- Поддержка кириллицы в строковых литералах
- Импорт системных библиотек (rtos, hal)
- Статический метод main() как точка входа

Выходные форматы

Промежуточное представление (.j8bir)

```
class Main {
    static public void main() {
       System.setParam(0, 16);
        System.setParam(2, 18);
        System.setParam(3, 1);
        final cstr text = "Hello world!\r\nПривет!\r\n";
        System.outCStr(text);
        System.stop();
    }
}
```

Назначение:

- Упрощенное AST после семантического анализа
- Константы подставлены вместо именованных параметров
- Используется для отладки фронтенда компилятора

Ассемблерный код (.asm)

```
; vm5277.avr_codegen v0.1 at Fri Nov 14 05:34:40 VLAT 2025
.equ core_freq = 16
.equ stdout_port = 18

.include "devices/atmega328p.def"
.include "core/core.asm"

j8bD81:
.db "Hello world!",0x0d,0x0a,0xf0,0xd2,0xc9,0xd7,0xc5,0xd4,"!",0x0d,0x0a,0x00

Main:
    rjmp j8bCMainMmain

j8bCMainMmain:
    ldi r16,low(j8bD81*2)
    ldi r17,high(j8bD81*2)
    movw r30,r16
    rcall os_out_cstr
    rcall mcu_stop
    rjmp j8bproc_mfin
```

Особенности:

- Платформо-специфичный ассемблер AVR
- Автоматическое подключение RTOS библиотек
- Строки хранятся в программной памяти (.db)
- Оптимизированный вызов системных функций

Листинг (.lst)

```
.EQU core_freq = 16
.EQU stdout_port = 18
.INCLUDE /path/to/atmega328p.def
.INCLUDE /path/to/common.inc

.ORG 0
0x0000 jmp _os_init
ports_table: 0x0002
.DB 0,35,38,41

; ... (полный листинг с адресами)
```

Назначение:

- Полный листинг с resolved адресами
- Полезен для отладки и анализа памяти
- Содержит все подключенные библиотеки

Карта памяти (.map)

```
# ==== Register aliases ====

R    R7    _core_reg_h
R    R6    _core_reg_l

# ==== Constants ====
C    _os_stat_pool = 256
C    _os_timer_freq = 100

# ==== Variables ====
V    os_dram_1b = 308
V    os_dram_2b = 436

# ==== Labels ====
L    j8bcmainmmain = 517
L    main = 515
```

Назначение:

- Распределение регистров и памяти
- Адреса меток и функций
- Размеры областей памяти

Прошивка (.hex)

```
:100000000C9476000023262900 6B6C6D0102099385
:100010000197E9F70895EF93FF938F93E5E0F0E000
```

```
:10002000E4913296FF27808184608083E5E0F0E0F0
:0604100050DEEBDFE1CF3E
:00000001FF
```

Особенности:

- Стандартный Intel HEX формат
- Готов для загрузки в МК через программатор
- Включает checksum для проверки целостности

Назначение файлов в workflow

	Использование
.j8b	Исходный код для разработки
.j8bir	Отладка фронтенда компилятора
.asm	Анализ сгенерированного кода
.lst	Детальная отладка и оптимизация
.map	Анализ использования памяти
.hex	Финальная прошивка для МК

Все форматы интегрированы в единый процесс компиляции и предоставляют полную информацию для разработки и отладки embedded-приложений.

7. Внесение вклада в развитие проекта

Как сообщить об ошибке (Bug Report)

На текущем этапе проект находится в активной разработке, и формальная система отслеживания ошибок не используется. Но можно воспользоваться баг-трекером GitHub.

Если вы обнаружили проблему или неожиданное поведение компилятора:

- Напишите напрямую разработчику: konstantin@5277.ru
- Опишите проблему на столько подробно на сколько сможете/посчитаете необходимым

Примечание: По мере появления заинтересованных участников будет рассмотрен вопрос выбора общего баг-трекера, и другого инструментария.

Обсуждение проекта

Присоединяйтесь к обсуждению развития платформы:

- **Telegram группа**: https://t.me/vm5277corecult - для общих вопросов и обсуждения возможностей
- **Личная почта**: konstantin@5277.ru - для персональных вопросов

Путь для разработчиков: сборка из исходных кодов

Исходные коды всех компонентов компилятора доступны в репозитории проекта. Для сборки используется стандартная Maven-инфраструктура.

Текущий статус документации для разработчиков:

- Документация по внутренней архитектуре запланирована на будущее
- Исходные коды содержат базовые комментарии для понимания структуры

8. Лицензирование и правовые аспекты

Лицензия проекта

Проект vm5277 распространяется под лицензией **Apache 2.0**, что означает:

- ✅ Свободное использование, модификация и распространение
- ✅ Возможность использования в коммерческих продуктах
- ✅ Предоставление "как есть" (AS-IS) без гарантий
- ✅ Обязательное указание авторства и изменений

!Двойное лицензирование и будущее развитие!

Важно: открытая кодобаза остаётся под лицензией Apache 2.0 навсегда и продолжит публичное развитие. Это ядро проекта всегда будет доступно сообществу.

Параллельно автор оставляет за собой право предлагать коммерческие лицензии для расширенного функционала, включая:

- Продвинутые оптимизации
- Специализированные драйверы
- Дополнительные платформы
- Расширенные инструменты разработки
- Прочее

Участие в проекте подразумевает согласие с тем, что ваш вклад в основную кодобазу (под Apache 2.0) может быть использован как основа для будущих коммерческих предложений, которые будут развиваться как отдельные продукты.

Использование сторонних товарных знаков

Проект уважает права третьих лиц и использует следующие обозначения в соответствии с принципом добросовестного использования:

- **Java®** - зарегистрированный товарный знак Oracle Corporation
- **AVR®, ATmega®** - товарные знаки Microchip Technology Inc.
- **PIC®** - товарный знак Microchip Technology Inc.
- **STM8®** - товарный знак STMicroelectronics

VM5277 не связан с указанными компаниями и не является официальной реализацией их технологий. Упоминания осуществляются исключительно для описания совместимости и характеристик.

Авторские права