# Введение

Компилируемые языки, такие как C или C++, применяются во многих областях , например для ускорения вычислений, создания системных компонент и утилит, написания прошивок для embedded устройств, а также разработки рантаймов других языков, например .NET и Java. ...

Компилируемые языки типа C/C++ активно применяются для <<быстрых>> вычислений, а также для написания рантаймов других языков, например .NET и Java. В силу особенностей с ними стоит использовать системы сборки, такие как Make, CMake, Autotools, Meson, QMake и др.

Архитектура RISC-V привлекает всё большее внимание как исследователей, так и разработчиков ПО. Тем не менее, выбор устройств на RISC-V пока что ограничен, доступные IP ядра отстают от спецификаций , а производительности эмуляторов, таких как QEMU, недостаточно для повседневных задач.

Перед разработчиками ПО встает задача сборки и тестирования своего ПО на платформе RISC-V. Для этого необходим CI, поддержка системы сборки...

Целью данной работы является систематизация знаний о способах сборки проекта для архитектуры RISC-V, а также настройки CI.

# Постановка задачи

Целью данной работы является изучение возможностей по созданию архитектуры для сборки проектов с открытым исходным кодом для платформы RISC-V.

Для достижения данной цели были поставлены следующие задачи:

1. Изучить способы сборки проекта под RISC-V;
2. Исследовать поддержку кросс-компиляции в различных системах сборки;
3. Изучить поддержку self-hosted CI runner;

# Обзор

Всегда будет рассматривать компиляцию только для ОС Linux.

Для начала введём некоторые термины, использующиеся при кросс-компиляции (в соответствии с ):

* Build — система, на которой производится сборка;
* Host — система, под которую производится сборка;
* Target — в случае компиляторов, система, для которой будет генерироваться код.

Для небольших приложений обычно достаточно минимального набора библиотек, предоставляемого компилятором.

В случае же если требуются дополнительные зависимости, то можно либо собрать их самому, либо создать sysroot какого-нибудь дистрибутива.

Первый способ обычно достаточно трудозатратен и обычно не используется. В случае, если зависимость отсутствует в репозиториях, то допустим, что за её компиляцию и установку отвечает система сборки проекта.

Поэтому обычно используется второй способ. Sysroot — минимальный образ системы для host платформы, который включает в себя необходимые зависимости для сборки и работы ПО. Для его создания используются такие утилиты как debootstrap, buildroot, yocto, dnf.

## Виды сборки

### Нативная сборка

Самый обычный вид сборки, когда build = host. В данном случае от разработчика требуется минимум усилий для сборки своего проекта, так как среда внешне не отличается от привычной.

Сам процесс компиляции происходит довольно быстро, а зависимости устанавливаются из системных репозиториев. Тем не менее у данного подхода есть и недостатки, например необходимость покупки устройств на платформе RISC-V.

### Кросс-компиляция

Способ компиляции, при котором build host. Требует использования компилятора, target которого есть ожидаемый host.

Для этого системе сборки необходимо сообщить об используемом компиляторе, а также, в случае необходимости, о пути к sysroot с зависимостями для сборки.

Основная проблема данного метода: требуется поддержка как со стороны системы сборки, так и проекта. О поддержке кросс-компиляции различными системами сборки более подробно рассказано в [2.2](#subsec:build_systems). Некоторые проекты, например OCaml, не поддерживают кросс-компиляцию вовсе.

### Нативная сборка в эмуляторе

В случае если проект не поддерживает кросс-компиляцию и нет возможность получить доступ к оборудованию с целевой платформой, можно использовать эмулятор.

Ядро ОС Linux поддерживает механизм binfmt\_misc. Он распознает заголовок исполняемого файла и в случае необходимости позволяет запускать его через программу-обертку, например Wine или QEMU.

В качестве эмулятора для сборки можно использовать QEMU. Он может работать в двух режимах: System Emulation и User Mode Emulation. В первом случае QEMU работает как виртуальная машина и эмулирует систему целиком, для этого ему требуется образ ОС, которая будет запущена внутри. Для целей сборки такой подход слишком тяжеловесен. Во случае же User Mode Emulation, QEMU исполняет программу путем трансляции машинного кода в машинный код платформы, на который он запущен, а системные вызовы “пробрасываются” наружу. Таким образом можно запустить исполняемый файл для другой архитектуры не запуская полноценную виртуальную машину.

Далее есть два варианта действий:

1. Создать sysroot со всеми необходимыми зависимостями, далее сделать chroot и собрать проект.
2. Воспользоваться Docker контейнером, собранным для архитектуры RISC-V[[1]](#footnote-24)

Данный способ хорош тем, что позволяет собирать проекты не поддерживающий кросс-компиляцию. Однако QEMU работает очень медленно, поэтому сборка большого проекта занимает очень много времени.

## Системы сборки

### Make

Самая простая система сборки, позволяющая “выстрелить себе в ногу”. Если Makefile правильно и аккуратно написан, то проблем скорее всего не будет. Всё нужно сделать в данном случае указать правильные переменные CC и CXX, и при необходимости дописать туда путь к sysroot.

### Autotools

В целом должно быть норм. Флаг –host есть, могут быть проблемы с сисрутом, но куда-нибудь (например в CC или CFLAGS) его уж можно подсунуть.

### CMake

Если не извращаться, то должно легко заработать. Но надо проверить на MROB.

### QMake

Вот здесь ничего хорошего.

QMake (и весь QT) надо собирать ручками для кросса. Сделать сисрут будет просто, а вот чтобы потом при копировании оно завелось — сложно. Если точнее нужно брать флаги в билд скриптах нужного дистрибутива.

Если не хочется этой боли, то можно под QEMU или совсем нативно. В первом случае даже рантайм собирается почти час :).

### Meson

Не самая приятная система. Плохая документация для файлов кросс тулчейнов, хотя жить можно. Ищет зависимости через pkg-config, поэтому для него (а точнее для более новой и адекватной версии — pkfconf) надо сделать .personality файл, в котором правильно указать пути. (А пути все абсолютные, а так же надо делать chroot, чтобы не гадать)

## Поддержка в CI

Такие хостинги для проектов как GitHub и GitLab, пусть и предоставляют собственные сервисы для настройки CI, тем не менее не имеют собственных runner’ов на архитектуре RISC-V.

Но есть возможность сделать self-hosted runner.

### GitHub

Приложение от GitHub написано в основном на .NET. На данный момент .NET поддерживает RISC-V базово. Кроме того, GH использует скрипты для MSBuild для сборки переносимого приложения. Есть draft PR[[2]](#footnote-34) для ppc64-le, но выглядит страшно. Кроме того, даже сейчас, в альфе .NET9, у них нет нормального GC.

### GitLab

Раннер гитлаба написан на Go, и существует собранный под RISC-V, но пока его можно найти только в их артефактах сборки. На LPi4A не проверял, т.к. наша версия имеет мало места на eMMC и докеру будет неприятно, а внешний SSD периодически отваливается. На VisionFive2 потребовалась пересборка ядра, спасибо вики Gentoo, что рассказали что нужно включить в конфиге ядра. Даже их официальная инструкция говорит о том, что требуется пересборка ядра. Кроме того, с гитлабом есть проблема: синхронизация из GitHub в GitLab требует GitLab Premium, наверное можно взять self-hosted, но это большая боль как по мне. Есть ещё Action, который синкает репо ручками, тоже кажется костылём.

### Jenkins

Java, которая умеет в риск, всё супер. В дебиановском репе пакеты под архитектуру All. На VisionFive2 по умолчанию старые репозитории, и он не стартует. Если вписать http://ftp.ru.debian.org/debian/ и установить более новую Java, то всё становится хорошо

# update-alternatives --set iptables /usr/sbin/iptables-legacy  
# update-alternatives --set ip6tables /usr/sbin/ip6tables-legacy

docker-pipelines — плагин чтобы использовать docker

# usermod -aG docker jenkins

И серверу и агенту на одной платке тесно. Надо разносить, но наверное для компании, если у них уже Jenkins, норм.

Надо ещё проверить как поллить ГХ на тему коммитов

# Реализация

# Эксперимент

# Заключение

1. <https://hub.docker.com/r/riscv64/debian> [↑](#footnote-ref-24)
2. <https://github.com/actions/runner/pull/2386> [↑](#footnote-ref-34)