Санкт-Петербургский государственный университет

Математико-механический факультет
Кафедра Системного Программирования
Программная инженерия

Группа 22.15-мм

Поддержка SCR1

Луконенко Никита Игоревич

Отчёт по учебной практике
в форме «Решение»

Научный руководитель:
ст. преподаватель кафедры ИАС К. К. Смирнов

Санкт-Петербург
2023

Оглавление

Введение

RISC-V [1] – это открытая архитектура набора команд (ISA), которая распространяется под лицензией Creative Commons и доступна для использования, модификации и распространения через открытые лицензии. Эта архитектура спроектирована с учетом простоты, модульности и масштабируемости, что делает ее привлекательной для широкого круга применений, начиная с микроконтроллеров и заканчивая высокопроизводительными вычислительными системами. Интерес к RISC-V быстро растет, архитектурой интересуется международная организация RISC-V International [2], представляющая из себя открытое сообщество разработчиков, которые работают над её развитием. В это сообщество входят следующие компании: NVIDIA, Intel, Huawei, Qualcomm, и множество других технологических гигантов со всего мира. В России тоже существует альянс RISC-V [5] , который представляет из себя объединение независимых разработчиков. Целью данного альянса является развитие и распространение архитектуры RISC-V в России. В альянс входят представители следующий компаний: Yadro, Syntacore, и ГК Элемент.

SCR1[1] – это ядро процессора на базе архитектуры RISC-V, с открытым исходным кодом. Оно было создано специально для микроконтроллеров. Проект имеет статус silicon-proven, это означает, что он прошел все этапы разработки и производства, а именно: был спроектирован, прошел тестирование и верификацию, был изготовлен на кремниевой подложке, имеет документацию, и его можно запустить без обращения к дополнительным ресурсам, используя только инструкцию, предоставляемую разработчиками. Проект написан на языке Си и Verilog.
Verilog [4] – это язык описания аппаратуры, который используется для моделирования и конструирования цифровых систем.

Проект SCR1 распространяется под лицензией SHL, что позволяет использовать его как в учебных, так и в коммерческих целях. Повышение его доступности позволит упростить процесс внедрения нового

[1] https://github.com/syntacore/scr1

пользователя в проект, а также дальнейшую разработку.

На момент начала практики в проекте SCR1 отсутствует система непрерывной интеграции как в репозитории, так и непосредственно на устройстве пользователя. Есть 247 форков, в которых потенциально могут находиться исправления багов и предупреждений, а также проект не работает с набором инструментов RISC-V GNU Compiler Toolchain, версия которого содержит библиотеку newlib новее, чем 4.1.0. Решение перечисленных проблем и будет целью данной практики.

1. Постановка задачи

Целью работы является повышение доступности проекта SCR1. Для её выполнения были поставлены следующие задачи:

1. Проверить работоспособность SCR1 на новых версиях компонент тулчейна.
2. Просмотреть все имеющиеся форки проекта, найти исправления багов, и применить их.
3. Реализовать непрерывную интеграцию SCR1.

2. Обзор

Для решения поставленных задач использовались следующие инструменты:

Тулчейн riscv-gnu-toolchain с новыми версиями библиотек. Данный инструмент рекомендуется разработчиками в официальном репозитории проекта[2].

Инструмент crosstool-ng [6]. Выбор данного инструмента обуславливается тем, что ct-ng создан специально для сборки тулчейнов. Он позволяет собрать набор инструментов полностью автоматически. Пользователю предоставляется возможность указать необходимые версии компонент, выбрать архитектуру, под которую собирается набор инструментов, и применить различные патчи к компонентам тулчейна посредством удобного интерфейса. Проект активно обновляется, предлагая пользователям большое количество готовых конфигурационных файлов, которые можно изменить для конкретных задач.

При сборке SCR1 с тулчейном, компонентой которого является библиотека newlib, версия которой младше, чем 4.1.0, появляется ошибка, связанная с функцией atoi. Данная функция зависит от функции strtoll, она в свою очередь зависит от библиотеки ctype, а ctype зависит от локалей, которые можно определить только в ходе выполнения программы. Для этого используется большое количество системных библиотек, которые далеко не всегда могут находиться в системе, работающей с аппаратным обеспечением напрямую. Было определено два возможных пути решения данной проблемы: Первым является применение патча, в котором реализована легковесная версия функции atoi. Данный патч доступен по ссылке в репозитории[3].

Вторым способом решения данной проблемы является использование инструмента crosstool-ng. С его помощью можно автоматически собрать свой тулчейн, который будет включать в себя все необходимые версии библиотек.

[2]https://github.com/riscv-collab/riscv-gnu-toolchain

[3]https://github.com/KirillSmirnov/scr1/tree/elim-atoi

Для реализации непрерывной интеграции было принято решение использовать shell скрипты, которые располагаются в отдельной папке проекта. В дальнейшем это позволяет провести непрерывную интеграцию, независимо от платформы, на которой размещается проект, будь то удаленная машина, или непосредственно устройство пользователя. Если пользователь решит провести непрерывную интеграцию на какой-то удаленной платформе, например github actions, то максимум действий, который ему будет необходимо сделать – это описать конфигурацию для машины, на которой будет происходить непрерывная интеграция, например в github.yml файле.

3. Реализация

У библиотек, которые используются в тулчейне, нет версии с долгосрочной поддержкой. Поэтому есть необходимость проверить работоспособность проекта на новых версиях компонент. В ходе проверки, проблем с компилятором GCC 13.2 и библиотекой GNU Binutils 2.40 не возникло.

Проблему, возникающую при обновлении библиотеки newlib можно решить, используя инструмент crosstool-ng, в котором есть возможность обновить библиотеки и компилятор, а версию newlib оставить на 4.1.0.

В ходе разбора форков SCR1 было выявлено большое количество лабораторных работ, и два репозитория, в которых предлагались исправления различных предупреждений и багов.

В одном[4] из них был найден баг в системе сборки. Из за наличия обратного слэша после разных команд, они склеивались, и в случае возникновения ошибки make не мог её отследить. Автор репозитория добавил двойной амперсанд в местах, где одна команда склеивается с другой, но его решение не исправило проблему. По этой причине было решено использовать альтернативное решение, которое заключалось в том, чтобы убрать лишние слэши, и теперь make может определить, в каких местах появляется ошибка.
Пример реализации wsynder:

```
$(sv_list) && \
cd verilator && \
$(MAKE) -f V$(top_module).mk
```

Пример того, какая реализация была предложена в исправлении:

```
$(sv_list);
cd $(bld_dir)/verilator && \
$(MAKE) -f V$(top_module).mk;
```

[4]https://github.com/wsnyder/scr1/tree/fix_shell_errors

В данном случае команда cd будет отделена от набора параметров для Verilator, и если произойдет ошибка в одной из команд, то make успешно её распознает, и сообщит о ней, прекратив дальнейшую сборку.

Во втором[5] репозитории находятся исправления предупреждений Verilator [3]: MULTIDRIVEN и WIDTHEXPAND. Первое вызвано тем, что один сигнал поступает из нескольких блоков always, каждый с разной тактовой частотой. Игнорирование этого предупреждения не влияет на корректность симуляции, но может её замедлить. Второе предупреждение может быть связано с тем, что при арифметических операциях два операнда имеют разную ширину, оно не влияет на корректность симуляции.

Для упрощения работы пользователя с проектом, необходимо реализовать непрерывную интеграцию как у пользователя на устройстве, так и удаленно. Весь механизм представляет из себя набор shell скриптов, которые позволяют автоматизировать установку необходимых компонент, а также запустить сборку и тестирование проекта. Далее приведу список, из чего конкретно состоит непрерывная интеграция:

- all.sh — запустит полный цикл установки тулчейна, Verilator, запуска сборки, тестов, и всех линтеров;
- all_dev.sh — ключевые отличия данного скрипта от all.sh: пользователь сам должен скачать тулчейн, и добавить путь в переменную окружения. Также Verilator собирается полноценно на устройстве пользователя, а не скачивается уже готовый из интернета;
- init_submodules.sh — инициализирует все подмодули проекта;
- load_toolchain.sh — скачает тулчейн из интернета;
- verilator_install_prod.sh — скачает собранный Verilator из интернета, добавит необходимые пути в переменную окружения;
- verilator_install.sh — склонирует репозиторий Verilator, и соберет его. Рекомендуется использовать только локально, для сборки на

[5]https://github.com/brabect1/scr1

платформе GithubActions есть verilator_install_prod.sh;

- run_make.sh — инициализирует модули, запустит сборку и тестирование проекта;
- run_verilator_lint.sh — проверит все файлы с разрешением .sv линтером Verilator;
- run_lint_shell_scripts.sh — скрипт, который запустит линтер shellcheck для всех shell скриптов.

Особенность реализации заключается в том, что для ускорения сборки на GitHub actions необходимо поделить официальный тулчейн sc-dt на части, и разместить его в открытом доступе. Весь архив с набором инструментов занимает 859 мегабайт на диске, а необходимый для работы проекта архив с тулчейном riscv64-unknown-elf занимает 203 мегабайта, эта разница сильно влияет на скорость непрерывной интеграции. Также сборка Verilator сильно увеличивала время непрерывной интеграции, поэтому было принято решение собрать его отдельно, и разместить на платформе Google Drive, а в ходе сборки проекта скачивать архив, с готовым к использованию инструментом.

В ходе реализации непрерывной интеграции был проведен замер времени на устройстве macbook pro, с процессором M1 и объемом оперативной памяти 16 гигабайт. Для сборки проекта использовался docker контейнер с операционной системой ubuntu 22.04 ARM64. Для работы контейнера было выделено 8 ядер процессора, 8 гигабайт оперативной памяти, 1 гигабайт файла подкачки, и 80 гигабайт памяти. Использование docker контейнера обуславливается тем, что в силу специфичности архитектуры процессора M1, и операционной системы MacOS, сборка на основной операционной системе сильно усложняется. Время, затрачиваемое на непрерывную интеграцию в контейнере, при сборке verilator составляет 14 минут 17 секунд. Если verilator заранее собран, то время сборки сокращается до 2 минут 57 секунд. На платформе github actions время непрерывной интеграции при использовании системы ubuntu 22.04 составляет 4 минуты 59 секунд. Это связано с тем,

что предустановленного инструмента verilator и тулчейна там нет, и их приходится скачивать дополнительно. Далее, для наглядности, в таблицах приведены параметры платформ, на которых проводилась сборка, а также затраченное время при непрерывной интеграции в различных условиях:

Устройство	ОС	ЦПУ	Кол-во ядер	ОЗУ	Память
Macbook Pro	ubuntu 22.04	M1	8	8 Гб	80 Гб

	Нет Verilator (сборка)	Есть Verilator
Macbook Pro M1	14 минут 17 секунд	2 минуты 57 секунд

	Github Actions
Нет Verilator(скачивание)	
	4 минуты 59 секунд
Нет toolchain(скачивание)	

Заключение

Результаты, достигнутые в ходе учебной практики:

1. Проверена работоспособность SCR1 на новых версиях компонент тулчейна:

    - Применен патч, в котором реализована легковесная версия функции atoi. Теперь проект собирается с новыми версиями компонент в тулчейне.
    - Собран тулчейн с использованием crosstool-ng на новых версиях библиотек, за исключением newlib.

2. Просмотрены все имеющиеся форки проекта. Применены исправления из форков к актуальной версии ядра SCR1. В частности были исправлены все предупреждения MULTIDRIVEN, и 48 предупреждений WIDTHEXPAND, а также был исправлен баг в системе сборки.

3. Реализована[6] непрерывная интеграция на платформу GitHub actions, и непосредственно на устройство пользователя.

[6] https://github.com/SurfaceYellowDuck/scr1

Список литературы

[1] Архитектура набора команд RISC-V Расширения: Обзор : Rep. / Китайский научно-исследовательский институт телекоммуникаций ; Executor: IEEE) Tianzheng Li Qian Wei Enfang Cui, (Member : 2023. — URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=10049118. — Дата обращения: 21.12.2023.

[2] RISC-V International community. — Дата обращения: 21.12.2023. URL: https://riscv.org/.

[3] Snyder Wilson. — Документация Veriator, 2023. — Дата обращения: 21.12.2023. URL: https://www.veripool.org/ftp/verilator_doc.pdf.

[4] Стандарт IEEE для Verilog® Язык описания аппаратного обеспечения : Rep. / Институт инженеров электротехники и электроники, Inc. ; Executor: IEEE Computer Society : 2006. — URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1620780. — Дата обращения: 21.12.2023.

[5] Альянс RISC-V. — Дата обращения: 21.12.2023. URL: https://riscv-alliance.ru/#about-tech.

[6] Документация crosstool-ng. — Дата обращения: 21.12.2023. URL: https://crosstool-ng.github.io/docs/.