Сборка ядра Linux версии 5.2.9 для платформ RISC-V

Аннотация

Проектирование встраиваемых систем – динамичная область знаний, которая интегрирует аппаратное и программное обеспечение. Со стороны сообщества Linux сделан колоссальный вклад в развитие мира встраиваемых систем в форме свободно распространяемого исходного кода и поддержки различных процессорных архитектур, включая RISC-V. Для переноса операционной системы исходный код ядра настраивается, кросс-компилируется и устанавливается на целевую систему. Мы использовали плату HiFive Unleashed, оснащенную множеством периферийных компонентов, которые позволяют создавать комплексные системы, где можно запускать разнообразные приложения. В рамках стоящих перед нами задач нас интересует в первую очередь оценка производительности сетевых операций. Эта статья дает общее представление о технике переноса и разработке приложений для встраиваемых сетевых систем.

Предпосылки и обоснование выбора

Для тестирования производительности сетевых операций на платформе HiFive Unleashed нам требовалась операционная система Linux с ядром, поддерживающим функции трассировки и оценки производительности. Первоначально было предпринято несколько попыток добиться желаемых результатов на основе поддерживаемого компанией SiFive пакета SiFive Freedom Unleashed (FU) SDK [1], включающего несколько ветвей с разными версиями ядра Linux. Однако создать загрузочный образ на основе собранного с использованием [1] ядра и поддерживаемого компанией SiFive демонстрационного образа Debian [2] или образа Fedora RISC-V [3], обеспечивающий нужную функциональность и достаточно стабильную работу, не удалось.

В результате был сделан выбор в пользу разрабатываемой компанией SiFive ветви SiFive OpenEmbedded (OE) [4] с поддержкой ядра Linux версии 5.

Сборка подготовленного образа

Ветвь кода SiFive OE построена на основе репозитория repo от Google и существенно отличается от остальной части FU SDK. Строго говоря, это новое направление, связанное с FU SDK лишь общей аппаратной платформой - HiFive Unleashed.

В отличие от FU SDK, здесь не только создаются ядро и компоненты загрузки для платы HiFive Unleashed, но и собирается достаточно функциональный дистрибутив Linux, похожий на Debian.

Рассмотрим сначала создание загрузочных образов с принятыми по умолчанию настройками.

Процесс начинается с установки программы repo, если в вашей системе ее еще нет. Процедура установки и инициализации repo достаточно подробно и внятно описана на странице Google [5], поэтому не будем на этом останавливаться.

После этого нужно создать каталог для дерева исходных кодов, который после загрузки и сборки компонент займет около 80 Гбайт. Далее все ссылки на файлы будут указываться относительно этого каталога (в нашем случае riscv-sifive).

```
$ mkdir riscv-sifive
$ cd riscv-sifive
$ repo init -u git://github.com/sifive/meta-sifive -b master -m tools/manifests/sifive.xml
$ repo sync
```

Последняя команда может выполняться достаточно долго в зависимости от скорости соединения с сетью и производительности дисковой подсистемы компьютера. Общий объем загруженных файлов составляет приблизительно 300 Мбайт.

Затем вводится команда

```
$ repo start work --all
```

для начала работы со всеми ветвями проекта. Следом необходимо настроить среду сборки требуемых образов с помощью команды

```
$ . ./meta-sifive/setup.sh
```

Для тех, кто не знает или запамятовал, отметим, что точка в начале строки означает запуск принятого по умолчанию интерпретатора команд (например, bach). После ввода команды на консоль будет выведена достаточно много сообщений, частично проиллюстрированных ниже.

```
Init OE
You had no conf/local.conf file. This configuration file has therefore been
created for you with some default values. You may wish to edit it to, for
example, select a different MACHINE (target hardware). See conf/local.conf
for more information as common configuration options are commented.

You had no conf/bblayers.conf file. This configuration file has therefore been
created for you with some default values. To add additional metadata layers
into your configuration please add entries to conf/bblayers.conf.

The Yocto Project has extensive documentation about OE including a reference
manual which can be found at:
   http://yoctoproject.org/documentation

For more information about OpenEmbedded see their website:
   http://www.openembedded.org/
```

```
### Shell environment set up for builds. ###

You can now run 'bitbake <target>'

Common targets are:
   core-image-minimal
   core-image-sato
   meta-toolchain
   meta-ide-support

You can also run generated qemu images with a command like 'runqemu qemux86'
Adding layers
NOTE: Starting bitbake server...
...

NOTE: Starting bitbake server...
Creating auto.conf
--------------------------------------------------
MACHINE=freedom-u540 bitbake demo-coreip-cli
--------------------------------------------------

Buildable machine info
--------------------------------------------------
* freedom-u540: The SiFive HiFive Unleased board
* qemuriscv64: The 64-bit RISC-V machine
--------------------------------------------------
```

На этом этапе система готова к сборке загрузочного образа с принятыми по умолчанию параметрами. Для начала сборки следует ввести в консоли команду, приведенную в конце показанного выше вывода

```
$ MACHINE=freedom-u540 bitbake demo-coreip-cli
```

Эта команда указывает тип машины, для которой выполняется сборка (freedom-u540) и образ (demo-coreip-cli), собираемый с помощью команды bitbake. После запуска команды начнется весьма долгий процесс загрузки ядра и требуемых пакетов, их настройки и сборки.

```
Initialising tasks: 100% |#################################################################|
Time: 0:00:18
Parsing of 2389 .bb files complete (0 cached, 2389 parsed). 3490 targets, 134 skipped, 0
masked, 0 errors.
NOTE: Resolving any missing task queue dependencies

Build Configuration:
BB_VERSION           = "1.43.1"
BUILD_SYS            = "x86_64-linux"
NATIVELSBSTRING      = "mageia-7"
TARGET_SYS           = "riscv64-oe-linux"
MACHINE              = "freedom-u540"
DISTRO               = "nodistro"
DISTRO_VERSION       = "nodistro.0"
TUNE_FEATURES        = "riscv64 littleendian"
meta                 = "work:6b36db836547a23f43c5f97bf3706d7b210c209c"
meta-oe
meta-python
meta-multimedia
meta-networking
meta-gnome
meta-xfce            = "work:4e0538516b1e0ef42dc79bd08f7895f0052063ac"
meta-riscv           = "work:99cdf8d0cfe4b515ccac76c816091b146d0a012b"
meta-sifive          = "work:87a209be777318fefb87da6e295f4c34540717f3"

NOTE: Fetching uninative binary shim from
http://downloads.yoctoproject.org/releases/uninative/2.6/x86_64-nativesdk-
libc.tar.xz;sha256sum=133387753a9acf3e1b788103c59fac91e968e2ee331d7a4b9498e926ada7be57
Initialising tasks: 100% |#################################################################|
Time: 0:00:04
Sstate summary: Wanted 1950 Found 0 Missed 1950 Current 0 (0% match, 0% complete)
NOTE: Executing Tasks
NOTE: Setscene tasks completed
...
```

Следует отметить, что в процессе первоначальной настройки с большой вероятностью возникнут те или иные проблемы и процесс завершится ошибкой. В таком случае следует внимательно просмотреть содержимое вывода на консоль и log-файлы. Обычно, при возникновении ошибки, программа выводит на консоль имя файла, в который были записаны сообщения. Анализ вывода позволяет обнаружить недостающие или некорректно настроенные компоненты системы и устранитье неполадки. Пример сообщения о создании журнала ошибок приведен ниже:

```
ERROR: Logfile of failure stored in: riscv-sifive/build/tmp-glibc/work/x86_64-linux/shared-
mime-info-native/1.10-r0/temp/log.do_compile.39743
```

Некоторые ошибки удается найти не сразу по причине большого объема вывода программы сборки. Если прокрутка экрана вверх не позволяет увидеть место возникновения ошибки в процессе сборки, имеет смысл воспользоваться перенаправлением вывода в файл и последующим анализом этого файла. В нашем случае возникла проблема, связанная с пустым файлом

```
riscv-sifive/build/tmp-glibc/work/x86_64-linux/shared-mime-info-native/1.10-r0/build/
freedesktop.org.xml
```

Решить проблему после ее обнаружения удалось путем копирования в упомянутый каталог файла /usr/share/mime/packages/freedesktop.org.xml. Другая проблема была связана со сборкой пакета networkmanager и решить ее удалось только “вручную”, путем сборки части пакета вне процесса bitbake после анализа вывода программы и обнаружения точки возникновения ошибки.

Процесс сборки занимает достаточно продолжительное время (на системе x86 с 40 ядрами и гигабитным каналом доступа в Internet процесс тянулся более часа). В конце концов после некоторого ожидания и правки ошибок будет выведено сообщение вида

```
...
NOTE: Tasks Summary: Attempted 5681 tasks of which 5423 didn't need to be rerun and all
succeeded.
NOTE: Writing buildhistory
NOTE: Writing buildhistory took: 1 seconds
NOTE: Build completion summary:
NOTE:   do_populate_sysroot: 0.0% sstate reuse(0 setscene, 6 scratch)
NOTE:   do_package_qa: 0.0% sstate reuse(0 setscene, 8 scratch)
NOTE:   do_package: 0.0% sstate reuse(0 setscene, 7 scratch)
NOTE:   do_packagedata: 0.0% sstate reuse(0 setscene, 7 scratch)
NOTE:   do_package_write_ipk: 0.0% sstate reuse(0 setscene, 8 scratch)
NOTE:   do_populate_lic: 0.0% sstate reuse(0 setscene, 2 scratch)

Summary: There was 1 WARNING message shown.
```

Это означает, что процедура установки, настройки и сборки компонент завершилась без ошибок и полученный образ можно копировать на карту микро-SD для загрузки платы HiFive Unleashed. Подготовленные образы для установки на карту микро-SD будут размещаться в каталоге riscv-sifive/build/tmp-glibc/deploy/images/freedom-u540.Процесс сборки завершается в каталоге riscv-sifive/build. Для переноса образов на карту микро-SD необходимо выполнить команды

```
$ cd  ./tmp-glibc/deploy/images/freedom-u540
$ zcat demo-coreip-cli-freedom-u540.wic.gz | sudo dd of=/dev/sdX bs=512K iflag=fullblock
oflag=direct conv=fsync status=progress
```

где вместо sdX нужно указать имя реального устройства, с которым ваша система видит подключенную к ней карту микро-SD. В нашем случае карта имела в системе имя sdk, которое дальше и будет указываться. Для установки образов достаточно карты размером 8 Гбайт. Если вы возьмете карту большего размера, на ней все равно будут созданы разделы таких размеров, как будто использовалась карта на 8 Гбайт. При желании можно после копирования изменить размер корневого раздела с помощью gparted или иной подходящей программы.

```
$ zcat demo-coreip-cli-freedom-u540.wic.gz | sudo dd of=/dev/sdk bs=512K iflag=fullblock
oflag=direct conv=fsync status=progress
[sudo] пароль для user:
7124549632 байт (7,1 GB, 6,6 GiB) скопирован, 523 s, 13,6 MB/s
13600+1 записей получено
13600+1 записей отправлено
7130334208 байт (7,1 GB, 6,6 GiB) скопирован, 523,467 s, 13,6 MB/s
```

Отключаем карту микро-SD от сборочного компьютера, переносим ее в гнездо платы HiFive Unleashed и включаем питание платы. Для наблюдения за процессом использовалась консоль программы screen, запущенной на отладочной машине, которая соединена с платой кабелем USB. Вывод консоли начала процесса загрузки платы HiFive Unleashed с созданным образом частично показан ниже.

```
SiFive FSBL:       2019-09-18-128f282-dirty
Using FSBL DTB
HiFive-U serial #: 00000220
Loading boot payload........


OpenSBI v0.4-21-ga2a7763 (Sep 18 2019 16:39:38)

   ____                    _____ ____ _____
  / __ \                  / ____|  _ \_   _|
 | |  | |_ __   ___ _ __ | (___ | |_) || |
 | |  | | '_ \ / _ \ '_ \ \___ \|  _ < | |
 | |__| | |_) |  __/ | | |____) | |_) || |_
  \____/| .__/ \___|_| |_|_____/|____/_____|
        | |
        |_|

Platform Name          : SiFive Freedom U540
Platform HART Features : RV64ACDFIMSU
Platform Max HARTs     : 5
Current Hart           : 1
Firmware Base          : 0x80000000
Firmware Size          : 100 KB
Runtime SBI Version    : 0.1

PMP0: 0x0000000080000000-0x000000008001ffff (A)
PMP1: 0x0000000000000000-0x0000007fffffffff (A,R,W,X)


U-Boot 2019.07 (Sep 18 2019 - 16:32:19 +0000)

CPU:   rv64imafdc
Model: SiFive HiFive Unleashed A00
DRAM:  8 GiB
MMC:   spi@10050000:mmc@0: 0
In:    serial@10010000
Out:   serial@10010000
Err:   serial@10010000
Net:   eth0: ethernet@10090000
Hit any key to stop autoboot:  2 ### 1 ### 0
switch to partitions #0, OK
mmc0 is current device
```

```
Scanning mmc 0:1...
Found U-Boot script /boot.scr.uimg
678 bytes read in 3 ms (220.7 KiB/s)
## Executing script at 88100000
9326152 bytes read in 4707 ms (1.9 MiB/s)
## Loading kernel from FIT Image at 88300000 ...
   Using 'conf@sifive_hifive-unleashed-a00-microsemi.dtb' configuration
   Trying 'kernel@1' kernel subimage
     Description:  Linux kernel
     Type:         Kernel Image
     Compression:  gzip compressed
     Data Start:   0x883000fc
     Data Size:    9317433 Bytes = 8.9 MiB
     Architecture: RISC-V
     OS:           Linux
     Load Address: 0x80200000
     Entry Point:  0x80200000
     Hash algo:    sha256
     Hash value:   62f8986a3de46f822f9c75388982836bf6f43c4de9a5cf63d4288c7454f51dc0
   Verifying Hash Integrity ... sha256+ OK
## Loading fdt from FIT Image at 88300000 ...
   Using 'conf@sifive_hifive-unleashed-a00-microsemi.dtb' configuration
   Trying 'fdt@sifive_hifive-unleashed-a00-microsemi.dtb' fdt subimage
     Description:  Flattened Device Tree blob
     Type:         Flat Device Tree
     Compression:  uncompressed
     Data Start:   0x88be2e64
     Data Size:    6677 Bytes = 6.5 KiB
     Architecture: RISC-V
     Load Address: 0x82200000
     Hash algo:    sha256
     Hash value:   b704ade210b1ac4161e2abb16a5e0e1a437aae731b9b49e7b5c41f357e4a8e4e
   Verifying Hash Integrity ... sha256+ OK
   Loading fdt from 0x88be2e64 to 0x82200000
   Booting using the fdt blob at 0x82200000
   Uncompressing Kernel Image ... OK
   Using Device Tree in place at 0000000082200000, end 0000000082204a14

Starting kernel ...
```

Уже самое начало процесса загрузки показывает, что он организован иначе, нежели в образах FU SDK - вместо загрузчика BBL [6, 7] используется OpenSBI [8, 9]. Не будем сейчас вдаваться в детали, а лишь отметим, что загрузка системы в этом варианте во много раз быстрей по сравнению с FU SDK на основе BBL.

По-умолчанию, для входа в систему служит имя пользователя root с пустым паролем (Enter). Изучением полученной системы каждый может заняться самостоятельно, а мы рассмотрим некоторые вопросы настройки создаваемого образа.

Настройка параметров ядра и компонент системы

Настройка параметров и компонент образа описана достаточно подробно в документах [10 - 12]. Система имеет также графический Web-интерфейс Toaster [13] для настройки компонент собираемого образа. Желающие могут изучить в деталях варианты настроек самостоятельно по приведенным в конце ссылкам. Нам же нужно изменить конфигурацию ядра Linux и добавить некоторое число пакетов в образ, что можно сделать вручную, просто редактируя нужные файлы.

Параметры конфигурации ядра определяются файлом riscv-sifive/build/tmp-glibc/work/freedom_u540-oe-linux/linux-mainline/5.2.9+gitAUTOINC+aad39e30fb-r0/linux-freedom_u540-standard-build/.config. Этот файл создается или обновляется в процессе сборки образов на основе файла riscv-sifive/meta-sifive/recipes-kernel/linux/files/freedom-u540/defconfig, содержащего набор конфигурационных параметров ядра, а также других фрагментов конфигурации, размещенный в многочисленных файлах. Файл defconfig достаточно безопасно можно редактировать вручную, если вы представляете смысл и назначение изменяемых параметров. Однако следует отметить, что при создании итоговой конфигурации ядра файл defconfig анализируется в самом начале процесса и некоторые опции, заданные в нем могут быть переопределены другими фрагментами конфигурации.

В нашем случае к заданной по умолчанию конфигурации были лишь добавлены (отредактированы) опции, связанные с трассировкой и отладкой ядра. Подробно останавливаться на них мы не будем, поскольку каждый будет выбирать их самостоятельно в соответствии с задачами. Приведем лишь краткий список опций, которые нужны для трассировки и оценки производительности.

```
CONFIG_LATENCYTOP=y
CONFIG_NOP_TRACER=y
CONFIG_HAVE_FUNCTION_TRACER=y
CONFIG_HAVE_FUNCTION_GRAPH_TRACER=y
CONFIG_HAVE_DYNAMIC_FTRACE=y
CONFIG_HAVE_DYNAMIC_FTRACE_WITH_REGS=y
CONFIG_HAVE_FTRACE_MCOUNT_RECORD=y
CONFIG_HAVE_SYSCALL_TRACEPOINTS=y
CONFIG_TRACER_MAX_TRACE=y
CONFIG_TRACE_CLOCK=y
CONFIG_RING_BUFFER=y
```

```
CONFIG_EVENT_TRACING=y
CONFIG_CONTEXT_SWITCH_TRACER=y
CONFIG_RING_BUFFER_ALLOW_SWAP=y
CONFIG_PREEMPTIRQ_TRACEPOINTS=y
CONFIG_TRACING=y
CONFIG_GENERIC_TRACER=y
CONFIG_TRACING_SUPPORT=y
CONFIG_FTRACE=y
CONFIG_FUNCTION_TRACER=y
CONFIG_FUNCTION_GRAPH_TRACER=y
CONFIG_PREEMPTIRQ_EVENTS=y
CONFIG_IRQSOFF_TRACER=y
CONFIG_SCHED_TRACER=y
CONFIG_HWLAT_TRACER=y
CONFIG_FTRACE_SYSCALLS=y
CONFIG_TRACER_SNAPSHOT=y
CONFIG_TRACER_SNAPSHOT_PER_CPU_SWAP=y
CONFIG_TRACE_BRANCH_PROFILING=y
CONFIG_PROFILE_ALL_BRANCHES=y
CONFIG_STACK_TRACER=y
CONFIG_DYNAMIC_FTRACE=y
CONFIG_DYNAMIC_FTRACE_WITH_REGS=y
CONFIG_FUNCTION_PROFILER=y
CONFIG_FTRACE_MCOUNT_RECORD=y
CONFIG_TRACEPOINT_BENCHMARK=y
CONFIG_PREEMPTIRQ_DELAY_TEST=m
```

Если редактировать файл вручную вы не рискнете или результаты такого редактирования не устроят (нужные опции будут переопределены в других файлах), можно воспользоваться инструментами devtool, подробно описанными в [12] (параграф 15.3.3 и далее). Поскольку детали процесса в нашем случае несколько отличаются и не вполне очевидны, ниже приведен список команд, которые позволят активизировать на сборочном хосте пакет интерактивной настройки ядра menuconfig.

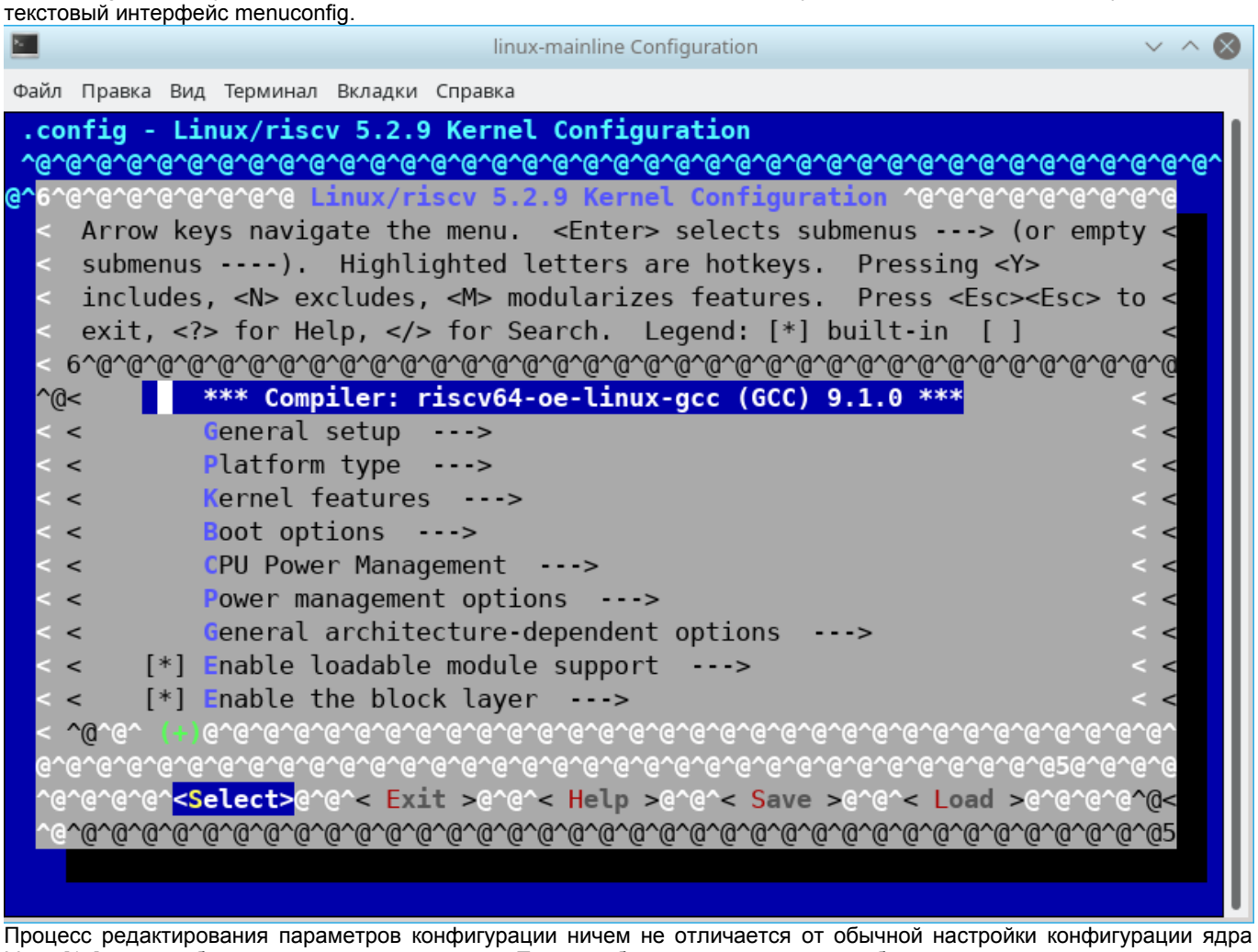
Из каталога сборки (build) запускаем команду для создания стартового файла конфигурации ядра (.config).

```
bitbake linux-mainline -c kernel_configme -f
```

Если вы уже собирали образ с принятыми по умолчанию настройками, файл конфигурации ядра .config уже имеется в дереве исходных кодов и команду можно опустить, но в любом случае ее запуск не нанесет никакого урона. Далее из того же каталога вводим команду

```
bitbake linux-mainline -c menuconfig
```

На экране появится новое консольное окно linux-mainline Configuration с выводом процесса запуска команды make menuconfig, используемой ядром Linux для интерактивной настройки конфигурации, а затем в этом окне будет выведен текстовый интерфейс menuconfig.



Процесс редактирования параметров конфигурации ничем не отличается от обычной настройки конфигурации ядра Linux [14] и мы не будем здесь останавливаться. После выбора нужных опций не забудьте сохранить результаты.

После редактирования конфигурации ядра следует расширить набор включаемых в образ операционной системы пакетов и утилит в соответствии с вашими потребностями. Компоненты добавляются с помощью переменной CORE_IMAGE_EXTRA_INSTALL += в файле build/conf/local.conf. Полный список доступных пакетов, отсортированных по алфавиту, можно получить с помощью команды bitbake -s. При добавлении нужных вам пакетов можно не задумываться над зависимостями, они будут проверены автоматически с добавлением нужных пакетов без вашего участия. Добавленные в нашем случае приложения показаны ниже.

```
CORE_IMAGE_EXTRA_INSTALL += "apt numactl ptpd hwloc netperf man tzdata bootchart \
                             swig cmake jsoncpp tcpdump"
```

После выполнения описанных выше операций нужно повторить сборку ядра и пакетов ОС, а также копирование полученных образов на карту микро-SD.

```
$ MACHINE=freedom-u540 bitbake demo-coreip-cli
…
$ cd  ./tmp-glibc/deploy/images/freedom-u540
$ zcat demo-coreip-cli-freedom-u540.wic.gz | sudo dd of=/dev/sdk bs=512K iflag=fullblock
oflag=direct conv=fsync status=progress
```

Работу с полученным образом для трассировки и оценки производительности рассмотрим в следующей публикации.

Заключение

В этой статье были рассмотрены некоторые детали настройки конфигурации ядра и всего процесса разработки с RISC-V и встроенным Linux перед запуском реального приложения. В дальнейшем мы сможем добавлять и проектировать периферийные устройства и разрабатывать соответствующие драйверы для Linux, совместимые с RISC-V платой HiFive Unleashed.

Литература

[1] SiFive Freedom Unleashed SDK, https://github.com/sifive/freedom-u-sdk.

[2] HiFive U RiscV-pc alpha release, https://github.com/tmagik/freedom-u-sdk/releases/download/hifiveu-2.0-alpha.1/.

[3] Architectures/RISC-V/Installing, https://dl.fedoraproject.org/pub/alt/risc-v/disk-images/fedora/rawhide/20190703.n.0/Developer/Fedora-Developer-Rawhide-20190703.n.0-sda.raw.xz.

[4] SiFive OpenEmbedded Layer, https://github.com/sifive/meta-sifive.

[5] Installing Repo, https://source.android.com/setup/build/downloading#installing-repo.

[6] All Aboard, Part 6: Booting a RISC-V Linux Kernel, https://www.sifive.com/blog/all-aboard-part-6-booting-a-risc-v-linux-kernel.

[7] RISC-V Proxy Kernel and Boot Loader, https://github.com/riscv/riscv-pk.

[8] RISC-V Open Source Supervisor Binary Interface (OpenSBI), https://github.com/riscv/opensbi

[9] RISC-V Supervisor Binary Interface Specification, https://github.com/riscv/riscv-sbi-doc/blob/master/riscv-sbi.adoc.

[10] BitBake User Manual, https://www.yoctoproject.org/docs/2.7.1/bitbake-user-manual/bitbake-user-manual.html.

[11] A practical guide to BitBake, https://a4z.bitbucket.io/docs/BitBake/guide.html.

[12] Yocto Project Reference Manual, https://www.yoctoproject.org/docs/2.7.1/ref-manual/ref-manual.html#devtool-modifying-a-recipe.

[13] Toaster User Manual, https://www.yoctoproject.org/docs/2.7.1/toaster-manual/toaster-manual.html.

[14] Menuconfig, https://en.wikipedia.org/wiki/Menuconfig.

Николай Малых

nmalykh@protocols.ru