Тестирование платы HiFive Unleashed с ядром Linux

Плата HiFive Unleashed [1] производства компании SiFive основана на процессоре Freedom U540 [2], включающем одно 64-битовое ядро E51 RISC-V, используемое для мониторинга и управления, а также четыре 64-битовых ядра E54 RISC-V, применяемых для решения прикладных задач. Схемы платы подробно описаны в документе [3], а внешний вид представлен на рисунке 1, заимствованном из [1].

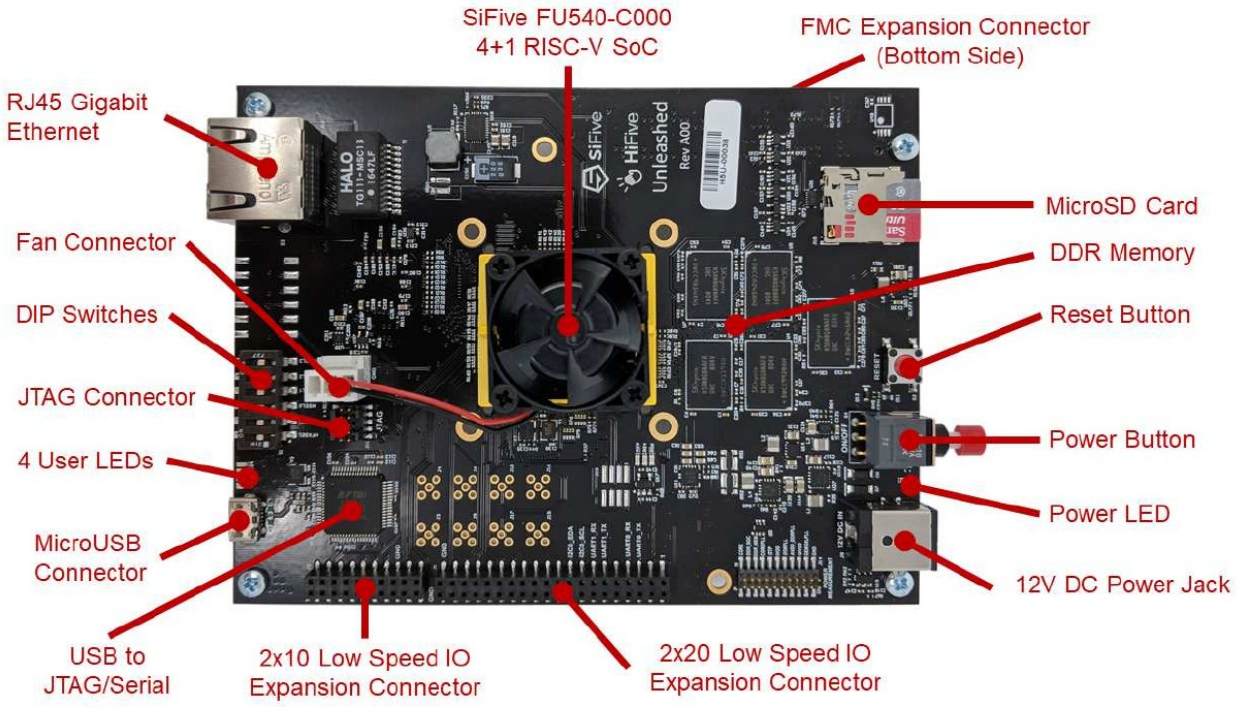

Рисунок 1. Внешний вид платы HiFive Unleashed.

Подробное рассмотрение устройства платы в наши задачи не входило, поэтому отошлем интересующихся читателей к документам [1-3]. Мы же рассмотрим более подробно вопросы загрузки ОС Linux, а также возможности трассировки ядра для оценки производительности сетевых операций.

В комплект поставки платы входит карта микро-SD с загрузчиком BBL и простой оболочкой Buildroot на основе ядра Linux 4.15.0. Для наших задач это не представляло сколь-нибудь существенного интереса, поэтому мы сразу начали сборку своей системы на основе Freedom Unleashed Software Development Kit [4]. Репозиторий GitHub для Freedom U-SDK содержит полный набор исходных кодов и инструментов для кросс-компиляции загрузчика bbl на основе ядра Linux (в настоящее время версии 4.19) и самого ядра. Обеспечивается также загрузка и установка демонстрационной ОС Linux семейства Debian. Для работы с SDK в вашей системе должны быть установлены перечисленные ниже пакеты[1].

- git;
- build-essential;
- autotools;
- texinfo;
- bison;
- flex;
- lib64gmp-devel;
- lib64mpfr-devel;
- gawk;
- lib64zlib-devel;
- lib64openssl-devel;
- dtc;
- mtools.

Кроме того, потребуется около 16 Гбайт свободного пространства, из которого примерно половину составляет код, загружаемый из сети.

Загрузка и сборка SDK

От своего имени (не root) вводим команду

```
$ git clone https://github.com/sifive/freedom-u-sdk.git
```

После загрузки файлов, которая может занять достаточно продолжительное время, переходим в нужный каталог

```
$ cd freedom-u-sdk
```

и загружаем требуемые для сборки субмодули с помощью команды

```
$ git submodule update --recursive -init
```

[1]Список приведён для системы Mageia v7 x86_64. В файле README.md в корневом каталоге Freedom U-SDK приведён список пакетов, требуемых для Ubuntu 16.04 x86_64.

Пока конфигурационные параметры ядра в дереве ещё отсутствуют (они создаются в процессе сборки) и править нечего, поэтому просто запускаем от своего имени из корневого каталога SDK команду

```
$ make
```

После этого начнётся процедура настройки и сборки ядра с установленными по умолчанию параметрами. Эти параметры, как выяснилось, не вполне подходили для наших экспериментов, но a-priori этого не было известно. Процедура сборки ядра и загрузчиков занимает достаточно много времени и можно расслабиться или заняться другими делами[1].

Тут возможны два варианта - дождаться завершения сборки с принятыми по умолчанию настройками или прервать ее и заняться настройкой параметров конфигурации ядра в соответствии со своими задачами. Для полноты картины рассмотрим более длинный вариант со сборкой первоначально без какой-либо своей настройки.

Итак, процесс в конце концов завершился и на экране появилось сообщение

```
GPT (for SPI flash or SDcard) and U-boot Image files have
been generated for an ISA of rv64imafdc and an ABI of lp64d

/home/user/SRC/freedom-u-sdk/work/image-ddbe382-dirty.fit
/home/user/SRC/freedom-u-sdk/work/hifive-unleashed-ddbe382-dirty.gpt

To completely erase, reformat, and program a disk sdX, run:
 make DISK=/dev/sdX format-boot-loader
 ... you will need gdisk and e2fsprogs installed
 Please note this will not currently format the SDcard ext4 partition
 This can be done manually if needed
```

Следующим этапом является перенос созданных образов на карту микро-SD для установки этой карты на плате HiFive Unleashed и последующей загрузки системы. Для этого нужна будет микро-SD карта размером 8 Гбайт, которую следует поместить в тот или иной считыватель, а затем ввести команду

```
$ make DISK=/dev/sdX format-boot-loader
```

заменив sdX именем устройства, на которое карта SD отображается в вашей системе[2]. Предположим, что это /dev/sdj.

```
$ sudo make DISK=/dev/sdj format-boot-loader
[sudo] пароль для user:
/sbin/sgdisk --clear  \
--new=1:2048:65502 --change-name=1:"Vfat Boot" --typecode=1:EBD0A0A2-B9E5-4433-87C0-68B6B72699C7       \
--new=2:264192:15521840 --change-name=2:root --typecode=2:0FC63DAF-8483-4772-8E79-3D69D8477DE4 \
--new=3:1100:2020 --change-name=3:uboot --typecode=3:5B193300-FC78-40CD-8002-E86C45580B47 \
--new=4:1024:1099 --change-name=4:uboot-env --typecode=4:a09354ac-cd63-11e8-9aff-70b3d592f0fa \
      /dev/sdj
Setting name!
partNum is 0
Setting name!
partNum is 1
Setting name!
partNum is 2
Setting name!
partNum is 3
The operation has completed successfully.
/sbin/partprobe
dd if=/home/user/SRC/freedom-u-sdk/work/HiFive_U-Boot/u-boot.bin of=/dev/sdj3 bs=4096
113+1 записей получено
113+1 записей отправлено
464941 байт (465 kB, 454 KiB) скопирован, 0,051439 s, 9,0 MB/s
dd if=/home/user/SRC/freedom-u-sdk/work/hifive-unleashed-vfat.part of=/dev/sdj1 bs=4096
7931+1 записей получено
7931+1 записей отправлено
32488448 байт (32 MB, 31 MiB) скопирован, 2,81572 s, 11,5 MB/s
```

По завершении работы команды можно проверить, что на SD-карте создана таблица GPT с 4 разделами

```
# fdisk -l
[…]
Диск /dev/sdj: 7,4 GiB, 7948206080 байт, 15523840 секторов
Disk model: STORAGE DEVICE
Единицы: секторов по 1 * 512 = 512 байт
Размер сектора (логический/физический): 512 байт / 512 байт
Размер I/O (минимальный/оптимальный): 512 байт / 512 байт
Тип метки диска: gpt
Идентификатор диска: E51737DC-8AD9-4575-8DFD-AF167E7F246B

Устр-во    начало    Конец  Секторы Размер Тип
/dev/sdj1    2048    65502    63455    31M Microsoft basic data
/dev/sdj2  264192 15521840 15257649   7,3G Файловая система Linux
/dev/sdj3    1100     2020      921 460,5K HiFive Unleashed FSBL
/dev/sdj4    1024     1099       76    38K неизвестный

Элементы таблицы разделов упорядочены не так, как на диске.
```

Установленные в результате образы позволят загрузить лишь среду buildroot, толку от которой весьма мало, как можно видеть из приведенного ниже вывода.

```
Welcome to Buildroot

Password:
# help
Built-in commands:
------------------
       . : [ [[ alias bg break cd chdir command continue echo eval exec
       exit export false fg getopts hash help history jobs kill let
       local printf pwd read readonly return set shift source test times
       trap true type ulimit umask unalias unset wait
```

[1]При попытке ускорить сборку за счёт использования множества потоков с опцией make -j процесс завершается ошибкой 2 по причине несинхронизированной сборки отдельных компонент. В этом случае можно повторить команду и тогда все будет собрано корректно и в нужном порядке. Но и в этом случае времени потребуется немало.

[2]Определить имя устройства можно с помощью команды fdisk -l. Здесь следует соблюдать осторожность, поскольку указание неверного имени устройства может привести к удалению всех данных с неверно указанного устройства.

Перечисленные по команде help доступные команды оболочки Buildroot для нас интереса не представляют, поэтому сразу пойдем дальше и установим на SD-карту образ Debian Linux, предоставляемый SiFive. Для этого введите команду

```
$ sudo make DISK=/dev/sdj format-demo-image
```

В результате из сети будет загружен образ sifive-debian-demo-mar7.tar.xz (он сохранится локально в каталоге Freedom U-SDK), который будет скопирован в раздел /dev/sdj2 с помощью утилиты dd. Процедура загрузки и установки образа (примерно 1,1 Гбайт) займет некоторое время, поэтому наберитесь терпения.

После завершения процесса SD-карту можно отключить от компьютера и установить в гнездо платы HiFive Unleashed. Установив карту, подсоединяем кабель микро-USB к компьютеру, который будет служить для контроля за процессом загрузки и последующей работы с платой. Для подключения к плате можно воспользоваться утилитой screen или другой программой эмуляции терминала по вашему выбору. Включаем питание платы и вводим на консоли управления команду

```
$ sudo screen -L /dev/ttyUSB1 115200
```

Опция -L задает запись консольного вывода в журнальный файл screenlog.0. В процессе работы этот файл может дать вам обширную информацию о процессе загрузки и поможет при устранении проблем, если таковые возникнут. Если вам удобней сохранять консольный вывод в другой файл, его можно задать опцией -Logfile <имя файла> в командной строке screen.

Если все пошло нормально, через несколько секунд на консоли появится вывод процесса загрузки и примерно через 30 секунд после включения должен замигать зелёный светодиод, ближний к разъему микро-USB на плате. Если что-то пошло не так, попробуйте изменить положение DIP-переключателей на плате. В описании платы указано, что по умолчанию переключатели должны быть установлены в положение 1111 (левая позиция 4 нижних переключателей на рисунке 1). На практике у меня созданный с помощью Freedom U-SDK образ грузился при положении переключателей 1101 (второй снизу на рисунке 1 переключатель сдвинут вправо). Этот факт отмечен и в файле README.md, хранящемся в корневом каталоге Freedom U-SDK.

После загрузки система запросит имя (root) и пароль (sifive), а после их ввода вы окажетесь в среде buildroot (если не установили образ Debian). Можно поэкспериментировать с этой средой, но для наших задач в ней ничего интересного не нашлось. Если же вы установили Debian, после ввода имени и пароля вы окажетесь в обычной консоли Linux. Изначально состав установленных приложений не богат, но можно установить пакеты из настроенного изначально репозитория с помощью обычных команд apt или apt-get. На этом вопросе мы останавливаться не будем.

Итак, мы загрузили демонстрационную ОС Debian Linux и готовы начать работу с ней. Вот только сразу же сталкиваемся с неприятным сюрпризом - файловая система смонтирована в режиме только чтения (read-only). Поиск командной строки загрузки ядра, где задаётся режим монтирования файловой системы, привёл к файлу uEnv.txt, размещенному в первом разделе SD-карты (Vfat Boot). Для монтирования файловой системы и управления загрузкой в этот файл нужно внести некоторые изменения. Забегая вперёд, отмечу, что исходный файл, который при создании загрузочного образа копируется на SD-карту, расположен в каталоге conf дерева исходных кодов Freedom U-SDK. Поэтому, если вы планируете экспериментировать со сборкой ядра, лучше отредактировать файл uEnv.txt в этом каталоге, чем менять каждый раз его копию на SD-карте.

Итак, начинаем смотреть файл uEnv.txt (в каталоге исходных кодов или на SD-карте по вашему усмотрению). Начало этого файла выглядит как показано ниже. Номера строк слева добавлены мной для удобства ссылок на них в последующем тексте.

```
01 # The current convention (SUBJECT TO CHANGE) is that this file
02 # will be loaded from the first MSDOS(fat) GPT partition on the
03 # MMC card.
04
05 bootargs=debug console=tty0 console=ttySIF0 root=/dev/mmcblk0p2 rootwait
06
07 # To boot from partition 2 of an NVME drive (with a PCI iofpga,
08 # such as the MicroSemi expansion board, uncomment below:
09
10 #bootargs=debug console=tty0 console=ttySIF0 root=/dev/nvme0n1p2
11
12 # to boot an initramfs (buildroot or debian/etc) use this
13 setupchosen=run setupvml; run setupird
14
15 # to boot with straight to the root= parition, uncomment below
16 # so we do not set the ramdisk pointers
17 #setupchosen=run setupvml
18
19 # The FIT file to boot from
20 fitfile=hifiveu.fit
21
22 # The rest of this is mostly of interest to u-boot developers
23 # below much match what's in FIT (ugha)
```

Приступим к редактированию файла uEnv.txt. Перво-наперво откроем файловую систему Linux для записи и чтения. Для этого в строку 5 добавим rw в конце. Можно также убрать параметр debug в начале строки, если избыток отладочной информации вам мешает. В итоге в файле останется строка вида

```
bootargs=rootwait console=tty0 console=ttySIF0 root=/dev/mmcblk0p2 rw
```

Далее, поскольку мы планируем загрузить нормальную ОС Linux, следует убрать символ комментария в начале строки 17 и добавить такой символ в начало строки 13 (иначе будет грузиться Builsroot, а не Debian).

После этого можно установить карту в гнездо платы HiFive и загрузить Linux. Для входа в систему используется имя пользователя root и пароль sifive. После загрузки плата должна получить адрес IP по протоколу DHCP (для этого в вашей сети должен быть действующий сервер DHCP) и запустить демон sshd. Если при этом не возникло никаких ошибок (обычно это так), вы сможете работать с платой уже не только из консоли screen, но и удалённо, по протоколу SSH.

Как уже было отмечено выше, набор пакетов в демонстрационном образе Debian достаточно скромен и для работы его придется дополнить с помощью стандартных команд apt или apt-get. Я бы рекомендовал сразу установить net-tools и

lshw. Далее в тексте предполагается, что все нужные для работы пакеты установлены и напоминаний о них больше не будет.

Для начала разберёмся, что мы имеем в плане аппаратных компонент с помощью команды lshw.

```
root@HiFiveU:~# lshw
hifiveu
   description: Computer
   product: ed_cluster_alloc_exit: OK
   width: 64 bits
   capabilities: smp
 *-core
      description: Motherboard
      physical id: 0
    *-memory
         description: System memory
         physical id: 0
         size: 7993MiB
    *-cpu:0
         physical id: 1
         bus info: cpu@0
         width: 32 bits
    *-cpu:1
         physical id: 2
         bus info: cpu@1
         width: 32 bits
    *-cpu:2
         physical id: 3
         bus info: cpu@2
         width: 32 bits
    *-cpu:3
         physical id: 4
         bus info: cpu@3
         width: 32 bits
 *-network
      description: Ethernet interface
      physical id: 1
      logical name: eth0
      serial: 70:b3:d5:92:f2:20
      size: 1Gbit/s
      capacity: 1Gbit/s
      capabilities: ethernet physical tp mii 10bt 10bt-fd 100bt 100bt-fd 1000bt 1000bt-fd autonegotiation
      configuration: autonegotiation=on broadcast=yes driver=macb duplex=full ip=192.168.0.3 link=yes
multicast=yes port=MII speed=1Gbit/s
```

Мы видим 64-битовую систему с материнской платой, поддерживающей многопроцессорную архитектуру SMP, 8 Гбайт (7993 MiB) оперативной памяти, 4 процессора и 1-гигабитный сетевой адаптер, для которого используется драйвер macb (запомните это - пригодится при настройке конфигурации ядра). Утилита почему-то воспринимает процессоры 32-разрядными (я не стал разбираться с этим вопросом), однако другая утилита lscpu показывает более достоверные параметры процессоров.

```
root@HiFiveU:~# lscpu
Architecture:        riscv64
Byte Order:          Little Endian
CPU(s):              4
On-line CPU(s) list: 0-3
Thread(s) per core:  1
Core(s) per socket:  1
Socket(s):           4
L1i cache:           16K
```

На этом подготовку к сборке нового ядра можно считать завершённой. Добавлю лишь, что других устройств в системе не обнаружено и при настройке конфигурации ядра их можно будет смело отключить в целях экономии времени на сборку и снижения размера полученного в результате ядра.

Возвращаемся к системе, где у нас находится дерево исходных кодов Freedom U-SDK, и начинаем настраивать конфигурацию ядра. Для этого переходим в каталог work/linux (ссылки здесь и далее указываются относительно корня дерева Freedom U-SDK). Здесь мы видим множество файлов с расширением .dwo и после них (при сортировке по именам) будут два файла .config и .config.old. Скопируйте на всякий случай первый из этих файлов, он может пригодится для возврата к исходной конфигурации в случае ошибок. Далее нужно этот файл отредактировать. Можно сделать это вручную, но в таком случае достаточно велика вероятность ошибок, да и работа получится трудоёмкой. разумней воспользоваться общепринятыми при настройке конфигурации ядра командами make menuconfig (консольный интерфейс выбора опций) или более удобным вариантом make xconfig, имеющим графический интерфейс. Если для работы этих команд потребуется установить дополнительные пакеты, система сообщит вам об этом.

Не будем здесь вдаваться в детали, ограничившись указанием опций, которые нужно включить, и кратким списком того, что заведомо не потребуется и можно отключить. Итак отключаем в конфигурации ядра:

- поддержку звука;
- поддержку графики;
- все сетевые интерфейсы (устройства) за исключением Ethernet;
- все драйверы Ethernet за исключением Cadence.

Следует включить опции CONFIG_HVC_DRIVER, CONFIG_HVC_RISCV_SBI и CONFIG_VIRTIO_CONSOLE в разделе Non-8250 serial port support. Это обеспечит вывод информации о загрузке системы уже с самых первых стадий процесса.

Поскольку мы планируем заняться трассировкой ядра, включим опции трассировки, как показано на рисунке 2.

Сохраняем созданную конфигурацию и выходим из интерфейса настройки параметров ядра. Не следует беспокоиться о том, что те или иные важные опции не были учтены, поскольку в начале сборки нового ядра будут заданы вопросы, уточняющие важные для нашего случая параметры. Собирать ядро командой make из каталога work/linux не следует,

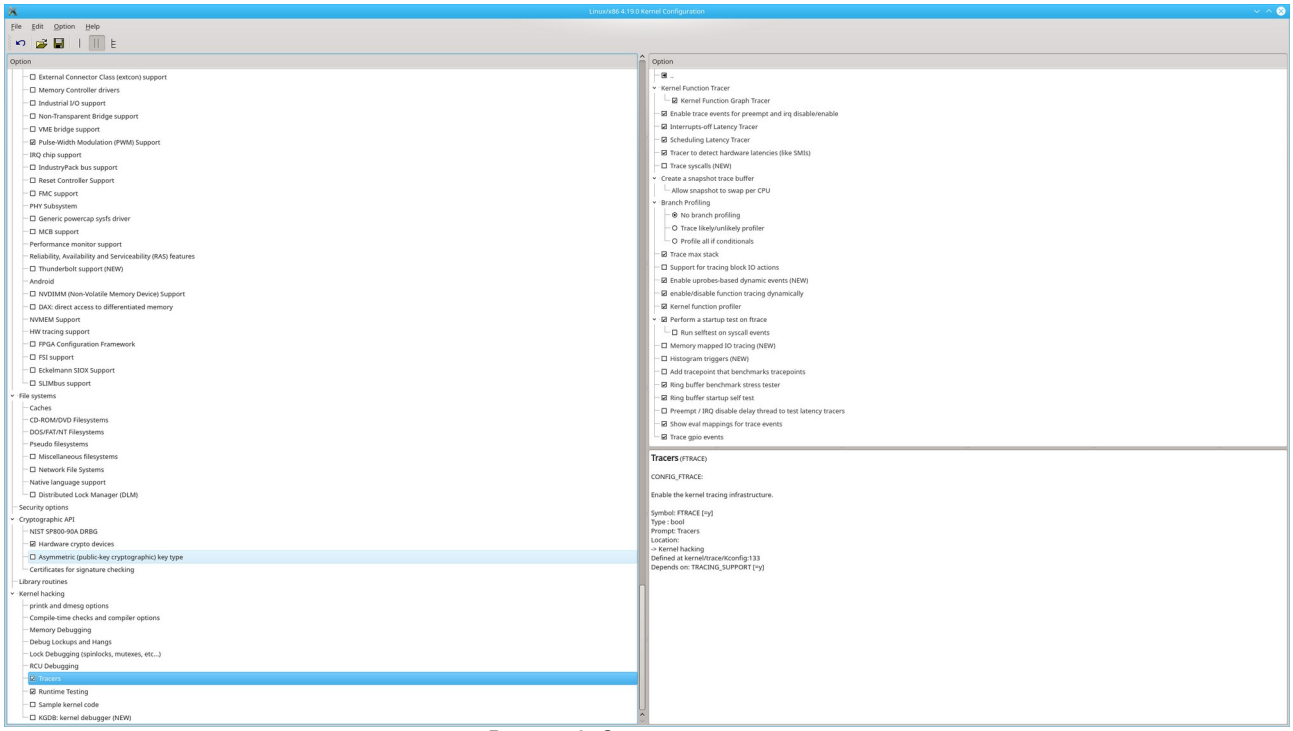

Рисунок 2. Опции трассировки.

поскольку в нашем случае сборка ядра является лишь частью общего процесса. Поэтому возвращаемся в корень Freedom U-SDK и вводим команду make (или make -j).

```
[user@Lhotze freedom-u-sdk (master)]$ make
make -C /home/user/SRC/freedom-u-sdk/linux O=/home/user/SRC/freedom-u-sdk/work/linux \
       ARCH=riscv \
       CROSS_COMPILE=/home/user/SRC/freedom-u-sdk/work/buildroot_initramfs/host/bin/riscv64-sifive-linux-
gnu- \
       PATH=/home/user/SRC/freedom-u-sdk/work/buildroot_initramfs/host/bin:/usr/share/colorgcc:/usr/
local/bin:/usr/bin:/usr/local/games:/usr/games:/usr/lib64/qt5/bin:/usr/lib64/qt4/bin:/home/user/b
in \
       vmlinux
make[1]: вход в каталог «/home/user/SRC/freedom-u-sdk/linux»
make[2]: вход в каталог «/home/user/SRC/freedom-u-sdk/work/linux»
  GEN     ./Makefile
scripts/kconfig/conf  --syncconfig Kconfig
*
* Restart config...
*
*
* Platform type
*
Base ISA
  1. RV32I (ARCH_RV32I) (NEW)
> 2. RV64I (ARCH_RV64I) (NEW)
```

Как уже было отмечено выше, при сборке выдаются запросы на установку некоторых важных опций после предварительной настройки конфигурации ядра. На приведённый выше вопрос отвечаем 2 (Enter), поскольку ядро нам нужно 64-битовое. На следующие несколько вопросов отвечаем просто нажатием клавиши Enter, соглашаясь с предложенными вариантами.

```
Kernel Code Model
  1. medium low code model (CMODEL_MEDLOW) (NEW)
> 2. medium any code model (CMODEL_MEDANY) (NEW)
choice[1-2?]:
Maximum Physical Memory
  1. 2GiB (MAXPHYSMEM_2GB) (NEW)
> 2. 128GiB (MAXPHYSMEM_128GB) (NEW)
choice[1-2?]:
Symmetric Multi-Processing (SMP) [Y/n/?] y
  Maximum number of CPUs (2-32) (NR_CPUS) [8] 8
CPU Tuning
> 1. generic (TUNE_GENERIC) (NEW)
choice[1]: 1
Emit compressed instructions when building Linux (RISCV_ISA_C) [Y/n/?] (NEW)
*
* supported PMU type
*
Base Performance Monitoring Unit (RISCV_BASE_PMU) [Y/n/?] (NEW)
FPU support (FPU) [Y/n/?] (NEW)
*
* Boot options
*
Built-in kernel command line (CMDLINE_BOOL) [Y/n/?] y
  Built-in kernel command string (CMDLINE) [earlyprintk] earlyprintk
  Built-in command line overrides bootloader arguments (CMDLINE_FORCE) [N/y/?] (NEW)
```

Далее нужно будет изменить некоторые из предлагаемых вариантов

```
*
* Character devices
*
Enable TTY (TTY) [Y/n/?] y
  Virtual terminal (VT) [Y/n/?] y
    Enable character translations in console (CONSOLE_TRANSLATIONS) [Y/n/?] y
    Support for console on virtual terminal (VT_CONSOLE) [Y/n/?] y
    Support for binding and unbinding console drivers (VT_HW_CONSOLE_BINDING) [Y/n/?] y
```

```
 Unix98 PTY support (UNIX98_PTYS) [Y/n/?] y
 Legacy (BSD) PTY support (LEGACY_PTYS) [Y/n/?] y
   Maximum number of legacy PTY in use (LEGACY_PTY_COUNT) [256] 256
 Non-standard serial port support (SERIAL_NONSTANDARD) [N/y/?] n
 HSDPA Broadband Wireless Data Card - Globe Trotter (NOZOMI) [N/m/y/?] n
 GSM MUX line discipline support (EXPERIMENTAL) (N_GSM) [N/m/y/?] n
 Trace data sink for MIPI P1149.7 cJTAG standard (TRACE_SINK) [N/m/y/?] n
/dev/mem virtual device support (DEVMEM) [Y/n/?] y
/dev/kmem virtual device support (DEVKMEM) [N/y/?] n
TTY driver to output user messages via printk (TTY_PRINTK) [N/m/y/?] n
RISC-V SBI console support (HVC_RISCV_SBI) [N/y/?] (NEW)
```

Для последнего вопроса по умолчанию предусмотрен ответ N и его нужно заменить на Y, поскольку консоль SBI обеспечивает вывод информации на самых ранних стадиях загрузки, а нам эта информация будет нужна. Более подробно об этом можно узнать из блога Палмера Даббета [5].

Аналогично предлагается поступить и со следующим вопросом, включив контроллер прерываний PLIC, по умолчанию отключенный.

```
*
* Device Drivers
*
SiFive Platform-Level Interrupt Controller (SIFIVE_PLIC) [N/y/?] (NEW)
```

На последующие вопросы, если они возникнут, просто соглашаемся с принятым по умолчанию вариантом. После завершения настройки опций ядра начнётся процесс компиляции и сборки ядра и загрузчика BBL, по завершении которого нужно повторить описанную выше процедуру копирования образов на карту SD. Если вы не меняли файл uEnv.txt в каталоге conf, а ограничились его редактированием на SD-карте, придётся повторить и эту процедуру.

Получив карту SD с новыми образами, устанавливаем ее в гнездо платы и загружаем систему. При установке описанных выше опций на консоль будет выводиться достаточно много отладочной информации и процесс загрузки займёт 2-3 минуты. Однако выводимая информация содержит достаточно много сведений и часть их будет важна для решения вопросов о возможности трассировки ядра, как будет отмечено ниже.

Напомним, что для последующего просмотра и анализа загрузочных сообщений целесообразно включить запись log-файла в терминальной программе (-L для консоли screen). Загрузим систему и начнем изучать файл с сообщениями загрузки.

Первое, на что следует обратить внимание, - это сообщение о том, что харт с идентификатором 0 замаскирован.

```
[    0.000000] CPU with hartid=0 has a non-okay status of "masked"
```

Сначала это показалось странным, но чёткого обозначения идентификаторов хартов в документации найти не удалось. Видимо причина этого сообщения заключается в том, что харт 0 - это управляющее ядро U51, которое отфильтровывается из дерева устройств загрузчиком BBL [5]. Поскольку никаких явных проблем в дальнейшем с этим не возникает, проигнорируем данную информацию.

Следующее важное сообщение гласит о невозможности трассировки функций

```
[    0.060000] Running postponed tracer tests:
[    0.060000] Testing tracer function: .. no entries found ..FAILED!
[    0.210000] ------------[ cut here ]------------
[    0.220000] WARNING: CPU: 0 PID: 1 at kernel/trace/trace.c:1513 run_tracer_selftest+0x122/0x180
[    0.220000] Modules linked in:
[    0.230000] CPU: 0 PID: 1 Comm: swapper/0 Not tainted 4.19.0-sifive-1+ #37
[    0.230000] Call Trace:
[    0.230000] [<ffffffe00004f718>] walk_stackframe+0x0/0xc4
[    0.230000] [<ffffffe00004f9cc>] show_stack+0x3c/0x46
[    0.230000] [<ffffffe000522ef0>] dump_stack+0x8e/0xc8
[    0.230000] [<ffffffe000055c7a>] __warn.part.3+0xb8/0xfa
[    0.230000] [<ffffffe000055dfa>] warn_slowpath_null+0x4a/0x58
[    0.230000] [<ffffffe0000e99c2>] run_tracer_selftest+0x122/0x180
[    0.230000] [<ffffffe00000b42a>] init_trace_selftests+0x8a/0x10a
[    0.230000] [<ffffffe00004d070>] do_one_initcall+0x44/0x184
[    0.230000] [<ffffffe000000fb8>] kernel_init_freeable+0x216/0x2ce
[    0.230000] [<ffffffe00053582e>] kernel_init+0x1c/0x10e
[    0.230000] [<ffffffe00004e250>] ret_from_exception+0x0/0xc
[    0.240000] ---[ end trace 44c325814a888e3d ]---
[    0.240000] ------------[ cut here ]------------
```

И действительно после загрузки системы трассировщик function отсутствует в списке доступных.

```
root@HiFiveU:~# cat /sys/kernel/debug/tracing/available_tracers
hwlat nop
```

Аналогичная картина наблюдается и для остальных трассировщиков, за исключением hwlat и nop, как можно видеть из приведённых ниже фрагментов вывода.

```
[ … ]
[    0.260000] Testing tracer nop: PASSED
[    0.270000] Testing tracer irqsoff:
[    0.270000] failed to start irqsoff tracer
[    0.270000] .. no entries found ..FAILED!
[ … ]
[    0.300000] Testing tracer wakeup:
[    0.300000] failed to start wakeup tracer
[    0.310000] .. no entries found ..FAILED!
[ … ]
[    0.330000] Testing tracer function_graph:
[    0.330000] Failed to init function_graph tracer, init returned -19
[    0.340000] FAILED!
```

При этом кольцевой буфер ядра для записи сообщений трассировки и отслеживание событий работают корректно, что подтверждает приведённый ниже фрагмент вывода.

```
[   23.250000] Ring buffer PASSED!
[   23.250000] Running tests on trace events:
[   23.260000] Testing event initcall_finish: OK
[   23.380000] Testing event initcall_start: OK
[   23.500000] Testing event initcall_level: OK
[   23.620000] Testing event task_rename: OK
[   23.740000] Testing event task_newtask: OK
```

```
[   23.860000] Testing event cpuhp_exit: OK
[   23.980000] Testing event cpuhp_multi_enter: OK
[   24.100000] Testing event cpuhp_enter: OK
[   24.220000] Testing event softirq_raise: OK
[   24.340000] Testing event softirq_exit: OK
[   24.460000] Testing event softirq_entry: OK
[   24.580000] Testing event irq_handler_exit: OK
[   24.700000] Testing event irq_handler_entry: OK
[   24.820000] Testing event signal_deliver: OK
[   24.940000] Testing event signal_generate: OK
[   25.060000] Testing event workqueue_execute_end: OK
[   25.180000] Testing event workqueue_execute_start: OK
```

Список доступных для трассировки событий можно увидеть после загрузки с помощью команды

```
# cat /sys/kernel/debug/tracing/available_events
```

Список этот достаточно велик, поэтому здесь не приведён. Вы можете самостоятельно выбрать набор трассируемых событий, записав 1 для включения или 0 для выключения трассировки соответствующего события. Например,

```
echo 1 > /sys/kernel/debug/tracing/events/sched/sched_wakeup/enable
```

включит трассировку событий sched_wakeup, а команда

```
echo 0 > /sys/kernel/debug/tracing/events/sched/sched_wakeup/enable
```

выключит ее. Однако рассмотрение вопросов трассировки событий не входило пока в наши планы и не рассматривается здесь. Интересующимся читателям рекомендуется обратиться к документу [6].

Отметим дополнительно, что трассировка ветвлений при настройке конфигурации была выключена осознанно, поскольку во многих случаях попытка ею воспользоваться приводит к «зависанию» системы. Аналогичная ситуация наблюдается и с трассировщиком аппаратных задержек hwlat. Неоднократные попытки воспользоваться тем и другим с разными настройками результатов не дали. Отметим, что система при этом не умирает и ее сетевая активность сохраняется (по меньшей мере она отвечает на ping). Но попытки ввода команд в консоли screen или ssh приводят к тому, что после нажатия клавиши Enter никакие другие действия уже не возможны и никакого вывода на консоль не наблюдается.

Заключение

В результате проведённых экспериментов, результаты которых практически неизменно повторялись, можно сделать вывод о недоступности трассировки функций ядра Linux на платформе HiFive Unleashed. Причины этого пока не совсем ясны и требуется дополнительное изучение. Отметим также, что трассировка событий в системе работает в полном соответствии с документацией, содержащейся в исходном коде ядра Linux.

Литература

[1] SiFive HiFive Unleashed Getting Started Guide, https://sifive.cdn.prismic.io/sifive%2Ffa3a584a-a02f-4fda-b758-a2def05f49f9_hifive-unleashed-getting-started-guide-v1p1.pdf

[2] SiFive FU540-C000 Manual, https://sifive.cdn.prismic.io/sifive%2F834354f0-08e6-423c-bf1f-0cb58ef14061_fu540-c000-v1.0.pdf

[3] HiFive Unleashed Schematics, https://sifive.cdn.prismic.io/sifive%2Ff7173056-bf37-4407-87cb-d5ab76abf61a_hifive-unleashed-a00-schematics.pdf

[4] Freedom Unleashed Software Development Kit, https://github.com/sifive/freedom-u-sdk

[5] Palmer Dabbelt, 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

[6] Исходный код ядра Linux, файл Documentation/trace/events.rst.

Николай Малых

nmalykh@protokols.ru