Учебное пособие по программированию микропроцессора «Байкал-Т1»

Содержание

1 Введение

Для программирования Байкал-Т1 вам потребуется специальный комплект средств разработки SDK (Software Development Kit), поставляемый компанией «Байкал Электроникс»[1], который можно скачать по ссылке https://www.baikalelectronics.ru/upload/iblock/9ca/sdk_baikal_mips_4.13.run

Для установки и использования SDK потребуется компьютер под ОС Linux (хост-компьютер). Скомпилированные программы предназначены для исполнения на целевой платформе. Целевой платформой является оборудование на базе микропроцессора Байкал-Т1:

- разработанное и изготовленное пользователем;
- тестовые комплекты БФК-1.6 или BFK-3.1, поставляемые АО «Байкал Электроникс»;
- оборудование на базе Байкал-Т1 от других поставщиков.

Кроме того, в качестве целевой платформы может использоваться программный эмулятор процессора, исполняемый на хост-компьютере, и поставляемый в составе SDK. Применение эмулятора позволяет разрабатывать и отлаживать программы до создания физических образцов целевого оборудования.

Пакет SDK позволяет разрабатывать программы для платформы Байкал-Т1, исполняемые как под ОС Linux (поставляемой в составе SDK), так и на “голом железе” (bare metal) для разработки собственного системного ПО.

Необходимые ресурсы хост-компьютера:

- ОС Linux с ядром версии не ниже 4.4.1;
- Не менее 4 ГБ оперативной памяти
- Не менее 2 ГБ свободного места на локальном диске.
- Пакеты `coreutils`, `parted`, `kpartx` (для модификации образов файловых систем), `xz`.
- Для некоторых операций могут потребоваться root – права (суперпользователя).

[1] На сегодняшний день (апрель 2018) нет сведений об аналогичных комплектах для BE-T1000 от других поставщиков.

2 Подготовка среды программирования

2.1 Структура SDK

В составе пакета SDK поставляются средства для компиляции, сборки и отладки программ для платформы Байкал-Т1:

- Образ ядра ОС Linux, содержащий исходный код ОС и код целевой платформы, драйверы для всех реализованных устройств и конфигурационный файл для сборки. Поставляемая версия ядра – 4.4.24 или более новая;
- Образ корневой файловой системы (`InitRD`) в виде сжатого диска размером 32 Мб. Файловая система включает в себя минимальный набор необходимых утилит и основные библиотеки;
- Набор драйверов для периферийных устройств, контроллеры которых входят в состав микросхемы Байкал-Т1, в исходных кодах и в скомпилированном виде;
- Средства для кросс-компиляции на основе комплекса программ gcc из-под x86 для целевой архитектуры MIPS32® , в том числе отладчик gdb;
- Функциональный эмулятор процессора Байкал-Т1 на основе ПО с открытым кодом QEMU. Эмулятор позволяет выполнять приложение, скомпилированное при помощи средств кросс-компиляции в файл формата elf под архитектуру MIPS32.

```
baikal
    bin – бинарные исполняемые файлы (утилиты для сборки и прошивки, эмулятор)
    doc – документация
    img – создаваемые ROM образы для прошивки, образы ядра, загрузчика и файловых систем
    lib – библиотеки для работы с адаптером EJTAG
    prebuilts – исходные ROM образы для прошивки, образы ядра и файловых систем
    src
        bootrom – исходные коды для сборки прошивки ROM (BIOS)
        configs – конфигурационные файл для сборки компонентов SDK
        dfu-util – исходные коды dfu-utils
        examples – файлы примеров кода
        genext2fs – утилита для создания файловой системы ext2 без прав супер- пользователя
        initrd – исходные коды и каркас корневой файловой системы
        kernel – исходные коды ядра ОС Linux
        openocd – исходные коды OpenOCD
        qboot – исходный код начального загрузчика для поддерживаемых плат- форм
        qemu – исходные коды эмулятора QEMU
        ramfs – исходные коды и каркас initramfs
        u-boot – исходные коды начального загрузчика U-Boot
        x-tools – исходные коды средств кросс-компиляции и утилит
    usr
        eclipse – графическая среда разработки ПО
        oprofile – OProfile инструмент профилирования для Linux
        scripts – вспомогательные скрипты для модификации и запуска образов системы
        share – прочие файлы и страницы помощи
        x-tools – предкомпилированные средства кросс-компиляции.
```

Рисунок 1. Структура директорий SDK

2.2 Состав программных компонентов

2.2.1 Средства кросс-компиляции (cross-tools)

Средства кросс-компиляции включают в себя набор уже скомпилированных программ `mipsel-unknown-linux-gnu`. Данный набор программ кросс-компиляции содержит инструментарий сборки приложений для целевой платформы MIPS32el на x86_64 машине:

- Binutils-.25.1
- cloog-0.18.4
- Expat-2.1.0
- gcc-5.2.0
- gdb-7.10
- gmp-6.0.0a
- isl-0.14
- libiconv-1.14
- linux-headers-4.4.11
- mpc-1.0.3
- mpfr-3.1.3
- ncurses-6.0
- strace-4.10

2.2.2 Ядро ОС Linux 4.4.100 (Linux kernel)

В состав SDK включён образ ядра ОС Linux, собранный для целевой платформы, а также исходный код ядра, целевой платформы, загрузчика, драйверов для всех реализованных устройств в микропроцессоре Байкал-Т1 и конфигурационные файлы для сборки.

2.2.3 Образ корневой файловой системы (InitRD)

SDK поставляется с образом корневой файловой системы в виде сжатого диска размером 16 МБ для плат, для `qemu` 32 МБ. Файловая система включает в себя минимальный набор необходимых утилит и основные библиотеки. SDK также содержит исходный код используемых утилит и библиотек.

Программы:

- busybox-1.26.2
- dropbear-2016.74
- i2ctools-3.1.0
- lmsensors-3.3.4
- ethtool-4.2
- kexec-2.0.14
- pciutils-3.5.1
- spitools-(24.08.2017)

Библиотеки:

- glibc-2.22
- libgomp-1.0.0
- libatomic.1.1.0

2.2.4 Средства программной эмуляции (qemu-mipsel)

Поставляемый в составе SDK эмулятор QEMU содержит исполняемый файл и исходный код пользовательского образа системы MIPS32el. Пользовательский эмулятор позволяет

выполнять приложение, скомпилированное при помощи средств кросс-компиляции в файл формата elf под архитектуру MIPS32el.

- Один процессор, до 6 ядер MIPS32 P5600;
- Периферийные устройства в соответствии с картой памяти микропроцессора Байкал- T1 (с реализацией в `iomem` только заявленных устройств);
- Назначенные прерывания CPU для микропроцессора Байкал-T1 (для реализованных устройств);
- Блок UART (2 шт.);
- Блок таймеров (3 шт.);
- Блок сторожевого таймера (Watchdog Timer, WDT);
- Блок линий общего назначения (GPIO, 32 линии);
- Системный блок управления (PMU);
- Блок обработки ошибок шины APB;
- Блок контроллера AHCI (SATA, до 3 устройств);
- Блок контроллера I2C (2 шт.) с конечными устройствами (EEPROM 24c02, RTC DS1307);
- Блок контроллера SPI (2 шт.) с конечными устройствами (Flash N25Q256a);
- Блок 1GbE MAC (2 шт.).

2.3 Установка SDK

SDK поставляется в виде самораспаковывающегося архива, файла с именем вида `sdk_baikal_mips_4.13.run`. Для его установки на хост-компьютер под управлением ОС Linux должен быль установлен архиватор `xz` и необходимо произвести описанные ниже действия.

Скопировать с сайта `www.baikalelectronics.ru` в домашнюю директорию `/home` файл `sdk_baikal_mips_4.13.run`, установить атрибут **+x**, разрешающий исполнение файла и запустить программу.

```
$ cd ~
$ wget -P ~/www.baikalelectronics.ru/upload/iblock/9ca/sdk_baikal_mips_4.13.run
$ chmod +x  sdk_baikal_mips_4.13.run
$ ./sdk_baikal_mips_4.13.run
```

Программа установится в подкаталог `~/baikal`, который она создаст в текущем каталоге, и выдаст на терминал сообщение о завершении установки.

Для продолжения работы следует проверить наличие пакетов parted, kpartx, и в случае отсутствия установить их.

2.4 Проверка работоспособности системы сборки SDK

Для проверки корректности установки SDK произвести запуск скрипта сборки для любой доступной целевой платформы. Следуя ниже описанным шагам.

1. Перейти в директорию с управляющими скриптами:

```
$ cd ~/baikal/usr/scripts
```

2. Запустить полную сборку для целевой платформы:

```
$ ./build-boot-img.sh bfk3 --all
```

3. Дождаться окончания работы скрипта.

4. Следующий вывод подтверждает успешность работы и корректность установки SDK:

```
=== Boot ROM image build utility for Baikal-T* based boards ===

  +-------------+-------------+-------------+-------------+-------------+
  |   Section   | Start Offset |   Max Size   |  Data Size   |   Padding    |
  +-------------+-------------+-------------+-------------+-------------+
  | BOOTLOADER  |  0x00000000  |  0x00080000  |  0x0007fd10  |  0x000002f0  |
  | UBOOT_ENV   |  0x00080000  |  0x00010000  |  0x00010000  |  0x00000000  |
  | BOARD_INFO  |  0x00090000  |  0x00010000  |  0x0000010c  |  0x0000fef4  |
  | LINUX_FDT   |  0x000a0000  |  0x00010000  |  0x00003e69  |  0x0000c197  |
  | MULTIIMAGE  |  0x000b0000  |  0x00dd0000  |  0x0073f75b  |  0x006908a5  |
  +-------------+-------------+-------------+-------------+-------------+

  Created ROM image file: 'bfk3.rom', 15204352 bytes (14848 kbytes)

 Bootrom: All done
 INFO:  Build process is done

  ####  #    #  ####   ####  ######  ####   ####
 #      #    # #    # #    # #      #      #
  ####  #    # #      #      ####    ####   ####
      # #    # #    # #    # #           #      #
  ####   ####   ####   ####  ######  ####   ####
```

2.5 Проверка работоспособности эмулятора QEMU для MIPS32el

Для проверки корректности установки QEMU для MIPS32el можно произвести запуск эмулятора с поставляемым образом операционной системы и пустым файлом жесткого диска.

1. Перейти в директорию с управляющими скриптами:

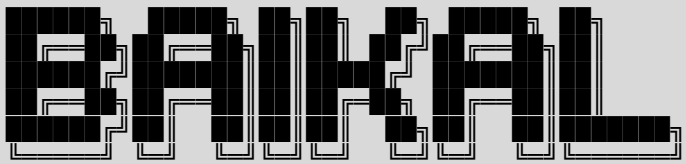
```
$ cd ~/baikal/usr/scripts
```

2. Запустить эмулятор с образом операционной системы и жесткого диска по умолчанию:

```
$ ./run-qemu-mipsel.sh -test
```

Проверка версии ядра выполняется из командной строки консоли командой uname –a. Для окончания симуляции необходимо запустить команду halt, а для выхода из эмулятора нажмите сочетание клавиш <CTRL+a> и <x> в окне терминала.

```
      . .     . . . . .     . . .   .'.     .     . . .  .
     \ /   ' . '       ' . '     '.'   '.'   ' '       '  \ /
      { === Baikal Embedded Linux (BEL), r2018.02-BT1-4.13  === }
     / \_____________________________________________________/ \
```



```
    Done. Have fun!

    Please press Enter to activate this console.
    / # uname -a
    Linux baikal--unknown 4.4.100-qemu #2 SMP Thu Feb 15 17:19:28 MSK 2018 mips
GNU/Linux
    / #
```

3 Программирование под ОС Linux

3.1 Сборка и загрузка ядра ОС Linux с помощью TFTP

Для выполнения загрузки Linux с помощью загрузчика u-boot необходимо пересобрать ядро и корневую файловую систему, скопировать полученные файлы `bfk3.vmlinux.bin`, `initrd.gz`, `bfk3.baikal.dtb` из папки `~/baikal/img` в папку, сконфигурированную для tftp-сервиса, например, `/srv/tftp` и загрузить в режиме tftp.

```
$ cd ~/baikal/usr/scripts
$ ./build-boot-img.sh bfk3 -a
$ cd ~/baikal/img
$ cp bfk3.vmlinux.bin /srv/tftp
$ cp bfk3.dtb /srv/tftp
$ cp initrd.gz /srv/tftp
$ cd /srv/tftp
$ chmod 777 bfk3.vmlinux.bin bfk3.dtb initrd.gz
```

При отсутствии ошибок выполнения целевую плату необходимо перезапустить из консоли командой reboot. U-boot меню загрузки выглядит следующим образом:

```
--== BFK3 boot menu ==--

1.  Boot from SPI flash to minimal FS (rom + ramdisk)
2.  Boot from SPI flash to SATA disk1 (rom + sda1)
3.  Boot from SPI flash to SATA disk2 (rom + sdb1)
4.  Boot from SATA disk1 to minimal FS (disk1 + ramdisk)
5.  Boot from SATA disk1 (sda1)
6.  Boot from SATA disk2 (sdb1)
7.  Boot from Network to minimal FS (tftp + ramdisk)
8.  Boot from Network to SATA disk1 (tftp + sda1)
9.  Boot from Network to SATA disk2 (tftp + sdb1)
10. Boot from Network to minimal FS (dhcp + tftp + ramdisk)
11. Boot from Network to SATA disk1 (dhcp + tftp + sda1)
12. Boot from Network to SATA disk2 (dhcp + tftp + sdb1)
13. Boot from Network to minimal FS (nfs + ramdisk)
14. Boot from Network to SATA disk1 (nfs + sda1)
15. Boot from Network to SATA disk2 (nfs + sdb1)
U-Boot console

  Press UP/DOWN to move, ENTER to select
```

3.2 Программа "Hello World!" на C.

Для создания приложения «Hello world», которая в последствии будет находится в образе сборки ОС Линукса, необходимо выполнить ниже указанный список действий:

1. Создать директорию приложения в директории `~/baikal/src/initrd/programs`, например, `hello`:

```
$ mkdir ~/baikal/src/initrd/programs/hello
```

2. Разархивировать в него исходные файлы приложения `hello`.

```
$ cd ~/baikal/src/initrd/programs/hello
$ tar xvf hello.tar
```

3. При необходимости внести изменения в общий `Makefile`, который находится по адресу: `~/baikal/src/initrd/program`, добавлением соответствующих названий приложений (выделено темным):

```
    . . .
    all:busybox i2ctools lmsensors fbtest ethtool service hello
        @echo "All done"
    . . .
    hello:
        $(MAKE) -C $@ CC="$(CC)" DESTDIR=$(PREFIX)
    . . .
        $(MAKE) -C hello CC="$(CC)" DESTDIR=$(PREFIX) install
    . . .
        cd hello && $(MAKE) clean
    . . .
    .PHONY: all clean install busybox i2ctools lmsensors fbtest ethtool service
hello
```

4. Для добавления программы в пакет образа требуется пересобрать ядро, выполнив скрипт из домашней директории

```
$ ~/baikal/usr/scripts/build-boot-img.sh bfk3 --all
```

5. Затем появится файл образа ядра, корневой файловой системы и начального загрузчика `bfk3.rom` для загрузки на БФК
6. Если необходимо проверить работу в QEMU, тогда необходимо выполнить следующий скрипт для создания файла

```
$ ~/baikal/usr/scripts/build-boot-img.sh qemu --all
```

7. После создания образа `qemu.baikal-mipsel.elf` целевой платформы, можно выполнить команды для проверки работы приложения:

```
$ cd ~/baikal/usr/scripts
$ run-qemu-mipsel.sh -img ~/baikal/img/qemu.baikal-mipsel.elf
/# ./sbin/crossexample
Hello World
```

Для внесения приложения в собственный пакет initrd для SDK 4.11 и выше необходимо выполнить ниже указанный список действий:

1. Открыть с помощью любого, например, nano или vim текстового редактора скрипт `build-initrd-img.sh` и добавить желаемые пакеты в переменную `EXTRA_PKG`

```
$ nano ~/baikal/usr/scripts/build-initrd-img.sh
-------------------------------
# Directory "xutils" and makefile "xutils.mk",
# Directory "xxxtools" and makefile "xxxtools.mk",
EXTRA_PKG="hello"
```

2. пересобрать загрузочный образ ОС Linux и добавленный пакет будет находиться в директории `/sbin`

4 Linux драйверы

4.1 Модульный драйвер

В ОС Linux ядро монолитное, то есть, все драйвера и подсистемы работают в своем адресном пространстве, отделенном от пользовательского. в подавляющем большинстве сборок ядра Linux возможна модификация части кода ядра без его перекомпиляции, и даже без его выгрузки, достигающихся путем загрузки и выгрузки некоторых частей ядра, которые называются модулями. В данном разделе мы разработаем простейший модульный драйвер. Для этого выполним следующие действия:

1. Создаем папку и копируем в нее исходные файлы из архива:

```
$ mkdir ~/baikal/src/initrd/programs/module_driver
$ cd ~/baikal/src/initrd/programs/module_driver
$ tar xvf module_driver.tar
```

2. При необходимости вносим изменения в путях до средств кросс-компиляции
3. В данном каталоге создаются файлы исходного кода `module_driver.c`

```
#include <linux/module.h>
#include <linux/version.h>
#include <linux/kernel.h>

static int __init ofd_init(void) /* Конструктор */
{
     printk(KERN_INFO "Hello world by module driver!");
     return 0;
}

static void __exit ofd_exit(void) /* Дескруктор */
{
     printk(KERN_INFO "Bye cruel world(module driver)!");
}

module_init(ofd_init);
module_exit(ofd_exit);
```

```
    MODULE_LICENSE("GPL");
    MODULE_AUTHOR("Baikal Electronics");
    MODULE_DESCRIPTION("Example Module Driver");
```

4. И файл сборки `Makefile` для нашего приложения:

```
    # Makefile - makefile of our first driver
    # if KERNELRELEASE is defined, we've been invoked from the
    # kernel build system and can use its language.
    ARCH=mips
    COMPILER ?= ~/baikal/usr/x-tools/mipsel-unknown-linux-gnu/bin/mipsel-
unknown-linux-gnu-
    LD := $(CROSS_COMPILE)ld
    STRIP := $(CROSS_COMPILE)strip

    ifneq (${KERNELRELEASE},)
        obj-m := module_driver.o
    else
        KERNEL_SOURCE := ~/baikal/src/kernel
        PWD := $(shell pwd)

    default:
        ${MAKE} -C ${KERNEL_SOURCE} SUBDIRS=${PWD} ARCH=$(ARCH)
CROSS_COMPILE=$(COMPILER) modules

    clean:
        ${MAKE} -C ${KERNEL_SOURCE} SUBDIRS=${PWD} clean

    install: default
        mkdir -p $(DESTDIR)/lib/modules/
        install --mode=755 --strip --strip-program=$(STRIP) --target-
directory=$(DESTDIR)/lib/modules/ $(TARGETS)

    .PHONY:
        all clean install $(TARGETS)
    endif
```

5. Для создания бинарного файла драйвера `module_driver.ko` необходимо выполнить команду `make` из данной директории
6. После получения объектного файла ядра небходимо перенести его на целевую платформу. В случае если требуется пересобрать rootfs, то необходимо скопировать бинарный файл в директорию его скопировать в директорию `~/baikal/_build/initrd/rootfs/lib/modules/` и пересобрать ядро, выполнив скрипт:

```
$ ~/baikal/usr/scripts/build-boot-img.sh bfk3 -a
```

7. После запуска целевой-платформы проверка работоспособности разработанного модуля, осуществляется выполнением следующих команд :

```
/# insmod /lib/modules/module_driver
Hello world by module driver
/# rmmod module_driver
Bye cruel world(module driver)!
/#
```

4.2 Встроенный драйвер

Встроенный драйвер отличается от модульного драйвера тем, что он компилируется вместе с ядром ОС Linux, и во время загрузки ОС подключается автоматически. То есть, для конкретной сборки этот драйвер будет являться частью ОС. В данном разделе мы разработаем простейший встроенный драйвер. Для этого выполним следующие действия:

1. Создаем папку и копируем в нее исходные файлы из архива:

```
$ mkdir ~/baikal/src/kernel/drivers/examples/embedded_driver
$ cd ~/baikal/src/kernel/drivers/examples/embedded_driver
$ tar xvf embedded_driver.tar
```

2. При необходимости вносим изменения в путях до средств кросс-компиляции:
3. В `Makefile`, который находится в папке `~/baikal/src/kernel/drivers` необходимо описать правила для сборки исходных файлов. В данном примере эта правило позволит из исходного файла `embedded_driver.c` получить объектный файл, причем опция `-y` указывает на то, что драйвер будет встроенным в ядро выглядит следующим образом:

```
...
obj-y += embedded_driver/embedded_driver.o
...
```

4. После данных операций следует пересобрать ядро и запустить его, после успешного запуска ОС проверка установки драйвера осуществляется выполнением команды:

```
$ dmesg | grep "Hello"
"Hello world by internal test driver!"
```

5 Bare-metal приложения

Bare-metal приложения – это приложения, работающие на «голом железе» без ОС. В связи с этим, такие приложения должны содержать в себе весь необходимый функционал, требуемый для корректной загрузки и выполнения.

5.1 SimpleAdd приложение

В качестве еще одного примера разработаем простейшую программу SimpleAdd в качестве bare-metal приложения. Для этого необходимо выполнить следующие действия:

1. Создаем папку `~/baikal/src/examples/baremetal` и копируем в нее исходные файлы из архива `simple.tar`

```
$ mkdir ~/baikal/src/examples/simple
$ cd ~/baikal/src/examples/simple
$ tar xvf simple.tar
```

2. При необходимости внести изменение в путях `makefile` до средств кросс-компиляции:

```
. . .
CROSS_COMPILE = ~/baikal/usr/x-tools/mipsel-unknown-linux-gnu/bin/mipsel-
unknown-linux-gnu-
. . .
```

3. Или с помощью export внести собственное значение в переменную CROSS_COMPILE.

```
$ export CROSS_COMPILE=~/baikal/usr/x-tools/mipsel-unknown-linux-gnu/bin/
mipsel-unknown-linux-gnu-
```

4. Воспользоваться утилитой `make` для создания необходимых файлов. Утилита make создаст объектные файлы в директории `obj`, файлы для загрузки в `qemu` в директории `~/baikal/src/examples/baremetal/build`.
5. Для проведения тестирования на БФК необходимо загрузить файл пригодный для совместного использования с `dfu-util`, с помощью которой происходит обновление программного обеспечения. Он находится в директории `~/baikal/src/examples/baremetal/build/simple.bin`
6. Пример вывода будет находиться в файле `trace` в директории откуда запускался QEMU.

5.2 Bare-metal приложение с периферией

В данном разделе мы разработаем программу, использующую периферийное устройство процессора «Байкал-Т1», в данном случае UART. Для этого выполним следующие действия:

1. Создаем папку `~/baikal/src/examples/baremetal_periphery` и копируем в нее исходные файлы из архива `baremetal_periphery.tar`

```
$ mkdir ~/baikal/src/examples/baremetal_periphery
$ cd ~/baikal/src/examples/baremetal_periphery
$ tar xvf baremetal_periphery.tar
```

2. Воспользоваться утилитой `make` для создания необходимых файлов.
3. При необходимости внести изменение в путях до средств кросс-компиляции в `makefile`:

```
. . .
CROSS_COMPILE = ~/baikal/usr/x-tools/mipsel-unknown-linux-gnu/bin/mipsel-unknown-linux-gnu-
. . .
```

4. Или с помощью export внести собственное значение в переменную CROSS_COMPILE.

```
$ export CROSS_COMPILE=~/baikal/usr/x-tools/mipsel-unknown-linux-gnu/bin/mipsel-unknown-linux-gnu-
```

5. Утилита make создаст объектные файлы в директории `obj`, файлы для загрузки в `qemu` в директории `build`.
6. Для проведения тестирования на БФК необходимо загрузить файл пригодный для совместного использования с `dfu-util`, с помощью которой происходит обновление программного обеспечения. Он находится в директории

```
~/baikal/src/examples/bare_metal_periphery/build/uart.bin
```

7. Необходимо выполнить в консоли следующий текст:

```
sudo ./dfu-util -D ../src/examples/bare_metal_periphery/build/uart.bin -d abf0:1234 -a 0
```

8. После чего необходимо заранее открыть консоль

```
$ picocom –b 115200 /dev/ttyConsole
```

9. Далее перезагрузить плату, и наблюдать результат аналогичный выводу QEMU,

10. Вышеуказанный пример можно проверить также на симуляторе QEMU, результат будет представлен в директории, откуда запускался симулятор. Команда для запуска QEMU(путь может требовать коррекции):

```
    $ ~/baikal/bin/qemu-system-mipsel -bios build/uartbin.bin -D trace -d asm -
singlestep -machine baikal-t -net none -icount 0 -vnc none -serial stdio
```

В консоль будут выведены результаты тестирования

```
    $ ~/baikal/bin/qemu-system-mipsel -bios build/uartbin.bin -D trace -d asm -
singlestep -machine baikal-t -net none -icount 0 -vnc none -serial stdio
    qemu-system-mipsel: Bin boot was loaded '/srv/tftp/uart.bin'
    Warning: nic dwgmac.0 has no peer
    Warning: nic dwgmac.1 has no peer
    Warning: nic dwxgmac.0 has no peer
     Test uart0, base addr=0xbf04a000
     Set UART loopback mode for uart0.
     Test for UART#0 PASSED
     Clear UART loopback mode for uart0.
     Test uart1, base addr=0xbf04b000
     Set UART loopback mode for uart1.
     Test for UART#1 PASSED
     Clear UART loopback mode for uart1.
```