Использование эмулятора QEMU для программирования процессора Baikal-T1

Редактор: **Михаил Бессонов**
Доступ: **Публичный**

История изменений

Версия	Дата	Исполнитель	Описание
1.0	27.11.2016	Т. Яловая	Выполнен перенос раздела из документа «Комплект средств разработки ПО для микропроцессора Baikal-T1» в новый документ.

Содержание

Список таблиц

1 Работа с симулятором. Запуск программ, трассировка и отладка

Функциональный симулятор системы Baikal-T1 реализован на основе программного симулятора QEMU. Он позволяет загружать образ ядра Linux или любую bare-metal программу, собранную в elf-файл. После запуска симулятор позволяет общаться с системой посредством UART при помощи терминальной программы.

Наличие блока контроллера SATA предоставляет возможность создавать и использовать пользовательские образы дисковых устройств (реализованных в виде файла образа диска).

Симулятор QEMU имеет быстродействие, достаточное для комфортной работы, при этом эффективная частота достигает десятка МГц. Так как QEMU является функциональным симулятором, он позволяет отлаживать пользовательские и системные программы, но не обеспечивает оценку производительности программ.

1.1 Основные опции симулятора

1.1.1 Опция -singlestep

Симулятор может работать в двух режимах:

- Режим динамической (бинарной) трансляции;
- Режим интерпретации (пошаговый режим).

В режиме динамической трансляции симулятор последовательно декодирует и транслирует в код хост-платформы линейные участки бинарного target-кода. Эти транслированные участки сохраняются, и в случае передачи кода на оттранслированный и сохраненный участок повторного декодирования и трансляции не происходит. За счет этого достигается большая производительность. Однако для некоторых целей, например, для получения трассировки исполняемой программы, необходим режим интерпретатора, в котором инструкции target-платформы транслируются и интерпретируются в пошаговом режиме. Режим интерпретатора устанавливается при помощи опции командной строки `-singlestep`.

1.1.2 Опция -icount <N>

Моделируемые блоки периферии имеют собственные временные периоды (например,

таймеры). Эти периоды могут быть выбраны относительно произвольно по отношению к частоте процессора, т.е. количество выполненных инструкций процессора в течение одного периода таймера будет различным. Кроме того, при моделировании важна повторяемость выполнения групп инструкций от запуска к запуску.

Повторяемость достигается заданием опции `-icount <N>` (нет синхронизации таймеров с внешним миром). Число инструкций, выполняемых процессором за 1 нс, будет $1/2^N$. Таким образом, при `-icount 0` процессор будет работать с эффективной частотой (по отношению к фиксированным частотам периферии), равной 1 ГГц.

Влияние данной опции на работу программ наглядно демонстрируется процедурой калибровки задержки при загрузке ядра ОС Linux:

Для icount=0:

```
Calibrating delay loop... 0.77 BogoMIPS (lpj=3865)
```

Для icount=2:

```
Calibrating delay loop... 0.02 BogoMIPS (lpj=120)
```

Изменение значения <N> меняет динамику прихода внешних прерываний, что даёт очень эффективный способ поиска гонок в выполняемых программах.

Для реализации синхронного режима работы, при котором внутренние временные интервалы выбираются в соответствии с реальным временем хост-машины, используется опция `-icount auto`. В данном режиме работа операционной системы в эмуляторе будет наиболее близка к работе в реальном времени. Использование данного режима допустимо только для эмуляции однопроцессорной системы. При реализации многопроцессорной системы работа в синхронном режиме не может быть гарантирована.

1.1.3 Опции -bios <file_name> и -kernel <file_name>

Для запуска симулятора необходимо указать основной исполняемый файл. В наиболее реальной ситуации исполняемой программой является образ начального загрузчика. Начальный загрузчик может иметь один из двух форматов:

- Бинарный файл (последовательность машинных команд);
- Формат elf32.

Бинарный файл размером не более 4 Мб будет загружен, начиная с физического адреса

`0x1fc000000`.

Файл формата elf32 будет загружен по физическим адресам, связанными с виртуальными адресами секций следующим соотношением:

```
physaddr = virtaddr & 0x1fffffff
```

Оператор «&» является побитовой операцией «И». Так стартовый вектор, имеющий виртуальный адрес `0xbfc00000`, транслируется в физический адрес `0x1fc00000`. Это соответствует преобразованию адресов сегмента kseg1 в физические адреса.

Вместо загрузчика может быть выполнена любая другая bare-metal программа. Файл начального загрузчика передается симулятору с использованием опции `-bios <file_name>`.

Одновременно с начальным загрузчиком или вместо него может быть выполнен любой elf32 файл, например, файл с образом операционной системы. Это позволяет ускорить загрузку за счет исключения дополнительного копирования ядра ОС и корневой файловой системы загрузчиком с блочного устройства в память. Файл образа операционной системы должен иметь формат elf32. Образ загружается опцией `-kernel <file_name>`.

Файл формата elf32 будет загружен по физическим адресам, связанными с виртуальными адресами секций следующим соотношением:

```
physaddr = virtaddr & 0x7fffffff
```

Это соответствует преобразованию адресов из сегмента kseg0 в физические адреса.

Допускается использование общего образа, содержащего начальный загрузчик, ядро ОС и образ файловой системы в одном файле.

При загрузке образа посредством опции `-kernel` так же доступны опции `-append <cmd line>` и `-initrd <initrd file>`, с помощью которых можно передать в симулятор параметры ядра ОС (например, корневой раздел на блочном устройстве) и initrd соответственно.

1.1.4 Опция -machine <platform name>

Данная опция определяет целевую платформу (тип процессора и периферию), которую будет моделировать симулятор. Модель Baikal-T1 задается значением `-machine baikal-t`.

1.2 Примеры использования симулятора для отладки разрабатываемого ПО

Машина, на которой происходит моделирование является, хост-машиной. Машина, модель которой реализует симулятор, является target-машиной. В поставляемом пакете BSP предполагается использование машины архитектуры x86 для запуска симулятора (хост-машина) и платформы Baikal-T1 с архитектурой mips32 в качестве target-машины.
В дальнейшем символы $ или # будут использоваться для обозначения приглашения интерпретатора командной строки хост-машины, а > – для обозначения приглашения интерпретатора командной строки target-машины (терминал QEMU).

1.2.1 Трассировка выполнения bare-metal программ

Для изучения возможностей симулятора будет использована простейшая bare-metal программа на ассемблере (файл `<Baikal SDK>/src/examples/test.S`):

```
$ cat test.s
.set noat
.global _start
.text
_start:
li  $1, 1
li  $2, 2
add $3, $1, $2
wait
nop
```

Компиляция теста:

```
$ mipsel-unknown-linux-gnu-gcc test.S -c -g -o test.o -Wall
```

Для сборки теста понадобится линковочный скрипт файл `<Baikal SDK>/src/examples/test.lds`:

```
$ cat test.lds
OUTPUT_FORMAT("elf32-tradlittlemips")
TARGET(binary)
```

```
SECTIONS
{
      . = 0xBFC00000,
      .text : {
            "test.o" (.text)
      }
      .rodata : {
            *(.rodata)
      }
      .data : {
            *(.data)
      }
      .bss : {
            *(.bss)
      }
}
```

Здесь адрес `0xBFC00000` – это стартовый виртуальный адрес, который транслируется в физический адрес стартового вектора `0x1FC00000`.

Далее следует собрать программу `test` при помощи редактора связей (линковщика):

```
$ <Baikal_SDK>/usr/x-tools/mipsel-unknown-linux-gnu/bin/mipsel-unknown-linux-gnu-ld  --script=test.lds -o test
```

Ниже приведен дизассемблер полученного в результате исполняемого файла `test`:

```
$ <Baikal_SDK>usr/x-tools/mipsel-unknown-linux-gnu/bin/mipsel-unknown-linux-gnu-objdump -Dz test
test:     file format elf32-tradlittlemips

Disassembly of section .text:

bfc00000:   24010001    li    at,1
bfc00004:   24020002    li    v0,2
```

```
bfc00008:   00221820    add   v1,at,v0
bfc0000c:   42000020    wait
bfc00010:   00000000    nop
bfc00014:   00000000    nop
bfc00018:   00000000    nop
bfc0001c:   00000000    nop
...
```

Для анализа трассировка выполнения перенаправляется в файл `trace`.

Далее необходимо запустить программу на qemu-system-mipsel, загружая получившийся файл test в boot:

```
$ <Baikal_SDK>/bin/qemu-system-mipsel -bios test -D trace -d asm
-singlestep -machine baikal-t -net none -icount 0 -vnc none
```

Опция `-d asm` позволяет получить дизассемблер выполняемого кода. Затем выполняется остановка моделирования (например, с помощью комбинации клавиш <Ctrl+C>) и производится анализ трассировки из файла с данными:

```
$ cat trace
#0 0xbfc00000:  li  at,1
#0 0xbfc00004:  li  v0,2
#0 0xbfc00008:  add v1,at,v0
#0 0xbfc0000c:  wait
```

Результат совпадает с кодом вышеприведенной простейшей программы test.S.

Также можно производить трассировку, показывающую изменения состояния процессора, для этого следует запустить моделирование с трассировкой `-d exec,cpu`:

```
$ <Baikal_SDK>/bin/qemu-system-mipsel -bios test -D trace -d
exec,cpu -singlestep -machine baikal-t -net none -icount 0 -vnc
none
```

Важная опция для получения последовательной трассы выполнения – это `-singlestep`.

Выполняется остановка моделирования (например, с помощью комбинации клавиш <Ctrl+C>) и производится анализ трассировки из файла с данными:

```
$ cat trace
```

```
Trace #0 0x40b1d000 [bfc00000]
#0 pc=0xbfc00000 HI=0x00000000 LO=0x00000000 ds 0090 00000000 0
GPR00: r0 00000000 at 00000000 v0 00000000 v1 00000000 a0 00000000
a1 00000000 a2 00000000 a3 00000000
GPR08: t0 00000000 t1 00000000 t2 00000000 t3 00000000 t4 00000000
t5 00000000 t6 00000000 t7 00000000
GPR16: s0 00000000 s1 00000000 s2 00000000 s3 00000000 s4 00000000
s5 00000000 s6 00000000 s7 00000000
GPR24: t8 00000000 t9 00000000 k0 00000000 k1 00000000 gp 00000000
sp 00000000 s8 00000000 ra 00000000
CP0: Status [00400004] Cause [00000000] EPC [00000000]
Config0 [80020482] Config1 [fee3718b] LLAddr [00000000]
Trace #0 0x40b1d080 [bfc00004]
#0 pc=0xbfc00004 HI=0x00000000 LO=0x00000000 ds 0090 00000000 0
GPR00: r0 00000000 at 00000001 v0 00000000 v1 00000000 a0 00000000
a1 00000000 a2 00000000 a3 00000000
GPR08: t0 00000000 t1 00000000 t2 00000000 t3 00000000 t4 00000000
t5 00000000 t6 00000000 t7 00000000
GPR16: s0 00000000 s1 00000000 s2 00000000 s3 00000000 s4 00000000
s5 00000000 s6 00000000 s7 00000000
GPR24: t8 00000000 t9 00000000 k0 00000000 k1 00000000 gp 00000000
sp 00000000 s8 00000000 ra 00000000
CP0: Status [00400004] Cause [00000000] EPC [00000000]
Config0 [80020482] Config1 [fee3718b] LLAddr [00000000]
Trace #0 0x40b1d100 [bfc00008]
#0 pc=0xbfc00008 HI=0x00000000 LO=0x00000000 ds 0090 00000000 0
GPR00: r0 00000000 at 00000001 v0 00000002 v1 00000000 a0 00000000
a1 00000000 a2 00000000 a3 00000000
GPR08: t0 00000000 t1 00000000 t2 00000000 t3 00000000 t4 00000000
t5 00000000 t6 00000000 t7 00000000
```

```
GPR16: s0 00000000 s1 00000000 s2 00000000 s3 00000000 s4 00000000
s5 00000000 s6 00000000 s7 00000000
GPR24: t8 00000000 t9 00000000 k0 00000000 k1 00000000 gp 00000000
sp 00000000 s8 00000000 ra 00000000
CP0: Status [00400004] Cause [00000000] EPC [00000000]
Config0 [80020482] Config1 [fee3718b] LLAddr [00000000]
Trace #0 0x40b1d1c0 [bfc0000c]
#0 pc=0xbfc0000c HI=0x00000000 LO=0x00000000 ds 0090 00000000 0
GPR00: r0 00000000 at 00000001 v0 00000002 v1 00000003 a0 00000000
a1 00000000 a2 00000000 a3 00000000
GPR08: t0 00000000 t1 00000000 t2 00000000 t3 00000000 t4 00000000
t5 00000000 t6 00000000 t7 00000000
GPR16: s0 00000000 s1 00000000 s2 00000000 s3 00000000 s4 00000000
s5 00000000 s6 00000000 s7 00000000
GPR24: t8 00000000 t9 00000000 k0 00000000 k1 00000000 gp 00000000
sp 00000000 s8 00000000 ra 00000000
CP0: Status [00400004] Cause [00000000] EPC [00000000]
Config0 [80020482] Config1 [fee3718b] LLAddr [00000000]
```

Данные трассировки демонстрируют, как изменяется состояние процессора (опция `-d cpu`: регистры общего назначения, статусные регистры), в зависимости от исполняемой команды (опция `-d exec`). Изменения точно соответствуют исполняемому коду программы.

1.2.2 Отладка bare-metal программы с помощью gdb

Прежде всего необходимо запустить на симуляторе qemu-system-mipsel тестовый пример, загружая получившийся файл test в boot:

```
$ <Baikal_SDK>/bin/qemu-system-mipsel -bios test -machine baikal-t
-net none -icount 0 -vnc none -gdb tcp::1234 -S
```

Важные опции для использования gdb это: `-gdb tcp::1234 -S`. `1234` обозначает порт сервера gdb. Также при компиляции программы в объектный файл нужно указать ключ `-g` для поддержки отладочной информации.

После старта симулятор остается в остановленном состоянии, ожидая подключения клиента gdb.

Необходимо запустить клиентскую часть gdb:

```
$ mipsel-unknown-linux-gnu-gdb
```

После старта указать исполняемую программу:

```
(gdb) file test
```

Указать желаемую архитектуру:

```
(gdb) set arch mips
```

После чего следует присоединиться к симулятору:

```
(gdb) target remote localhost:1234
_start () at test.s:5
5       li  $1, 1
```

Далее можно производить пошаговое выполнение и отладку:

```
(gdb) stepi
6       li  $2, 2
(gdb) stepi
7       add $3, $1, $2
(gdb) info registers
      zero     at     v0     v1     a0     a1     a2         a3
 R0   00000000 00000001 00000002 00000000 00000000 00000000
00000000 00000000
             t0         t1         t2         t3         t4         t5
t6         t7
 R8   00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000
             s0         s1         s2         s3         s4         s5
s6         s7
 R16  00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000
             t8         t9         k0         k1         gp         sp
```

```
s8       ra
 R24  00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000
            sr       lo       hi      bad    cause       pc
      00400004 00000000 00000000 00000000 00000000 bfc00008
           fsr      fir
      00000000 00739500
```

1.3 Использование симулятора для запуска ОС Linux

1.3.1 Запуск ОС Linux на симуляторе

Для упрощения запуска образа ОС Linux на симуляторе пакет BSP содержит скрипты в директории `<Baikal SDK>/usr/scripts`. Скрипты позволяют производить запуск симулятора с подключенным или отключенным образом жесткого диска. Однако в ряде случаев может потребоваться прямой запуск симулятора (например, для запуска со средствами отладки).

Пример строки запуска образа ОС на qemu-system-mipsel с диском приведен ниже:

```
$ <Baikal_SDK>/bin/qemu-system-mipsel -kernel
<Baikal_SDK>/img/linux-mipsel-baikal.elf -net none -icount 0 -vnc
none -nographic  -drive if=none,file=disk.img,format=raw,id=drive0
-device ide-drive,bus=ide.0,drive=drive0,id=sata0 -machine
baikal-t
```

Важной опцией при запуске является `-machine baikal-t`, которая создает target-машину, являющуюся моделью Baikal-T1.

В процессе загрузки ядра ОС Linux на консоль будут выводится системные сообщения. В случае успешной загрузки пользователю будет предоставлено приглашение интерпретатора командной строки с правами суперпользователя. Работа в терминале симулятора не отличается от работы на удаленном компьютере.

1.3.2 Трассировка прерываний ОС Linux на симуляторе

При исполнении ядра Linux трассировка с использованием дизассемблера и изменения

состояния регистров приводит к достаточно большим трассам. Как показывает практика, часто встречающаяся ошибка связана с потерей прерывания. Находить такие ошибки помогает трассировка прерываний. Помимо этого, она может использоваться и для разбора других ошибок. Для получения трассировки прерываний нужно запустить моделирование с ключом `-d int,gic`. Дополнительный ключ `-d gic` предоставляет информацию об функционировании контроллера прерываний. Пример запуска:

```
$ <Baikal_SDK>/bin/qemu-system-mipsel -kernel
<Baikal_SDK>/img/linux-mipsel-baikal.elf -net none -icount 0 -vnc
none -nographic  -drive if=none,file=disk.img,format=raw,id=drive0
-device ide-drive,bus=ide.0,drive=drive0,id=sata0 -machine baikal-
t -D trace -d int,gic
```

Участок такой трассировки реальной загрузки ОС Linux приведен ниже:

```
#0 wait pc 80105600
gic: set shared irq #10 level 1
gic(#0): eic RIPL prio 33
#0 Set interrupt
#0 pc 80105600 ei Status 11000000 => 11000001
#0 Exception 'interrupt' enter [80105604]
EPC [80105604]
Status [11000003]
Cause [10808400]
BadVAddr [00623000]
DEPC [00000000]
#0 pc 80104094 Status 11000003 (00) => 11000000 (00) Cause
10808400
gic: read 0x480
gic: read 0x400
gic: read 0x484
gic: read 0x404
gic: read 0x488
```

```
gic: read 0x408
#0 pc 8016ba54 di Status 11000000 => 11000000
#0 pc 8016baac Status 11000000 (00) => 11000000 (00) Cause
10808400
gic: read 0x14
gic: read 0x10
gic: read 0x14
#0 pc 80177f78 di Status 11000000 => 11000000
#0 pc 8016e55c di Status 11000000 => 11000000
gic: read 0x14
gic: read 0x10
gic: read 0x14
#0 pc 8016e684 Status 11000000 (00) => 11000000 (00) Cause
10808400
#0 pc 80177f98 Status 11000000 (00) => 11000000 (00) Cause
10808400
gic: set shared irq #10 level 0
gic(#0): eic RIPL prio 0
#0 Reset interrupt
```

Вышеприведенный участок начинает с команды `wait`. Процессор прекращает выполнение этой команды по прерыванию от таймера (см. таблицу 2). Ниже идет обработка прерывания, заканчивающаяся сбросом уровня сигнала прерывания на процессоре (сообщение в трассе `Reset interrupt`).

Также, как можно видеть выше, в трассе отслеживаются команды, способные повлиять на маскировку внешних прерываний процессором: `mtc0 $12, 0, di, ei`.

1.3.3 Трассировка обращений в IO ОС Linux на симуляторе

Одной из необходимых возможностей при отладке работы драйверов устройств является отслеживание обращений в пространство ввода-вывода (IO). Для этих целей в симуляторе существует трассировка таких обращений по опции `-d iomem`. Пример запуска

```
$ <Baikal_SDK>/bin/qemu-system-mipsel -kernel
<Baikal_SDK>/img/linux-mipsel-baikal.elf -net none -icount 0 -vnc
none -nographic  -drive if=none,file=disk.img,format=raw,id=drive0
-device ide-drive,bus=ide.0,drive=drive0,id=sata0 -machine baikal-
t -D trace -d iomem
```

Ниже приведён участок трассы, содержащей инициализацию и обращения в UART0 (см. таблицу 1):

```
IO write [0x1f04a00c/4] 0x80
IO write [0x1f04a000/4] 0xd
IO write [0x1f04a00c/4] 0x3
IO write [0x1f04a008/4] 0x71
IO read  [0x1f04a014/1] 0x60
IO write [0x1f04a000/1] 0x42
IO read  [0x1f04a014/1] 0x60
IO write [0x1f04a000/1] 0x61
IO read  [0x1f04a014/1] 0x60
IO write [0x1f04a000/1] 0x69
IO read  [0x1f04a014/1] 0x60
IO write [0x1f04a000/1] 0x6b
IO read  [0x1f04a014/1] 0x60
```

1.4 Запуск предустановленного образа операционной системы (ОС Debian) в симуляторе QEMU

Пакет программного обеспечения Baikal-T1 BSP содержит предустановленный образ операционной системы Debian версий 6 (Squeeze) и 7 (Wheezy). Также, начиная с версии SDK 2.3.4, пакет включает образ Debian из ветки testing (Jessie). Для запуска системы необходимо использовать прекомпилированный образ ядра ОС Linux `<Baikal SDK>/img/linux-debian-mipsel.elf` или собственный образ ядра ОС, способный производить загрузку с жесткого диска.

При использовании сети в симуляторе его запуск должен производиться с правами

суперпользователя (для обеспечения доступа к сетевому интерфейсу). Аналогично симулятор может быть запущен при помощи скрипта `run-qemu-mipsel.sh`:

```
$ sudo run-qemu-mipsel.sh -img linux-debian-mipsel.elf \
-sata1 debian-wheezy-mipsel.img
```

Образ диска с ОС Debian может быть примонтирован в файловую систему хост-машины аналогично любым образам жесткого диска при помощи утилиты `disk-tool.sh`:

```
# disk-tool.sh mount debian-wheezy-mipsel.img /mnt
```

1.5 Запуск операционной системы в режиме эмуляции на хост-машине

При использовании симулятора в пользовательском режиме на хост-машинах с современными дистрибутивами операционных систем (Debian, Fedora) существует возможность запуска операционной системы без использования системного симулятора. При этом исполняемые программы пользовательского режима, в том числе и интерпретатор командной строки, будет транслироваться через симулятор. Системные вызовы будут перенаправлены к текущему ядру хост-машины, и таким образом будут доступны все модули текущего ядра и периферийные устройства, в том числе и сетевой стек. Данный режим может быть использован для упрощения отладки приложений, не использующих специфическую периферию SoC Baikal-T1.

Для реализации данного метода необходимо иметь образ операционной системы для архитектуры MIPS32, развернутый в определенной локально директории (`<Baikal RootFS>`). В качестве базовой системы могут быть использованы предустановленные образы дистрибутивы ОС Debian, поставляемые в комплекте BSP, смонтированные в локальную директорию. Необходимо скопировать файл симулятора пользовательского режима в директорию `/usr/bin` развернутого образа и переименовать его в `qemu-mipsel-static`:

```
# sudo cp <Baikal SDK>/bin/qemu-mipsel
<Baikal RootFS>/usr/bin/qemu-mipsel-static
```

После этого необходимо переключится в режим симуляции на хост-машине:

```
# sudo chroot <Baikal RootFS>
```

После выполнения данных действий система будет работать в режиме архитектуры MIPS32:

```
# uname -m
mips

# uname -a
Linux host.baikal.int 3.13.0-40-generic #69~precise1-Ubuntu SMP
Fri Nov 14 10:29:31 UTC 2014 mips GNU/Linux
```

Дальнейшее использование операционной системы аналогично использованию выбранного дистрибутива на хост-машине. Необходимые пакеты могут быть собраны посредством компилятора или установлены посредством пакетного менеджера (для ОС Debian: `apt-get install / remove`)

1.6 Настройка сетевых интерфейсов и использование сети в симуляторе.

Для интеграции с сетевой инфраструктурой хост-машины симулятор использует TAP устройство. Для обеспечения функционирования данного устройства совместно с сетевым интерфейсом хост-машины необходима перенастройка сетевых интерфейсов и создание бриджа. Для управления интерфейсами необходимо, чтобы ядро операционной системы хоста поддерживало создание бриджевых интерфейсов и необходимо наличие утилиты `tunctl` из пакета `uml-utilities`.

Для создания необходимой сетевой инфраструктуры следует вручную исправить сетевые настройки. Для большинства современных дистрибутивов необходимо заменить в файле /etc/network/interfaces секции, касающиеся настроек текущих сетевых интерфейсов следующим образом:

Исходные сетевые настройки:

```
# cat /etc/network/interfaces
auto lo
iface lo inet loopback
auto eth0
```

```
iface eth0 inet dhcp
```

Сетевые настройки с бриджем:

```
auto lo
iface lo inet loopback
auto eth0
iface eth0 inet dhcp
auto br0
iface br0 inet dhcp
pre-up tunctl -t tap0 -g tuntap
pre-up ip link set tap0 up
bridge_ports eth0 tap0
bridge_stp off
bridge_maxwait 0
bridge_fd 0
post-down ip link set tap0 down
post-down tunctl -d tap0
```

После изменения настроек необходимо перезапустить сервис управления сетью:

```
# service networking restart
```

В случае, если хост-машина использует Network Manager (Debian, Ubuntu и аналогичные дистрибутивы), потребуется перезагрузка системы.

После изменения настроек в системе появится сетевой интерфейс бриджа BR0 и новый сетевой интерфейс TAP0.

При запуске симулятора необходимо указать при помощи параметра `-net` созданный сетевой TAP интерфейс. В общем случае для приведенного примера настройки сети к общим параметрам запуска добавится следующие параметры:

```
-netdev tap,id=virtnet1,ifname=tap0
-net nic,netdev=virtnet1,model=dwgmac
```

При использовании скриптов автоматического запуска симулятора `run-qemu-mipsel.sh` необходимо указать соответствие интерфейсов симулятора и TAP интерфейсов системы, при этом запуск данного скрипта должен производиться с правами суперпользователя (для

обеспечения доступа к сетевому интерфейсу):

```
$ sudo run-qemu-mipsel.sh -img linux.elf \
-sata1 debian-wheezy-mipsel.img \
-gmac1 tap=tap0
```

2 Таблицы адресов и прерываний процессора Baikal-T1

Таблицы адресов и прерываний приведены в соответствующих таблицах ниже.

Таблица 1: Адреса процессора Baikal-T1

Устройство	***Начальный адрес***	***Конечный адрес***	***Размер***
Low Memory DRAM Block	0x0000_0000	0x07FF_FFFF	128 MB
PCI Express mapped area	0x0800_0000	0x1BDB_FFFF	256 МБ + 61.75 MB
P5600 GIC[1]	0x1BDC_0000	0x1BDD_FFFF	128 KB
P5600 CPC	0x1BDE_0000	0x1BDE_7FFF	32 KB
Reserved	0x1BDE_8000	0x1BF7_FFFF	1.6 MB
Internal SRAM	0x1BF8_0000	0x1BF8_FFFF	64 KB
Reserved	0x1BF9_0000	0x1BFB_FFFF	192 KB
Internal ROM	0x1BFC_0000	0x1BFC_FFFF	64 KB
Reserved	0x1BFD_0000	0x1BFF_FFFF	192 KB
SPI Flash ROM[2]	0x1C00_0000	0x1CFF_FFFF	16 MB
Reserved	0x1D00_0000	0x1F03_FFFF	16.3 MB
Boot Controller	0x1F04_0000	0x1F04_0FFF	4 KB
DMA Cfg & Ch Regs	0x1F04_1000	0x1F04_1FFF	4 KB
DDR Ctrl	0x1F04_2000	0x1F04_2FFF	4 KB
DDR Phy	0x1F04_3000	0x1F04_3FFF	4 KB
GPIO	0x1F04_4000	0x1F04_4FFF	4 KB
Control GPIO	0x1F04_5000	0x1F04_5FFF	4 KB
I2C_0	0x1F04_6000	0x1F04_6FFF	4 KB
I2C_1	0x1F04_7000	0x1F04_7FFF	4 KB
Reserved	0x1F04_8000	0x1F04_8FFF	4 KB
Timer 1-3	0x1F04_9000	0x1F04_9FFF	4 KB
UART 0	0x1F04_A000	0x1F04_AFFF	4 KB

1 The area is accessible from the p5600 core and JTAG only (and not accessible by DMA)
2 The actual space in the SPI FLASH is 32MB. The Memory Mapped is however limited with the 3-byte addressing scheme (16 MB)

Устройство	***Начальный адрес***	***Конечный адрес***	***Размер***
UART 1	0x1F04_B000	0x1F04_BFFF	4 KB
WDT	0x1F04_C000	0x1F04_CFFF	4 KB
PMU + PMU_I2C	0x1F04_D000	0x1F04_DFFF	4 KB
SPI_0	0x1F04_E000	0x1F04_EFFF	4 KB
SPI_1	0x1F04_F000	0x1F04_FFFF	4 KB
SATA	0x1F05_0000	0x1F05_1FFF	8 KB
PCIe Cfg Regs	0x1F05_2000	0x1F05_2FFF	4 KB
PCIe DMA	0x1F05_3000	0x1F05_3FFF	4 KB
Eth XGMAC	0x1F05_4000	0x1F05_7FFF	16 KB
Reserved	0x1F05_8000	0x1F05_8FFF	4 KB
APB Terminator	0x1F05_9000	0x1F05_9FFF	4 KB
Main Interconnect	0x1F05_A000	0x1F05_AFFF	4 KB
Reserved	0x1F05_B000	0x1F05_CFFF	8 KB
Eth XGMAC XPCS	0x1F05_D000	0x1F05_DFFF	4 KB
1GEth_0 GMAC	0x1F05_E000	0x1F05_FFFF	8 KB
1GEth_1 GMAC	0x1F06_0000	0x1F06_1FFF	8 KB
Reserved	0x1F06_2000	0x1F0F_FFFF	
USB 2.0 Ctrl	0x1F10_0000	0x1F1F_FFFF	1 MB
Reserved	0x1F20_0000	0x1FBF_7FFF	
P5600 Global Control Block	0x1FBF_8000	0x1FBF_9FFF	8 KB
P5600 Core-Local Block	0x1FBF_A000	0x1FBF_BFFF	8 KB
P5600 Core-Other Block	0x1FBF_C000	0x1FBF_DFFF	8 KB
P5600 Debug Block	0x1FBF_E000	0x1FBF_FFFF	8 KB
Boot Block Area (Mirrored)	0x1FC0_0000	0x1FFF_FFFF	4 MB
Hi Memory DRAM Block	0x2000_0000	0xFFFF_FFFF	3.5 GB

Таблица 2: Прерывания процессора Baikal-T1

Устройство	***Номер прерывания в GIC***	***Номер аппаратного прерывания***
GIC Watchdog	1	0 (локальный)
GIC Timer	2	1 (локальный)
CPU Timer	3	2 (локальный)
Зарезервированы для IPI	7—22	0—15 (разделённый)
Блок обработки ошибок шины APB	23	16 (разделённый)
WDT	24	17 (разделённый)
GPIO	26	19 (разделённый)
Timer 1	31	24 (разделённый)
Timer 2	32	25 (разделённый)
Timer 3	33	26 (разделённый)
I2C 0	40	33 (разделённый)
I2C 1	41	34 (разделённый)
SPI 0	47	40 (разделённый)
SPI 1	48	41 (разделённый)
UART 0	55	48 (разделённый)
UART 1	56	49 (разделённый)
SATA	71	64 (разделённый)
ETH GMAC 0	79	73 (разделённый)
ETH GMAC 1	80	74 (разделённый)
ETH XGMAC	87	80 (разделённый)
Блок обработки ошибок шины AXI	134	127 (разделённый)

3 О компании «Байкал Электроникс»

Компания «Байкал Электроникс» создана в 2012 году. Мы специализируемся на проектировании интегральных микросхем и систем на кристалле на базе архитектур ARM и MIPS. Наши продукты предназначены для использования в энергоэффективных компьютерных и промышленных системах с разным уровнем производительности и функциональности. Мы оказываем поддержку нашим клиентам-разработчикам, обеспечивая сокращение времени и затрат на создание конечных изделий на базе микропроцессоров семейства «Байкал» в условиях жёсткой конкуренции.

Контакты:
143421, Московская область, Красногорский район, 26 км автодороги «Балтия», бизнес-центр RigaLand, блок Б, 2-й этаж
телефон: (495) 221-39-47
http://www.baikalelectronics.ru

