Обзор SDK ARM64 (версия 5.3)

История изменений

Версия	Дата	Описание
1.0	16.12.2021	Начальная версия

Содержание

Введение

SDK (Software Development Kit) ARM64 предназначен для создания сборок образов и различных модулей (таких как образ файловой системы, UEFI образ, модули ядра VDec или ядра Linux с определенной конфигурацией), а так же файловых систем с дополнительными пакетами утилит для процессоров Байкал-М.
Работа с SDK производится через терминал или командную строку ОС Linux.

Данный документ содержит описание процесса установки (раздел Установка SDK), краткое описание структуры распакованных файлов и папок (раздел Структура SDK), доступных после установки SDK, а также описание работы с SDK (раздел Работа с SDK).

1 Установка SDK

Установка производится на компьютер с операционной системой Linux (Debian системы и их производные).

Для установки SDK необходимо выполнить следующие действия:

1. Из раздела “Среда разработки” на сайте скачать с архив с последней версией SDK (на данный момент – 5.3) на локальный диск.
2. Распаковать архив `SDK-M-5.3.zip`.
3. В папке `SDK-M-5.3` найти файл с именем `baikal-arm64-5.3-20210924.run`.
4. Открыть в терминале Linux папку `SDK-M-5.3`, выполнить команду
   ```
   chmod +x ./baikal-arm64-5.3-20210924.run
   ```
   для разрешения исполнения установочного файла как программы.
5. Для запуска установки в терминале необходимо выполнить следующую команду:
   ```
   ./baikal-arm64-5.3-20210924.run
   ```

Внимание: Процесс установки может занять несколько минут. Не закрывайте терминал до окончания процесса установки.

Распакованный SDK будет находиться в директории `baikal` рядом с установщиком.

2 Структура SDK

SDK содержит следующее дерево каталогов:

- `_build` – временная директория для хранения сборок, создается автоматически после первой сборки
- `img` – временная директория для хранения образов, создается автоматически после первой сборки
- `prebuilts` – готовые (собранные заранее) загрузочные образы платы, предоставляемые вместе с SDK
- `src` – исходные файлы для компонентов SDK:
  - `acpica` – исходные файлы ACPICA[1](необходимы для построения IASL[2] для UEFI[3])
  - `arm-tf` – дерево исходных файлов доверенной прошивки ARM (TF-A)
  - `initrd` – дерево исходных файлов rootfs для Baikal Embedded Linux (BEL)
  - `kernel` – дерево исходных файлов ядра Linux
  - `mali` – исходные файлы драйвера и бинарные библиотеки Arm® Mali™-T628
  - `openocd` – исходные файлы OpenOCD
  - `uefi` – исходное дерево загрузчика UEFI
  - `vdec` – исходные файлы драйвера видеодекодера и бинарные библиотеки
- `tools` – основные скрипты SDK
- `utils` – дополнительные утилиты (например, `fs utils`, `dtc`)
- `xtools` – кросс-инструментальная цепочка ARM64 с компилятором, GDB и `glibc`

[1] Advanced Configuration and Power Interface Component Architecture
[2] Intel ACPI Source Language compiler/decompiler
[3] Unified Extensible Firmware Interface

Примечание: Каталог `prebuilts` содержит два важных образа прошивки SCP[4], которые невозможно собрать из исходников: `dbm.scp.flash.bin` и `mixt.scp.flash.bin` для плат DBM и MBM1.0/MBM2.0 соответственно.

3 Работа с SDK

Для установки библиотек, необходимых для работы SDK, в терминале Linux нужно выполнить следующую команду:

```
sudo apt-get install autoconf automake bison build-essential
clang flex fonts-freefont-ttf genext2fs imagemagick
libarchive-zip-perl libfdt-dev libglib2.0-dev libncurses-dev
libpixman-1-dev libssl-dev m4 python uuid-dev xxd zlib1g-dev
```

Для создания и сборки образов в SDK используется два скрипта:

- Основной скрипт сборки – `build-boot-img.sh`
- Скрипт сборки образа файловой системы – `build-initrd-img.sh`

3.1 Основной скрипт сборки

Для получения краткой справки для основного скрипта необходимо запустить терминал в директории SDK (в папке `baikal`, которая генерируется после завершения установки) и выполнить следующую команду:

```
./tools/build-boot-img.sh -h
```

На экран выводится основная информация скрипта: шаблон команды для сборки, список поддерживаемых плат, а так же список параметров запуска сборки скрипта. Шаблон команды для сборки выглядит следующим образом:

```
build-boot-img.sh <target> <options>
```

, где `<target>` – тип платы, для которой будет происходить сборка;

`<options>` – параметры, с которыми мы запускаем процесс сборки.

Для параметра `<target>` может использоваться одно из следующих возможных значений:

- `dbm` – отладочная плата DBM
- `mbm10` – материнская плата MBM1.0
- `mbm20` – материнская плата MBM2.0

В качестве `<options>` можно использовать комбинацию следующих параметров:

- `-c` или `--clean` – очистить предыдущие сборки (ядро, UEFI, TF-A)
- `-d` или `--defconfig` – собрать ядро .config из defconfig
- `-f` или `--dtb` – собрать большие двоичные объекты дерева устройств (DTB)
- `-k` или `--kernel` – собрать ядро Linux с текущей конфигурацией (.config)
- `-m` или `--modules` – собрать модули ядра с текущей конфигурацией (.config)
- `-i` или `--initrd` – собрать образ файловой системы initrd
- `-D` или `--vdec` – собрать модули ядра vdec (релиз)
- `-u` или `--uefi` – собрать UEFI (релиз)
- `-ud` или `--uefi-debug` – собрать UEFI (отладка)
- `-t` или `--tfa` – собрать TF-A (релиз)

[4] System control Processor

- `-td` или `--tfa-debug` – собрать TF-A (отладка)
- `-b` или `--bootrom` – собрать BootROM
- `-bl` или `--bootrom-linux` – собрать BootROM с образом восстановления Linux
- `-bt` или `--bootrom-truncate` – собрать BootROM и усечь 32 МБ
- `-h` или `--help` – отобразить справку

Для удобства, в качестве `<options>` можно использовать следующие комбинированные параметры:

- `-a` – применить все параметры, эквивалентно следующему списку параметров:
  ```
  --clean --defconfig --dtb --kernel --modules --initrd
  --vdec --uefi --tfa --bootrom
  ```
- `-ad` или `--all-debug` – применить все параметры (отладка), эквивалентно следующему списку параметров:
  ```
  --clean --defconfig --dtb --kernel --modules --initrd
  --vdec --uefi-debug --tfa-debug --bootrom
  ```

Для сборки полного набора образов и других бинарных файлов в терминале необходимо ввести одну из следующих команд:

- для DBM:
  ```
  ./tools/build-boot-img.sh dbm -a
  ```
- для MBM1.0:
  ```
  ./tools/build-boot-img.sh mbm10 -a
  ```
- для MBM2.0:
  ```
  ./tools/build-boot-img.sh mbm20 -a
  ```

Скрипт сохраняет созданные образы в директории `baikal/img`, как описано в разделе Результаты выполнения скриптов.

Так же в SDK присутствует возможность собрать ядро Linux с включенными параметрами отладки. По умолчанию, конфигурация ядра оптимизирована для повышения производительности. Включение параметров отладки может привести к незначительному снижению. производительности.

Чтобы собрать ядро Linux с включенными параметрами отладки, необходимо в терминале выполнить следующие две команды:

1. команду копирования содержимого файла `baikal_defconfig_debug` в файл `baikal_defconfig`:
   ```
   cp src/kernel/arch/arm64/configs/baikal_defconfig{_debug,}
   ```
2. команду сборки ядра и модулей ядра Linux с текущей конфигурацией, например для платы MBM2.0:
   ```
   ./tools/build-boot-img.sh mbm20 -d -k -m
   ```

Чтобы собрать ядро Linux с отключенными параметрами отладки, необходимо в терминале выполнить следующие две команды:

1. команду копирования содержимого файла `baikal_defconfig_release` в файл `baikal_defconfig`:
   ```
   cp src/kernel/arch/arm64/configs/baikal_defconfig{_release,}
   ```
2. команду сборки ядра и модулей ядра Linux с текущей конфигурацией, например для платы MBM2.0:
   ```
   ./tools/build-boot-img.sh mbm20 -d -k -m
   ```

3.2 Скрипт сборки образа файловой системы

Для получения краткой справки по данному скрипту, необходимо запустить терминал в директории SDK и выполнить следующую команду:

```
./tools/build-initrd-img.sh -h
```

Данная команда выводит на экран краткую справку с шаблоном команды и параметрами запуска. Шаблон команды для сборки выглядит следующим образом:

```
build-initrd-img.sh <platform> <options> <-p | --packages list>
```

, где `<platform>` – платформа для сборки (необязательное поле);

`<options>` – параметры запуска сборки файловой системы;

`<-p | --packages list>` – параметр запуска для установки дополнительных пакетов.

Для параметра `<platform>` может использоваться одно единственное значение:

- `baikal`

В качестве `<options>` можно использовать комбинацию следующих параметров:

- `-c` или `--clean` – очистить временную директорию (`img/tmp`) хранения сборок
- `-s` или `--skipbuild` – пропустить сборку, создать образ из существующего каталога
- `-m` или `--modules` – собрать и установить модули ядра
- `-d` или `--default` – собрать и установить пакеты по умолчанию (см. ниже)
- `-p` или `--packages` – установить дополнительные пакеты (перечисляются в виде списка после параметров в командной строке)
- `-l` или `--list` – вывести список доступных пакетов
- `-h` или `--help` – вывести справку

В параметре `<-p | --packages list>`, через пробел, необходимо указать список дополнительных пакетов, например:

```
build-initrd-img.sh -p benchmarks i2ctools
```

Скрипт поддерживает установку следующих пакетов:

- `benchmarks` – набор тестов производительности
- `busybox` – набор крошечных версий распространенных утилит UNIX в одном небольшом исполняемом файле
- `dropbear` – небольшой SSH-сервер и клиент
- `e2fsprogs` – набор служебных утилит для проверки целостности файловых систем, поиска и исправления ошибок, изменения настроек, форматирования
- `ethtool` – утилита для проверки и настройки сетевого интерфейса Ethernet
- `fbtest` – утилита для тестирования frame buffer
- `fio` – утилита для тестирования производительности жестких дисков
- `i2ctools` – набор утилит I2C для ядра Linux
- `iperf3` – утилита измерения пропускной способности сети TCP, UDP и SCTP
- `iperf` – утилита для измерения пропускной способности сети
- `kexec-tools` – утилита, позволяющая загрузить новое ядро «поверх» уже запущенного
- `lmsensors` – утилита для мониторинга за состоянием аппаратного обеспечения

- `pciutils` – утилиты, использующие библиотеку для переносимого доступа к регистрам конфигурации шины PCI и несколько утилит, основанных на этой библиотеке
- `spitools` – утилита командной строки, для использования устройства Linux Spidev

В случае если `<-p | --packages list>` не указан, но указан параметр `-d`, будет установлены следующие пакеты: `busybox`, `i2ctools`, `lmsensors`, `ethtool`, `dropbear`, `kexec-tools` и `pciutils`.

3.3 Результаты выполнения скриптов

После успешного выполнения скрипта, в каталоге `img` будут расположены следующие файлы:

- `img/${BRD_NAME}.Image` – образ ядра Linux
- `img/${BRD_NAME}.Image.gz` – сжатый образ ядра Linux
- `img/${BRD_NAME}.System.map` – файл System.map ядра Linux
- `img/${BRD_NAME}.bl1.bin` – бинарный файл, необходимый для записи в `boot_flash`
- `img/${BRD_NAME}.dtb` – BLOB[5] объект дерева устройств для выбранного типа платы
- `img/${BRD_NAME}.efi.fd` – образ прошивки EFI[6]
- `img/${BRD_NAME}.fat.img` – образ FAT с Baikal Embedded Linux (для создания такого образа необходимо собранное ядро, файловая система и в терминале нужно вызвать `./tools/build-boot-img.sh dbm -bl`)
- `img/${BRD_NAME}.fip.bin` – FIP[7] образ (BLs и EFI)
- `img/dbm.flash0.img` – только для плат DBM: образ `system_flash` (прошивка SCP и BL1)
- `img/dbm.flash.img` – только для плат DBM: образ `boot_flash` (DT, BLs и EFI)
- `img/${BRD_NAME}.flash.img` – только для MBM1.0 или 2.0: неполный флэш-образ (DT, BLs и EFI)
- `img/${BRD_NAME}.full.img` – только для MBM1.0 или 2.0: полный флэш-образ (Прошивка SCP, DT, BLs и EFI)
- `img/initrd.gz` – образ файловой системы Baikal Embedded Linux
- `img/img_mem.ko` – модуль ядра Linux для поддержки VDec
- `img/vxd.ko` – модуль ядра Linux для поддержки VDec

В названии файлов `${BRD_NAME}` – это `dbm`, `mbm10` или `mbm20` для плат DBM, MBM1.0 и MBM2.0 соответственно.

[5] Binary Large OBject
[6] Extensible Firmware Interface
[7] Firmware Image Package