МЦСТ Волга

Elbrus Board Support Package

Руководство программиста

Содержание

Введение

В документе даётся описание процессу установки, настройки и использованию компонентов LBSP, включающих в себя:

1. Загрузчик операционной системы – `BOOT`
2. Ядро и модули ядра
3. Средства разработки и отладки программ – `e2k toolchain`
4. Средства для работы с `EtherCAT` – промышленной шины, основанной на Ethernet
5. Прочие пакеты, входящие в состав дистрибутива Linux

Все приведённые инструкции описываются для для ВК `Монокуб-PC` (Рисунок 1), являющегося компактным и недорогим персональным компьютером на базе микропроцессора `Эльбрус-2С+`.

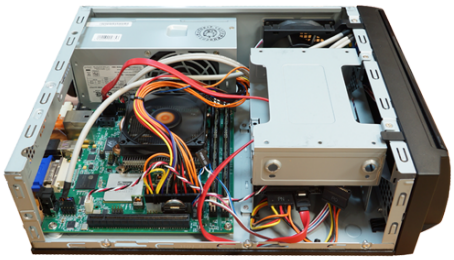

Рис. 1: Монокуб-PC

На материнской плате `Монокуб-PC` реализован стандартный набор периферийных интерфейсов: Gigabit Ethernet, SATA 2.0, IDE, USB 2.0, RS-232. Благодаря наличию независимых видеовыходов (VGA и DVI), возможно оснащение АРМ оператора двумя мониторами. Имеется слот PCI-Express x16 (используются 8 линий), позволяющий устанавливать карты расширения. Интерфейс IDE подведён к разъёму карты CompactFlash (на тыльной стороне материнской платы). На материнскую плату выведено 6 каналов (8-контактов) программируемого интерфейса ввода/вывода (GPIO), что может быть использовано для приема и генерирования внешних сигналов управления.

В статье подразумевается, что для разработки используется i386 машина с установленным Debian Wheezy 7. Инструкции по получению и установке ОС находятся на сайте проекта Debian [en].

Перед началом работы необходимо настроить подключение к локальным deb репозиториям и системе контроля версий git. Описание процесса настройки можно найти на сайте компании или в разделе Поддержка.

Программные компоненты LBSP

Загрузчик

BOOT — программа, которая начинает работать при включении питания платы. Назначение программы — инициализация процессора, памяти и южного моста. Далее `BOOT` загружает с диска исполняемый код ядра ОС, размещает его в оперативной памяти и передает управление коду ядра ОС.

Ядро ОС и модули ядра

Ядро ОС — программа, расположенная на загрузочном разделе, получающая управление от бута после того как бут загрузил исполняемый код этой программы в оперативную память. Ядро ОС работает в привилегированном режиме, это значит, что оно может выполнять все без исключения команды из системы команд процессора Эльбрус-2с+. Ядро инициализирует все компоненты системы, включая виртуальную память, локальные и внешние прерывания, память устройств, подсистему планирования процессов, сетевую и файловую системы, запускает процесс init и передает ему управление. После инициализации код ядра остается в оперативной памяти. В процессе работы приложения ядро выполняет обработку исключительных ситуаций, прерываний и сигналов, обеспечивает работу системных вызовов, а также обеспечивает работу драйверов устройств.

Модули ядра — программы для взаимодействия с шинами у устройствами. С точки зрения исходных текстов модули являются частью ядра ОС. Исполняемый код модулей хранится на диске. Модули загружаются (попадают в оперативную память и инициализируются) из процесса init, которому управление передается от ядра при старте системы. Модули работают в привилегированном режиме (как и ядро).

Средства разработки и отладки

Система программирования (toolchain) — набор программ для компиляции, сборки (линковки) и отладки всех компонент ОС, а также приложения. В набор входят: компилятор, линковщик, отладчик, утилиты binutils. Система программирования содержит в себе как toolchain для сборки на инструментальной машине целевой архитектуры — native режим, так и кросс-toolchain, то есть набор инструментов для выполнения компиляции и сборки компонентов ОС и приложения на архитектуры x86. Кросс-toolchain генерирует код для целевой архитектуры (e2k).

Взаимодействие программных компонент

Процесс инициализации встраиваемого вычислительного комплекса (ВК)

Процесс init - процесс, исполняемый в пользовательском режиме (в отличие от привилегированного режима, для программ, выполняемых в пользовательском режиме, недоступен ряд команд). Init загружает необходимые модули ядра, инициализирует сетевой интерфейс, выполняет запуск фоновых сервисов, и передает управление приложению или ФПО.

Пример прикладной программы, реализующей бесконечный цикл ФПО.

ФПО — Функциональное программное обеспечение, — боевая программа.

Листинг 1: Пример программы ФПО /fkmtests/fpo/example.c

```
#include <linux/watchdog.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/ioctl.h>
#include <signal.h>

/* Error codes */
#define E_OK    0
#define E_WD    (-1)

/* Watchgog global variables: */
static char *wd_dev_name = "/dev/watchdog";
static int wd_fd = 0;
static int timeout = 10;

/* kill fpo signal handler */
void signal_callback_handler(int signum) {
    if (signum == SIGTERM) {
        printf("FPO killed by SIGTERM, watchdog is not stopped
        - to be rebooted in %d seconds\n", timeout);
        sleep(timeout);
    }
} /* signal_callback_handler */

/* function setupWatchdog, returns 0 on success */
int setupWatchdog(void) {
    /* Open watchdog device file */
    wd_fd = open(wd_dev_name, O_WRONLY);
```

```
    if (wd_fd < 0){
        printf("\nCan not open watchdog device file %s\n", wd_dev_name);
        return E_WD;
    }
    /* Set timeout */
    if (ioctl(wd_fd, WDIOC_SETTIMEOUT, &timeout) < 0){
        printf("\nioctl WDIOC_SETTIMEOUT failed\n");
        return E_WD;
    }

    /* Register signal and handler */
    signal(SIGTERM, signal_callback_handler);
    return E_OK;
}

/* Entry point to FPO */
int main(void) {
    int status;
    printf("Setting up watchdog\n");
    status = setupWatchdog();
    if (status != E_OK) {
        printf("Error on setup watchdog, status= %d", status);
        return E_WD;
    }

    printf("Enter the main loop of FPO\n");
    while(1)
    {
        if (ioctl(wd_fd, WDIOC_KEEPALIVE, 0) < 0) {
        printf("\nioctl WDIOC_KEEPALIVE failed\n");
    }
        /* do the job here */
        sleep(timeout / 2);
    }
    /* Never here */
    return E_OK;
}
```

Установка системы

В данной секции рассмотрен процесс установки и первоначальной настройки дистрибутива LBSP 5.0 на инструментальную машину.

Получение образа

Установочный образ системы может быть загружен по следующей ссылке: `http://212.59.102.250/opensource/heap/instrumental/e2k-400.iso`. Приведенная ниже инструкция по установке предполагает, что образ после скачивания был записан на DVD-диск.

Установка программных компонент на пустой ВК

Для установки ОС следует выполнить следующие действия:

1. В случае использования внешнего USB-привода DVD-дисков: подключить привод к ВК.
2. Включить ВК.
3. Вставить в устройство чтения CD/DVD загрузочный диск.
4. Произвести перезапуск ВК кнопкой `<Reset>` на лицевой панели корпуса ВК, если кнопка `<Reset>` отсутсвует, то необходимо выключить питание ВК и включить его снова.
5. Дождаться появления сообщений загрузчика, и нажатием клавиши `SPACE` отменить возможную автозагрузку.
6. Нажатием клавиши `d` определить, каким номером определилось устройство `ATAPI device`.
7. Войти в режим редактирования клавишей `c` и задать параметры загрузки, как указано ниже:
   drive_number – номер устройства загрузки `ATAPI device`;
   partition_number - раздел загрузки: 0 ;
   command_string - командная строка ядра: console=tty0 consoleblank=0 ;
   filename: deb.img;
   initrd: deb.rd;
8. Инициировать загрузку клавишей `p`.
9. Оставить остальные параметры по умолчанию, посредством нажатия клавиши `Enter`.
10. Для продолжения работы следовать диалогу. Процедура выбора параметров установки имеет интуитивно-понятный интерфейс.

Первоначальная настройка ВК

Настройка сети

Для установки проводного соединения потребуется задать статический IP-адрес для интерфейса `eth0`, например:

```
ifconfig eth0 192.168.1.10 up
```

Если в сети используется DHCP, необходимо установить DHCP-клиент из репозиториев (см. ниже):

```
apt-get install dhcpcd
```

После чего можно подключиться к `eth0`, используя DHCP:

```
dhcpcd eth0
```

Подключение репозиториев

Для подключения дополнительных репозиториев потребуется отредактировать файл `/etc/apt/sources.list`, добавив в него следующие строки:

```
deb http://212.59.102.250/apt lenny main
deb http://212.59.102.250/apt lenny-updates updates
```

Далее следует обновить список репозиториев при помощи следующей команды:

```
apt-get update
```

Может потребоваться импортировать ключи репозиториев, используя следующую команду:

```
gpg --keyserver subkeys.pgp.net --recv-keys KEY
```

Управление пакетами

Управление пакетами осуществляется при помощи пакетного менеджера `apt`. Для получения дополнительной информации обратитесь к `man apt`.

Компиляция и отладка программ в cross-режиме

Введение

Для использования кросс-компиляции под платформу «Эльбрус» потребуется следующий набор ПО:

- **e2k cross toolchain** - кросс-компилятор со всеми необходимыми ему кросс-библиотеками и кросс-отладчик
- **crossroot** - дерево библиотек и заголовочных файлов, получаемое с целевой машины, для которой планируется выполнять кросс-сборку и кросс-отладку.

Предполагается, что для разработки используется Debian 7.8.0 Wheezy.

Установка e2k cross toolchain

Cross toolchain включает в себя следующие компоненты:

- Кросс-компилятор для архитектуры e2k
- Библиотеки нижнего уровня
- Отладчик gdb
- Трассировщик системных вызовов

Cross toolchain распространяется в виде бинарных deb пакетов для архитектуры i386.
Установка cross toolchain осуществляется при помощи следующих команд:

Листинг 2: Установка cross toolchain

```
i386# wget http://212.59.100.18/opensource-heap/instrumental/toolchains/spo-19/\
     cross/lcc-e2k-cross_1.1_i386.deb
i386# dpkg -i lcc-e2k-cross_1.1_i386.deb
```

После успешного завершения установки на машине i386 появляется каталог `opt/mcst` и каталоги `bin`, `bin.toolchain` и `gdb` внутри него.

Установка crossroot

Crossroot можно получить двумя способами:

1. Загрузить с сайта архив, содержащий crossroot.
2. Подготовить самостоятельно, используя ВК с установленной системой.

Листинг 3: Загрузка образа crossroot

```
i386# cd /opt/mcst
i386# wget http://212.59.100.18/opensource-heap/instrumental/toolchains/spo-19/\
    cross/crossroot_e2c+_911.tar.gz
i386# tar xvf crossroot_e2c+_911.tar.gz
```

Самостоятельное получение crossroot Чтобы самостоятельно получить crossroot на монокубе необходимо воспользоваться следующими инструкциями:

Листинг 4: Самостоятельное получение Crossroot

```
i386# ssh root@target_machine
  cd /
  mkdir crossroot
  cd crossroot
  cp -a /lib32 .
  cp -a /lib64 .
  ln -s lib64 lib
  mkdir usr
  cp -a /usr/lib32 usr/
  cp -a /usr/lib64 usr/
  ln -s usr/lib64 usr/lib
  cp -a /usr/libexec usr/
  cp -a /usr/include usr/
  cp -a /usr/local usr/
  sync
  cd ..
  tar -cvf crossroot_selfmade.tar ./crossroot
  gzip -9 ./crossroot_selfmade.tar
```

Ядро ОС и модули ядра

Сборка ядра и модулей ОС

Описывается два варианта сборки ядра и модулей ОС. Сборка производится на инструментальной машине.
Первый вариант - сборка в режиме native - предполагает использование в качестве инструментальной машины монокуб (Эльбрус-2с+) с установленным на нем дистрибутивом Debian 5.0 Lenny (LBSP 5.0) и системой программирования.
Второй вариант - сборка в cross-режиме - предполагает использование в качестве инструментальной машины x86 с установленными на ней дистрибутивом Debian 7.8 Wheezy или выше и средствами кросс-разработки.

Сборка ядра и модулей ОС в режиме native

Для выполнения сборки ядра и модулей ОС на машине Монокуб (Эльбрус-2с+) необходимо выполнить следующие шаги:

1. Получение исходных текстов ядра и модулей ОС из репозитория:

   ```
   git clone mcst-volga-git:kernel-3_14
   ```

2. Переход в каталог с текстами ядра и конфигурирование:

   ```
   cd kernel-3_14
   make defconfig
   ```

3. Запуск сборки ядра:

   ```
   make bootimage
   ```

   Процесс сборки займет 3 часа. В результате сборки в текущем каталоге появится файл `image.boot`.

   ```
   ls -la image.boot
   -rwxr-xr-x 1 root root 30387044 2014-12-04 22:15 image.boot
   ```

4. Запуск сборки модулей:

   ```
   make modules
   ```

   Процесс занимает порядка 4-х часов.

5. Установка собранных модулей в заданную директорию

   ```
   make modules_install INSTALL_MOD_PATH=<dest_dir>
   ```

   Для установки собранного ядра и модулей необходимо перенести ядро `image.boot` в каталог `/boot` диска, каталог с модулями `<dest_dir>` - в корневой каталог диска.

Сборка ядра и модулей ОС в режиме cross

Сборка ядра и модулей ОС в режиме cross предусмотрена для использования при отладке. В cross режиме ядро и модули ОС собираются на машине x86 с использованием cross-компилятора, что значительно быстрее чем в native режиме.

Для выполения сборки в cross-режиме необходимо выполнить следующие шаги:

1. Установка cross инструментов сборки (компилятор):

```
wget http://212.59.100.18/opensource-heap/instrumental/toolchains/spo-19/\
       cross/lcc-e2k-cross_1.1_i386.deb
dpkg -i lcc-e2k-cross_1.1_i386.deb
```

2. Получение исходных текстов ядра и модулей:

```
git clone mcst-volga-git:kernel_3-14
```

3. Устанавливаем переменной окружения `ARCH`:

```
export ARCH=e2k
```

4. Конфигурирование ядра:

```
make defconfig
```

   Эта команда приводит к тому, что конфигурация из файла `arch/e2k/defconfig` попадает в .config.

5. Сборка ядра и модулей:

```
make bootimage -j8 CC=/opt/mcst/bin/lcc &&
make modules -j8 CC=/opt/mcst/bin/lcc
```

6. Установка собранных модулей в заданную директорию:

```
make modules_install INSTALL_MOD_PATH=<dest_dir>
```

Сборка собственных модулей в cross режиме

Рассмотрим пример сборки собственного модуля в режиме cross. Действия производятся в дереве исходных текстов ядра.

1. Добавим собственный модуль, разместив его в `drivers/misc/hello.c`:

Листинг 5: drivers/misc/hello.c

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

static int __init hello_init(void)
```

```
{
        printk(KERN_DEBUG "Hello, World!\n");
        return 0;
}
static void __exit hello_exit(void)
{
        printk(KERN_DEBUG "Goodbye, World!\n");
}
module_init(hello_init);
module_exit(hello_exit);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("void");
MODULE_DESCRIPTION("hello");
```

2. Добавим строку в `drivers/misc/Makefile`:

   ```
   obj-$(CONFIG_HELLO)       += hello.o
   ```

3. Добавим описание модуля в `drivers/misc/Kconfig`:

   ```
   config HELLO
          tristate "Hello world example"
          help
            <help text>
   ```

4. Для того что бы модуль был включен в сборку необходимо в файле `arch/e2k/defconfig` добавить строку:

   ```
   CONFIG_HELLO=y
   ```

   И переконфигурировать ядро:

   ```
   make defconfig
   ```

5. После чего нужно запустить сборку модулей:

   ```
   make modules CC=/opt/mcst/bin/lcc
   ```

   Что приведет к сборке добавленного модуля - `hello.ko`.
   Для установки собранного ядра и модулей на машину целевой архитектуры необходимо перенести ядро `image.boot` в каталог `/boot`, каталог с модулями `<dest_dir>` - в каталог `/lib/modules/<название ядра>`.

EtherCAT

Введение

EtherCAT — стандарт промышленной сети, относимый к семейству Industrial Ethernet и технологиям используемым для распределенного управления в режиме реального времени. Дейтаграммы EtherCAT пропускаются внутри стандартного фрейма Ethernet.

Существует несколько свободных реализаций EtherCAT Master для Linux. IgH EtherCAT Master отличается от прочих решений возможностью работы в режиме жесткого реального времени, поддержкой большого числа сетевого оборудования и гибкостью. Ниже приведены инструкции для настройки IgH EtherCAT Master.

Сборка IgH EtherCAT Master

Установка необходимых пакетов

Перед установкой следует убедиться в наличии следующих пакетов:

```
apt-get install libtool autoconf make gcc build-essential
```

Получение исходных текстов

Скачать последнюю версию IgH EtherCAT Master можно при помощи git:

```
git clone mcst-volga-git:ethercat-master
```

Сборка IgH EtherCAT Master

В данном разделе описаны два способа сборки IgH EtherCAT Master: в cross и native режимах.

Сборка в режиме cross

В директории с исходным кодом IgH EtherCAT Master выполнить:

```
./bootstrap
```

Скрипт для сборки в кросс режиме выглядит следующим образом:

Листинг 6: Скрипт для кросскомпиляции EtherCAT Master

```
#!/bin/bash
export ARCH=e2k
export CC="/opt/mcst/bin.toolchain/e2k-linux-gcc"
export CXX="/opt/mcst/bin.toolchain/e2k-linux-g++"
export CPP="/opt/mcst/bin.toolchain/e2k-linux-cpp"
export CFLAGS='-I/opt/mcst/crossfs-3.0-rc5.e2k-8c/usr/include/'
export LDFLAGS='-L/opt/mcst/crossfs-3.0-rc5.e2k-8c/usr/lib64/'
export LIBS='-ldl -lc'
export CROSS_COMPILE='/opt/mcst/bin.toolchain/e2k-linux-'
```

```
export CROSS_ROOT="/opt/mcst/crossfs-3.0-rc5.e2k-8c"
export INSTALL_MOD_PATH="/path/to/lib/modules/3.14.46-elbrus-rt46-ga398952-dirty/

./configure --with-linux-dir=/path/to/linux-build \
--enable-e1000e \
--host=e2k \
--build=x86_64-linux \
--disable-8139too \
--prefix=/path/to/etherlab \
CC=/opt/mcst/bin.toolchain/e2k-linux-gcc \
CXX=/opt/mcst/bin.toolchain/e2k-linux-g++ \
CPP=/opt/mcst/bin.toolchain/e2k-linux-cpp \
CFLAGS='-I/opt/mcst/crossfs-3.0-rc5.e2k-8c/usr/include/' \
LDFLAGS='-L/opt/mcst/crossfs-3.0-rc5.e2k-8c/usr/lib64/' \
LIBS='-ldl -lc' \
CROSS_COMPILE='/opt/mcst/bin.toolchain/e2k-linux-'

make && make modules && make modules_install
```

Скрипт `.configure` достаёт информацию о системе из директории с исходниками ядра (параметр –with-linux-dir), поэтому при кросскомпиляции потребуется e2k-ядро, собранное на той же машине. Также потребуются модули ядра, собранные в режиме кросскомпиляции - EtherCAT установит свои модули в указанную директорию.

Далее можно передать собранные EC-модули, сервис `etc/init.d/ethercat`, конфигурационный файл и утилиту `ethercat` на e2k машину:

```
pushd /path/to/kernel/modules && tar cvf --atime-preserve modules.tar lib
scp modules.tar user@host:/home/user
scp -r /path/to/ethercat-hg/script root@host:/opt/etherlab
scp /path/to/ethercat-hg/tool/ethercat root@host:/usr/local/sbin/ethercat
```

На e2k машине потребуется распаковать модули ядра:

```
su -c 'tar xvf /home/komarov/modules.tar'
```

Сборка в режиме native

```
./bootstrap
```

Для вызова *configure* потребуются исходники ядра, сконфигурированные на e2k-машине:

```
./configure --enable-e1000e --build=e2k --with-linux-dir=/path/to/linux/sources
```

Далее:

```
make -j5
make modules -j5
make module_install
make install
```

Модули будут установлены в /lib/modules для версии ядра, указанной параметром `--with-linux-dir`. По умолчанию установка компонентов EC проводится в `/opt/etherlab`.

Настройка EtherCAT Master

```
ln -s /opt/etherlab/etc/init.d/ethercat /etc/init.d/ethercat
ln -s /opt/etherlab/etc/sysconfig/ethercat /etc/sysconfig/ethercat
```

Для первоначальной настройки ethercat потребуется отредактировать конфигурационный файл /etc/sysconfig/ethercat, указав MAC-адрес мастер-устройства и драйвера, используемые EC.

Например:

```
MASTER0_DEVICE="08:00:30:03:80:00"
DEVICE_MODULES="e1000e"
```

Ethercat по умолчанию не работает от обычного пользователя, поэтому нужно создать правило udev `/etc/udev/rules.d/99-EtherCAT.rules` со следующим содержимым:

```
KERNEL=="EtherCAT[0-9]", MODE="0666"
```

При необходимости в правила можно дописать группы пользователей, у которых будет доступ к `ethercat`:

```
KERNEL=="EtherCAT[0-9]", MODE="0666" GROUP="users"
```

Использование EC Master

Запуск EtherCAT Master

Запуск демона EtherCAT Master осуществляется с помощью следующей команды:

```
/etc/init.d/ethercat start
```

При запуске демон `ethercat` подгрузит необходимые модули ядра и создаст файл устройства `/dev/EtherCATx`.

Утилита ethercat

По умолчанию устанавливается в `/opt/etherlab/bin/ethercat`; в случае кросскомпиляции может быть найдена в исходниках в директории tool.

Проверить работу EC Master можно следующей командой:

```
ethercat master
```

Userspace library

Для работы с EtherCAT в пользовательских приложениях используется библиотека libethercat, по умолчанию расположенная в директории `/opt/etherlab/lib`. Примеры использования можно найти в официальной документации к EC Master и в каталоге с исходниками: `src/time_benchmark.c`.

Поддержка

Контакты

Сайт	http://mcst-volga.ru/
Телефон	+7 495 589-27-15
	+7 916 415-16-85
E-mail	support@mcst-volga.ru

Репозитории deb пакетов

В открытом доступе находятся репозитории, содержащие полный набор пакетов из Debian Lenny, собранных под архитектуру `e2k` и обновленные версии пакетов, используемых при разработке:

```
deb http://212.102.59.250/apt lenny main
deb http://212.102.59.250/apt lenny-updates main
```

Более подробную информацию о подключении и использовании репозиториев можно найти в `man 8 apt`, `man 5 sources.list`, либо в официальной документации Debian [en].

Репозитории git

Репозитории git доступны по адресу `http://212.10.259.250/gitweb`. Для получения доступа свяжитесь с нами одним из вышеуказанных способов.

Для работы с git потребуется настроить SSH на клиентской машине. Пример конфигурационного файла `~/.ssh/config` приведён ниже.

Листинг 7: /.ssh/config

```
Host mcst-volga-git
User gitolite3
Hostname 212.10.259.250
Port 22
IdentityFile /path/to/.ssh/git_private_key
```

Больше информации о конфигурации SSH и содержании файла `~/.ssh/config` можно найти в `man 1 ssh`.

После настройки SSH, можно работать с git. Посмотреть список доступных репозиториев и права доступа к ним можно с помощью следующей команды:

```
ssh mcst-volga-git info
```

Листинг 8: Пример: вывод команды ssh info

```
 R W    ec-virtual-slave
 R W    ethercat-master
 R      kernel-3_14
 R W    lbsp
 R      robotics-library
 R      robotics-library-demos
```

Для работы с git следует использовать следующий синтаксис:

```
git clone mcst-volga-git:repo_name
```

Для подробного описания работы с git обратитесь к `man 1 git`, либо к официальной документации [en].