MCST Volga

Elbrus Board Support Package

Developer guide

Table of contents

Introduction

The current document describes installation, configuration and usage of following LBSP components:

1. Bootloader – `BOOT`
2. Kernel and kernel modules
3. Development and debugging tools – `e2k toolchain`
4. Other LBSP packages

We assume that the following instructions is carried out using `Monocub-PC` (Picture 1). `Monocub-PC` is a compact and inexpensive personal computer based on `Elbrus-2C+` microprocessor.

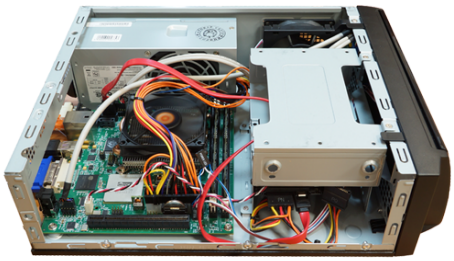

Figure 1: Monocub-PC

`Monocub-PC` provides following set of peripheral interfaces: Gigabit Ethernet, SATA 2.0, IDE, USB 2.0, RS-232. The workstations based on `Monocub-PC` can be used with two screens via two independent video outputs: VGA and DVI. It also has PCI-Express x16 slot that allows place the additional cards. The CompactFlash card slot connected with IDE interface on the back side of motherboard. The motherboard also provides 6 channels of GPIO interface.

In this document we assume that development is carried out using Debian Wheezy 7.8 on i386 host. The installation of debian system is describes in debian project's website.

Before starting make sure you connected to out debian repos and VCS git. Following steps desribed at our website and in Support section.

LBSP software components

Bootloader

BOOT is a program that starts when you turn on the power. `BOOT` initializes CPU, memory, and south bridge, takes a kernel executrable from the disc, puts it into memory and transfers control to the kernel code.

Kernel and kernel modules

Kernel is a program, located on the boot partition. It obtains control from the boot. OS kernel runs in the privileged mode. The kernel initializes all the system components, including virtual memory, local and external interrupts, memory devices, subsystem planning processes, and the network subsystem, starts the init process, and passes control to it. Kernel provides interrupt and exception handling, system calls and device drivers.

Kernel modules are software to communicate with devices on the system bus. From the viewpoint of source modules are part of the kernel. The executable code modules stored on disk. The modules are loaded (enter into memory and initialized) from the process init, which control is transferred from the kernel at system startup. The modules operate in the privileged mode (as kernel).

Development and debugging tools

Toolchain is a set of programs to compile, build (linking) and debug of all OS components. Toolchain contains: a compiler, linker, debugger, binutils. Toolchain is available for both native (Elbrus) and cross modes. In cross mode cross-toolchain runs on the i386 architecture and generates code for target architecture(Elbrus).

The interaction of software components

Initialization process

Init process – a process running in user mode. Init loads necessary kernel modules, initializes the network interfaces and starts program services.

An example of an application that uses watchdog

Listing 1: An example of an application that uses watchdog

```
#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;
}
```

System installation

In this section described installation and initial configuration process for LBSP 5.0 distribution.

Getting image

The installation system image can be downloaded by the following link: `http://212.59.102.250/opensource/heap/instrumental/e2k-400.iso`.

Installing software components

The following operations must be performed to install the OS:

1. When using an external USB-drive: Connect the drive to the computer.
2. Turn the power on.
3. Insert the CD / DVD bootable disk into the drive.
4. Perform a restart.
5. Wait for the boot loader messages, and pressing the Space key to cancel the possible startup.
6. Press the `d` key to determine the number of `ATAPI device`.
7. Press button `c` and set the boot parameters, as follows:
   drive_number – number defined `ATAPI device`;
   partition_number - 0 ;
   command_string – console=tty0 consoleblank=0 ;
   filename: deb.img;
   initrd: deb.rd;
8. Initiate the download by `p` button.
9. Leave the rest of the default settings by pressing `Enter` key.
10. To continue follow the dialogue. The procedure for selecting parameter setting has an intuitive interface.

Initial setup

Network configuration

To install a wired connection a static IP-address for the `eth0` interface should be set, for example:

```
ifconfig eth0 192.168.1.10 up
```

DHCP can be configured after installing dhcpcd in the following way (this can be done after connecting repositories, see below):

```
apt-get install dhcpcd
dhcpcd eth0
```

Connecting repositories

To connect additional repositories one should edit `/etc/apt/sources.list` file by the adding the following lines:

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

Next, the repository list should be updated using the following command:

```
apt-get update
```

If repository key is needed it can be imported in the following way:

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

Package management

Package management is carried out using `apt`. For more information, refer to the `man apt`.

Compiling and debugging programs in cross-mode

Introduction

To use cross-platform compilation for «Elbrus» the following set of software is to be installed:

- **e2k cross toolchain** - cross-compiler with it's internal cross-libraries and cross-debugger.
- **crossroot** - binaries of libraries and header files obtained with the target machine, for which it is planned to carry out cross-building and cross-debugging.

We assume that the development is carried out using Debian 7.8.0 Wheezy.

Installing Elbrus(e2k) cross toolchain

Cross toolchain includes the following components:

- Cross-compiler for architecture e2k
- Low level libraries
- gdb debugger
- System calls tracer

Cross toolchain is distributed as a binary deb package for the i386 architecture.
To install cross tollchain perform the following commands:

Listing 2: Cross toolchain installation

```
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
```

Directory `/opt/mcst` with bin, `bin.toolchain` and `gdb` directories inside are created on i386 machine when installation is completed successfully.

Installing crossroot

Crossroot can be obtained in two ways:

1. Download prebuild crossroot.
2. Prepare crossroot manually using installed e2k machine.

Listing 3: Getting crossroot image

```
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
```

Prepare crossroot manually To prepare crossroot manually the following commands should be issued:

Listing 4: Prepare crossroot manually

```
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
```

Building the kernel and kernel modules

Introduction

This chapter describes two ways of building kernel and kernel modules: native and cross building.
Native – to build in native mode – is to use the monocub machine (Elbrus-2c +) with LBSP 5.0 installed.
Cross – to build in cross-mode – is to use the x86 machine with Debian 7.8 Wheezy and cross–toolchain (compiler) installed.

Building the kernel and kernel modules in native mode

To perform the building of the kernel and kernel modules on the monocub machine (Elbrus-2c +) the following steps should be done:

1. Get the kernel source from the repository:

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

2. Go to the directory with the kernel source and configure kernel:

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

3. Run kernel build:

   ```
   make bootimage
   ```

   The building process will take about 3 hours. File `image.boot` will be created in the current directory:

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

4. Run modules build:

   ```
   make modules
   ```

   The process takes about 4 hours.

5. Install the built modules to the specified directory:

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

   Next step you should move kernel `image.boot` and kernel modules into `/boot` and `/lib/modules/<kernel version` directories on the target machine correspondingly.

Building the kernel and modules in cross mode

Building the kernel and modules in cross mode is intended for use in debugging. In this mode build is carried out on x86 machine using a cross-compiler, which is much faster than in native mode.

To perform the building in cross-mode, follow these steps:

1. Install cross-toolchain:

```
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. Get the source kernel and modules:

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

3. Setting environment variables `ARCH`:

```
export ARCH=e2k
```

4. Configure the kernel:

```
make defconfig
```

5. Build kernel and the modules:

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

6. Install modules to the specified directory:

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

Build your own modules in cross mode

Consider the example of building a custom module in cross mode. Actions listed below are to be done in the kernel source tree.

1. Add your own module, placing it in the `drivers/misc/hello.c`:

Listing 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. Add a line in `drivers/misc/Makefile`:

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

3. Add the module description in `drivers/misc/Kconfig`:

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

4. For the module to be included in the build must be in `arch/e2k/defconfig` file to add the line:

   ```
   CONFIG_HELLO=y
   ```

   And reconfigure the kernel:

   ```
   make defconfig
   ```

5. Then you need to start the build of the modules:

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

   What causes to the build of the added module - `hello.ko`.
   Next step you should move kernel and kernel modules into `/boot` and `/lib/modules/<kernel version>` directories on the target machine correspondingly.

Support

Contacts

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

deb repositories

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

git repositories

Open source git repostories available at gitweb. For granting access contact us with one of described above methods.

For git access you should setup SSH on client machine with following way (`~/.ssh/config`):

Listing 6: /.ssh/config

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

Next you can working with git. For get available git repositories you can use the following command:

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

Listing 7: ssh info example

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

For attempting git actions you can use following syntax:

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