Other languages:
English ? ??????????? ? ???????????

Name	Founded	Status	Members	Description
OpenSCADA generic embedding and programmable logical controllers (PLC)	October 2008	Continuously appending by: • new solutions; • performance measurements for the processing systems used by OpenSCADA; • properties for the storages related to OpenSCADA.	Roman Savochenko Maxim Lysenko (2010-2012) ? the page translation	This project is dedicated for the generic conception of OpenSCADA embedding to different hardware and creation of: runtime environments of PLC, PLC firmware and hardware configurations of specialized PLC's. Considered embedding to systems based on architectures x86 and ARM. Links: • Materials for the firmware creation for x86 platform and other PLC: ftp://ftp.oscada.org/OpenSCADA/PLC/ • The notice on loader errors (propagator): https://bugzilla.altlinux.org/show_bug.cgi?id=17416 • Materials for the program environment and the firmware creation for different ARM devices: TionPro270, SMH2Gi

Contents

Modern systems of Automatic Process Control (APCS) are quite complex. Conventionally, the hierarchy of PCS can be divided into two levels: the lower and upper levels. The lower level of the PCS contains a field of equipment (sensors and executive mechanisms), as well as programmable logical controllers (PLC). The upper level consists of a system of operational visualization and monitoring of the process ? SCADA system. PLC is the responsible part of the APCS, which performs function of the data acquisition from the field of equipment, calculation and making the regulatory, blocking and other actions on the regulating parts of the field of equipment.

OpenSCADA is an open implementation of the SCADA system, which is based on the modular architecture that allows you to build the end-user solutions for different requirements. The main purpose of OpenSCADA are the systems of the upper level, but the high degree of modularity and scalability allows you also to solve wide range of tasks of adjacent areas.

1 Programmable logical controllers

PLC market is saturated with wide range of products with different architecture and design. Architecturally PLC can be divided into three groups:

- hard-programmable PLC and modular matching to object devices (MOD);
- highly intellectual commercial PLC;
- PC-compatible PLC with the free access.

Hard-programmable PLC are typically based on a single-crystal microcomputer or chips of programmable logic. Program of such controllers is flashed one-time, sometime providing the software parameterization, or formed with a specialized environment endowed with functions of binary firmware compilation of the runtime with the user program, such as ISaGRAF and LabView. As an example of such PLC can be the modules of distributed PCI of Advantech company.

Highly intellectual commercial PLC are typically based on more powerful hardware architecture and are close to full-featured PC-computer. The main difference from standard PC-compatible PLC is the closed software, and often the hardware architecture. The program software of such controllers is usually based on real-time operating system, which is planning several user threads with separation of their priorities. User programming of these PLC is made working in the corporate software which forms, as a result, the binary code of the PLC thread. As an example of such device it can be the PLC of S7 series of Siemens company.

PC-compatible PLC with the free access is not the group of the PLC directly compatible with PC, but the PLC which don't have the integrated run-time and which are often delivered without an operating system. Architecture of the such PLC may be different, ranging from cost-effective solutions with the x86 architecture and ending decisions ARM and MIPS. Run-time of the such PLC is usually formed from the software of the same with the hard-programmable PLC class, the result of which is an executable binary file into one of the most common, scalable, or specialized operating system (DOS, QNX, Linux, WinCE, VxWorks). Frequently the specialized solutions for the problem can be met. As an example of this class it can be the PLC of PC/104 form factor.

Variants of the constructive implementation of the PLC can be divided into mono-block and modular. Mono-block PLC provides the fixed configuration of the MOD, specialized for the limited range of tasks. Modular design provides an easy extension of configuration of MOD for the appropriate task. There are also the hybrid design which is the mono-block, able to expand its MOD by external MOD blocks connected to one of the standard interfaces such as RS-485.

2 OpenSCADA as run-time of PLC

Architecture of OpenSCADA allows you to create the final solutions under various requirements and resources through the modular extension. This feature is useful in the light of resources allowed by PLC. Moreover, given the constant development of hardware, as well as continuous improvement of integration and efficiency of modern microprocessor solutions, OpenSCADA can consistently extend the functionality of the PLC, while maintaining the continuity with the old solutions. For example, on the basis of OpenSCADA can be built the solutions with minimal requirements on the level: CPU 100 MHz, memory and flash ROM of 32 MB.

As noted above, the resources of modern PLCs can fluctuate in quite a large range, and the PLC of fixed type, built on single-chip microcomputer, further and further forced out into the narrowly specialized fields with the advanced PC-architectures. This trend makes increasingly interesting the possibility of creating the unified open platform for the implementation of the PLC run-time based on the unified PC-platforms.

OpenSCADA allows the realization of the idea of creating an open platform for the implementation of the run-time of PLC. Currently you can make the PLC's run-time nothing inferior to the commercial intellectual controllers, and in many respects superior to them, due to the possibility of integration of functions specific to the SCADA systems into the run-time of the PLC, enhancing the functionality and user characteristics of the PLC and leading him to unified with SCADA code base, as well as optimizing the cost of the final solution.

List functions which are solved by OpenSCADA within the run-time of PLC:

- data acquisition of various range of devices in the synchronous, asynchronous or block mode;
- user data processing and making the control actions procedures on the Java-like high level language and formal language of block schemes (Soft-logic);
- archiving data, beginning from the temporary buffers in memory and ending with full-featured archives on the file system or database of varying rate and depth;
- integration into the APCS infrastructure through the implementation of standard protocols of interaction (ModBus, SNMP, OPC UA ...);
- integration with the DBMS for the data export, storage of the configuration or archives;
- free configuration and administration of the PLC network through an operational interface of the administration station and through the Web-interface;
- the possibility of implementation the operator's panels with the control interface for integrated Touch-panel;
- providing the Web-interfaces of operational and supervisory control.

3 Architecture x86, firmware and PLC program environment creation

Architecture x86 recently was positioned as embedded and it's real solutions into this industry rarely have resources "< i386" which is not enough for full-featured OS and advanced environment execution. For this reason and by reason of big the architecture unification the individual assemblies of Linux kernel and main programs of the OS environment performed rarely enough, and that typical mostly for the architecture ARM. More interest and

practical for x86, for wide the hardware circle, it is assembling firmwares with compressing a root file-system (RFS). But still possible individual assembling by aid of the build systems as "BuildRoot" or "PTXDist", bottom. And also here possible direct installing Linux distributions.

3.1 ALTLinux distributive, instruments for assembling the firmware environments with the compressed RFS

The following requirements were pulled out to the implementation of the PLC firmware of the section:

- Compactness. In connection with the direct dependence of prices on industrial flash drives with their volume, as well as the reality of the needless for frequent updates, the image of the firmware need to be packed, reaching the level of compactness up to the 8-50MB at a runtime PLC in the full-featured OS.
- Uniform source repository. Since the firmware is not something unchangeable, not expandable and finally fixed, then it must be based on the real, developing-supporting repository of OS packages. This will allow for a long time to form updates and expansions, not maintaining the mediated repository special.
- Debugged and simple building procedure. Given the fact that configuration of the firmware can be for some time be stabilized and in addition in the operating system's components and in OpenSCADA too it will be detected and eliminated the bugs, the procedure of building of the firmware should not be burdensome, but on the contrary ? easily adaptable.
- Clear implementation of the writing mode in the OS tree. Although the firmware means the creation of packaged not modified image of firmware, but the characteristics of the PLC runtime involves the modifying of the databases and archives. In addition, the possibility of correction of the initial configuration is an important requirement.
- Reliability and stability to the sudden shutdowns. The specificity of exploitation of PLC is usually the inability to properly shutdown, as well as the practical reality of the situation instantly and unpredictable power outage. PLC in these situations must keep working, i.e. to contain the journaling file system and ensure that its verification and automatic correction of errors.
- Conditional division of the PLC configuration to two types:
    - ♦ PLC without the local display, possibly with a simple text display.
    - ♦ Touch-panels with the PLC function.

Given the above requirements, for the creation of the firmware it was chosen the tool for creating the distributions mkimage of ALTLinux. **mkimage** is the tool for building Sisyphus-based system on basis of template. As an initial set of templates it was taken the set of templates of formation of ALTLinux distributions at git://git.altlinux.org/people/boyarsh/packages/mkimage-profiles-desktop by the command:

```
$ git clone git://git.altlinux.org/people/boyarsh/packages/mkimage-profiles-desktop
```

As the basis it was taken the "rescue" template, as the most compact and close to the target PLC.

Firstly building performed basing on the package base of the distributive ALTLinux 5.1, there is present the realtime kernel from XENOMAI. For obtaining some specific packages you need to connect the repository of packages "ALTLinux 5.1" from the OpenSCADA project:

```
rpm ftp://ftp.oscada.org/ALTLinux/5.1 openscada main
```

3.1.1 Assembling

Firstly it was created the configuration of PLC without local display in mind of the availability of this type of equipment and lack of equipment for the Touch-panels.

New PLC template was named "plc", it was tested on the boards of PC/104 form factor MOPSlcdLX of the Kontron company, ATH400-128 of the Diamond Systems company and modular PLC LP-8781 of the ICP DAS company. The archive of the resulting **mkimage** tree with the "plc" template can be downloaded here ftp://ftp.oscada.org/OpenSCADA/PLC (templates and materials of individual controllers are placed in their own directories).

The key points of the configuration of new template was the writing of the new init-script (rc.sysinit), the script of the after installation configuration of the firmware's image and the list of packages in the image of firmware. The first script is designed as the package "startup-plc". The second script was embedded in the template "plc" on the way: profiles/pls/image-scripts.d/01system. The list of packages was embedded in the template "plc" on the path: profiles/pkg/lists/pl?.in.

The procedure of creating the firmware from the image is the following:

```
# Creation the configuration script configure
$ ./autoconf
# Configuring of the builder to generate the disks' images
$ ./configure --with-imagetype=flash
# Building of the image
$ make plc.cd
```

The result is an output directory in the "profiles/out/" have look:

```
syslinux/       # boot-loader folder
  alt0/         # booting files
    full.cz     # image of the primary initialization system
    vmlinuz     # OS Linux kernel
  syslinux.cfg  # configuration file of the loader
pl?             # image of the working OS, OpenSCADA and other tools
```

3.1.2 Installation

It is possible to download the firmware to: USB-flash, IDE-flash and HDD. However, in the case of the USB-flash there is the problem with waiting for initialization of USB-subsystem and you'll have "to run" some dialogues.

The file system can be FAT or EXT2/3. In the case of EXT3 the root is mounted as EXT2, because of problems in the initializer. In the case of EXT2/3 you'll need to use not the **syslinux** boot, but **extlinux**, the configuration of which is almost the same one.

Next, lets mount the medium and place the files from the output directory on it as follows.

In the case with FAT and **syslinux**:

```
syslinux/       # boot-loader "syslinux" folder
  alt0/         # boot files
    full.cz     # image of the primary initialization system
```

```
    vmlinuz     # OS Linux kernel
  syslinux.cfg  # configuration file of the loader
pl?             # compressed image of the root file-system with the OS, OpenSCADA and other tool
work            # EXT3 file system with the operational data
```

In the case with EXT2/3 and **extlinux**:

```
extlinux/       # directory of the "extlinux" loader (to rename "syslinux")
  alt0/         # boot files
    full.cz     # image of the primary initialization system
    vmlinuz     # OS Linux kernel
  extlinux.conf # configuration file of the loader (to rename syslinux.cfg)
pl?             # compressed image of the root file-system with the OS, OpenSCADA and other tool
work            # EXT3 file system with the operational data
```

To ensure the reliable operation of the operating data stored in the file "work" with the file system EXT3. The file-system of this file is checked for integrity at the initialization. This file is created as follows:

```
# Creating the operating data file at 200MB
$ dd if=/dev/zero of=./work bs=1024 count=204800
# Creating the EXT3 file-system in the file
$ mkfs.ext3 -m 0 work
```

In the case of the file system EXT2/3 on the target disk the "work" file can not be created. In this case, the working data will be placed in the directory "root" of the target disk.  This is an unreliable solution because the root file system of the target disk is not static and its check is not possible, because of earlier mounting in the "ro" and the potential unreliability of the check of the file system, mounted at "ro", as well as because of the inability to remount as EXT3.

The next step is the configuration and initialization of the loader. To configure the loader it is necessary to edit the file "syslinux/syslinux.cfg" or "extlinux/extlinux.conf" as follows:

```
default pl?
prompt 1
timeout 200
implicit 1

label pl?
  kernel alt0/vmlinuz
# To use in the case of identification of the bootable partition by the label
  append initrd=alt0/full.cz live lowmem fastboot stagename=pl? showopts automatic=method:disk,l
# To use in case of identification of the bootable partition by the identifier
#  append initrd=alt0/full.cz live lowmem fastboot stagename=plc showopts automatic=method:disk,
```

In the case of selection the identification of the bootable partition by the identifier you can get the ID of our partition with the command: **blkid**.

In the case of the label it is a bit harder and this is done for different file systems in different ways.

For the file-systems EXT2/3 it is done by the utility **e2label**. For example: **$ e2label /dev/sdb1 PLC**.

For the FAT file system it does by the set of utilities that come with **mtools** or with **parted**, easier. With **mtools** you can do it as follows:

```
# Edit /etc/mtools.conf by the line addition: drive c: file="/dev/sdb1"
# Where /dev/sdb1 is bootable partition of the target disk
# Setting the partition's label
$ mlabel c:PLC
```

Now we can initialize the loader:

```
# For "syslinux"
$ syslinux /dev/sdb1  #previously the partition must be unmounted
# For "extlinux" (path points to mount point of the partition of the target disk /media/PLC)
$ extlinux --install /media/PLC/extlinux
```

That is all with the boot and initialization of firmware. If the resulting disk is not loaded:

- Bootable flag is missing at the bootable partition.
- Incorrect or damaged "MBR". To check and restore it is possible by usage of "ms-sys" utility: **$ ms-sys -s /dev/sdb**.
- Sections of the media are created or recreated with the help of **parted**. This utility align in strange way the partitions that do not allow them to boot on USB-flash. It is necessary to repartition the media with **fdisk**.
- Boot can also be not working on the old systems that do not know how to boot from USB-HDD. For them it will be necessary to adapt the geometry of the disk being arranged to USB-ZIP or something like that.

3.1.3 Result

The result is the firmware with the size from 30MB to 100MB, satisfying all announced requirements and it provides:

- Loading for 27 seconds from the controller's switch on and including the initialization of BIOS.
- Checking and restoration of the working file system in the "work" file.
- Storing the user data and changes of the firmware in the "work" file.
- Automatic network configuration with DHCP (or 192.168.0.1).
- Access to the controller via SSH and user-friendly interface of working, including **mc**. The passwords by default (root:123456; admin:123456).
- Time synchronization via NTP.
- OpenSCADA execution with the available network interfaces:
    - ♦ configuration and runtime environment via Web (ports: 10002 and 10004);
    - ♦ control interface of OpenSCADA (10005).

3.1.4 OpenSCADA

As the PLC runtime OpenSCADA is used. For this case we'll take the building with separate packages for each module and indicate to install the virtual package openscada-plc, which contains all the dependencies on all the OpenSCADA packages, typical used for this configuration. The package of gd2 graphics library has been rebuilt without the support of xpm graphic file format and library was called libgd2-noxpm. All this was done in order to avoid the heavy dependencies on the libraries of GUI XOrg.

The result is the runtime of the PLC with support:

- DB:
    - ♦ SQLite.

- Transports:
  - Serial interfaces;
  - TCP, UDP and UNIX sockets;
  - Security Sockets Layer (SSL).
- Transport protocols:
  - HTTP;
  - Self control protocol of OpenSCADA;
  - User protocol.
- Data sources:
  - Calculator on the Java-like language of high-level;
  - Calculator on the functional blocks;
  - Controllers of the Logical Level;
  - Various PLC by the protocol ModBus (RTU,ASCII,TCP);
  - OS data;
  - Hardware of "ICP DAS", for controllers LP-8x81.
- Archiving:
  - on the file-system.
- User Interfaces:
  - Visual Control Area (VCA) engine;
  - Visualizer VCA based on the Web-technologies;
  - Configurator of OpenSCADA based on the Web-technologies.

The configuration of OpenSCADA runs in demon mode in locale "en_US.UTF-8" (also available "uk_UA.UTF-8" and "ru_RU.UTF-8") using the local database SQLite, providing the following default network services:

- configuration and execution environment through the Web (ports: 10002 and 10004);
- control interface of OpenSCADA (port: 10005).

3.1.5 The implementation details

In this section let examine the details of the OS tree of the firmware, the initialization script rc.sysinit.plc and the script of preparation of the OS tree of the firmware.

To build the PLC firmware it was used the following list of packages:

```
interactivesystem

### Startup
kernel-image-@KERNEL@
rootfiles
startup
startup-plc
SysVinit

hwclock

### Common
coreutils
glibc-locales
glibc-nss
glibc-utils
glibc-timezones
```

```
sysfsutils
sysklogd
util-linux
schedutils

### Disk utils
hdparm

### Applications
binutils
pciutils
procps
shadow-suite

### Applications/Archiving
gzip

### Applications/File
less

### Filesystem utils
e2fsprogs

### Applications/Networking
etcnet
iputils
mailx
openssh-server
dhcpcd
ntpd

### Applications/Shells
bash
mc

udev
dbus
hal
libgd2-noxpm
fonts-ttf-liberation
lm_sensors
nut
nut-driver
nut-server
watchdog

### OpenSCADA Console
#openscada-plc

## OpenSCADA GUI
openscada-visStation
autologin
icewm
wm-select
xorg-x11-server
#xorg-drv-geode
xorg-x11-utils
fonts-ttf-dejavu
fonts-bitmap-terminus
```

List of the modules of the loader's system kernel with the purpose to reducing the initialization image size was decreased to the following ones:

```
loop.ko
mtouch.ko
pcmcia_core.ko
rsrc_nonstatic.ko
yenta_socket.ko
scsi_mod.ko
libusual.ko
ide-disk.ko
ide-core.ko
sd_mod.ko
usbcore.ko
ehci-hcd.ko
ohci-hcd.ko
uhci-hcd.ko
appletouch.ko
usbhid.ko
usbtest.ko
usb-storage.ko
mbcache.ko
jbd.ko
ext2.ko
ext3.ko
fat.ko
nls_base.ko
nls_cp866.ko
nls_koi8-r.ko
nls_utf8.ko
zlib_inflate.ko
squashfs.ko
unionfs.ko
vfat.ko
pata_cs5536.ko
amd74xx.ko
```

To the script of the tree preparation there were added the following functions:

- changing the name of the distribution kit;
- replacing the init-script to "inittab.plc";
- creating the user-admin "admin" and the passwords setting by default to "123456" for users "root" and "admin";
- initialization of the /etc/fstab;
- setting locale to "en_US.UTF-8";
- network configuration;
- clock setting on the whole and the time-zone setting to "/usr/share/zoneinfo/Europe/Kiev", for changing what you need replace the file "/etc/localtime" to need zone;
- enabling of the necessary services;
- removing of the documentations, help pages and information, icons, RPM-database and apt cache;
- removing of the not needed locales and the translations; left for presence only the locales: en_US, uk_UA and ru_RU;
- selection of only used kernel modules and deletion all the others is added; it is disabled by default and can be enabled to preparation the final firmware for specific equipment;

- the directory with the kernel (/ boot) is deleted in connection with its moving to the root of the boot partition.

Initialization script (rc.sysinit.plc) was provided with the following functions:

- remounting the root partition to the RW mode;
- checking and connection of the file system (/image/root) of the working user data (the file "work");
- reflection of the modifiable directories of the PLC tree (/etc, /var, /root, /home, /mnt and /lib) to the file system with the user's working data (/image/root) through the file system "aufs" or "unionfs".

As the result of these actions the mount table of the resulting PLC tree looks like:

```
[root@localhost ~]# cat /etc/mtab
rootfs / rootfs rw 0 0
udev /dev tmpfs rw,size=10240k,mode=755 0 0
/dev/hda1 /image vfat rw,fmask=0022,dmask=0022,codepage=cp866,iocharset=utf8,check=r 0 0
/dev/loop0 / squashfs ro 0 0
/proc /proc proc rw 0 0
sysfs /sys sysfs rw 0 0
tmpfs /tmp tmpfs rw 0 0
/dev/loop1 /image/root ext3 rw,noatime,nodiratime,errors=continue,data=ordered 0 0
/image/root/etc /etc unionfs rw,dirs=/image/root/etc=rw:/etc=ro 0 0
/image/root/var /var unionfs rw,dirs=/image/root/var=rw:/var=ro 0 0
/image/root/root /root unionfs rw,dirs=/image/root/root=rw:/root=ro 0 0
/image/root/home /home unionfs rw,dirs=/image/root/home=rw:/home=ro 0 0
/image/root/mnt /mnt unionfs rw,dirs=/image/root/mnt=rw:/mnt=ro 0 0
/image/root/lib /lib unionfs rw,dirs=/image/root/lib=rw:/lib=ro 0 0
devpts /dev/pts devpts rw,nosuid,noexec,gid=5,mode=620 0 0
shmfs /dev/shm tmpfs rw,nosuid 0 0
```

3.1.6 Setting of the GUI

One option of the firmware is built with a graphical interface, which, however, necessary to configure for automatic startup with the visualization area of OpenSCADA. In addition, it should be noted that the firmware with a graphical interface does not contain all the drivers and you may have to rebuild it under the right equipment.

After downloading and logging to the console it is necessary to configure the XServer, automatic graphical login, start of the graphical environment and automatic startup of OpenSCADA from the IceWM environment:

```
# XOrg configuration file downloading from ftp://ftp.oscada.org/OpenSCADA/PLC/xorg.conf,
# placing it to folder /etc/X11
# !! File /etc/X11/xorg.conf edit to self needs.

# Keyboard's layout switch adding for: English, Russian, Ukrainian
$ echo "-option grp:lctrl_lshift_toggle -variant ,winkeys,winkeys -layout us,ru,ua -model pc104"

# Test startup
$ startx

# Configuration of an automatic graphical login by creating the file /etc/sysconfig/autologin wi
$ echo "AUTOLOGIN=yes" > /etc/sysconfig/autologin
$ echo "USER=admin" >> /etc/sysconfig/autologin
$ echo "EXEC=/usr/bin/startx" >> /etc/sysconfig/autologin
```

```
# Enabling, startup of the graphics at boot
$ chkconfig dm on

# Starting the graphics subsystem
$ service dm start

# !! Further actions are made in the terminal from the graphical interface

# Creating the configuration file of automatic OpenSCADA startup from the IceWM
$ mkdir ~/.icewm
$ echo "#!/bin/sh" > ~/.icewm/startup
$ echo "xset -dpms; xset s off" >> ~/.icewm/startup
$ echo "/usr/bin/openscada_start" >> ~/.icewm/startup
```

3.1.7 ALTLinux T6 packages base

Next stage of the firmwares creation was moving to the package base of distributive ALTLinux T6. On the whole firmwares creation concept saved, with bits changes, but there were added some improvements and expansions:

- Used the new function of "propagator" (the system of pre-initiation of hardware, searching and starting to the firmwares on FS) for searching/creation a partition of EXT2/3/4 with the label "alt-live-storage" for the "root" partition forming and it's modification possibility. The function provides a feature of installing packages direct from the distribution repository, and updating packages packed to the firmware, exclude the kernel and some system's services.
- Added the possibility of creation the firmware as a combined ISO-image, which you can (besides it writing to CD/DVD) direct, at aid of the utility **dd**, write to USB-flash, HDD, SSD and get a work environment with the partition "alt-live-storage", the "root" reflection, on the storage's free place.
- For the possibility to creation of new firmwares to "LP-8x81" from ICP DAS was done moving the real-time kernel "rt-up" from the repository "ALTLinux 5.1" to "ALTLinux T6". The kernel "rt-up" was successfully adapted and get the working firmware based on the packages base "T6" for "LP-8x81".

Due the possibility of a free additional installation of needs packages direct from the repository gone needs to the separated built of the firmware with GUI. That is you can easy install the desired window manager (WM) or desktop environment include needed drivers, than create a separated firmware with a limited list of the drivers.

The script "startup-plc" was turned spare into the new firmwares besides the "root" FS remounting to writing does early on the initial stage. The script "profiles/pl?/image-scripts.d/01system" renamed to "profiles/pl?/image-scripts.d/init1-PLC", but it changed and expanded. The packages list of the firmware was left into "profiles/pkg/lists/pl?.in" and some changed.

For get the some specific packages you have to connect the repository "ALTLinux T6" from the OpenSCADA project:

```
rpm ftp://ftp.oscada.org/ALTLinux/t6 openscada main
```

The firmware creation procedure mostly remained unchanged:

```
# Creation of the configuration script "configure"
$ ./autoconf
# The builder configuration for the disk's images generation. The key "--with-imagetype" you can
# for creation the combined ISO-image
$ ./configure --with-distro=kdesktop --with-branding=altlinux-kdesktop --with-version=6.0 --with
```

```
# The image assembling
$ make plc.cd
```

The output folder's content with the image and the firmware installing process to file system FAT and EXT2/3/4 different only by renaming the FS archive's file from "plc" to "live". Installing the ISO-image to USB-flash, HDD, SSD performs by the command **dd**:

```
$ dd if=LP8x81-ALTLinuxT6-OpenSCADA_0.8.0.6-i586-plc.iso of=/dev/sd{x} bs=4096
```

Instead the file "work" you should create partition EXT3 with the label "alt-live-storage", if it is not the ISO-image. The new partition creation you can do with the help of **fdisk**, if the FAT partition was not created to all allowed the storage space, or with help of **parted** where the FAT partition you allowed to change. To the details about a partition creation the reader will send to the documentation on **fdisk** or **parted**.

Configuration of files "syslinux/syslinux.cfg" and "extlinux/extlinux.conf" were not changed, besides the FS file archive's name changed from "plc" to "live".

At the result we get the firmware with size from 60MB, which provides:

- Loading for 25 seconds from the controller's switch on and including the initialization of BIOS.
- Checking and restoration of the journal of the working file system "root" into "alt-live-storage".
- Storing the user data and changes of the firmware, also new and updated packages, in the partition "alt-live-storage".
- Automatic network configuration with DHCP (or 192.168.0.1), for the first interface.
- Access to the controller via SSH and user-friendly interface of working, including **mc**. The passwords by default omit and the user "root" allowed for connection to it via SSH.
- Time synchronization via NTP.
- OpenSCADA execution with the available network interfaces:
    - ♦ configuration and runtime environment via Web (ports: 10002 and 10004);
    - ♦ control interface of OpenSCADA (10005).

For the PLC firmware assembling used next packages list:

```
# INIT 3
#
acl
usbutils
screen
acpid
acpi
anacron
vim-minimal
mc
sound-scripts
alsa-utils
apt
udev
udev-initramfs
dbus
schedutils
pciutils
setserial
lm_sensors3
```

```
rsync
interactivesystem
su
system-report
mtools
netcat
strace
binutils
syslogd
glibc-utils
glibc-gconv-modules
glibc-nss
glibc-timezones
glibc-locales
shadow-utils
keyutils
lsof
sudo

# INIT3: BLOCK DEVICES
#
hdparm
fdisk
ms-sys
syslinux
dosfstools
e2fsprogs

#network
etcnet
dhcpcd
xinetd
iftop
lftp
wget
iptables
ntp
ntpd
ntpdate
tcpdump
multipath-tools
openssh-clients
openssh-server
ppp-pppoe

#### LP
kernel-modules-icp-@KERNEL_MOD@

#### Console OpenSCADA
libgd2-noxpm
openscada-plc
openscada-DAQ.DiamondBoards
openscada-DAQ.ICP_DAS
openscada-DAQ.Comedi
```

The Linux kernel modules list of the initial stage was some changed and include:

```
loop.ko
```

```
mtouch.ko
pcmcia_core.ko
rsrc_nonstatic.ko
yenta_socket.ko
scsi_mod.ko
libusual.ko
ide-disk.ko
ide-core.ko
ide-gd_mod.ko
sd_mod.ko
usbcore.ko
ehci-hcd.ko
ohci-hcd.ko
uhci-hcd.ko
appletouch.ko
usbhid.ko
usbtest.ko
usb-storage.ko
mbcache.ko
jbd.ko
ext2.ko
ext3.ko
fat.ko
nls_base.ko
nls_cp866.ko
nls_koi8-r.ko
nls_utf8.ko
zlib_inflate.ko
squashfs.ko
unionfs.ko
aufs.ko
vfat.ko
pata_via.ko
pata_cs5536.ko
amd74xx.ko
```

Script of the tree preparation "profiles/pl?/image-scripts.d/init1-PLC" performs the functions:

- changing the name of the distribution kit;
- enabling login from the user "root" via SSH, without a password;
- setting locale to "en_US.UTF-8";
- network configuration for use DHCP or set to "192.168.0.1" for the first interface;
- clock setting on the whole and the time-zone setting to "/usr/share/zoneinfo/Europe/Kiev", for changing what you need replace the file "/etc/localtime" to need zone;
- enabling of the necessary services;
- removing of the documentations, help pages and information, icons;
- removing of the not needed locales and the translations; left for presence only the locales: en_US, uk_UA and ru_RU;
- selection of only used kernel modules and deletion all the others is added; it is disabled by default and can be enabled to preparation the final firmware for specific equipment; it was unified for the modules list setting by groups of the kernel subsystem?s and specified for hardware;
- the directory with the kernel (/ boot) is deleted in connection with its moving to the root of the boot partition.

3.1.8 Real-time kernel

For a series of tasks are important, often also critical, criteria of the environment is the real-time handing level, then it is possibility of working the tasks according to the real-time priorities and provision of a reaction to events by that priorities.

The Linux kernel of itself provides POSIX real-time scheduling policies "SCHED_FIFO" and "SCHED_RR" with the priorities range (0...100). But the important criteria is "Timer frequency and the reaction to it" up to version of Linux kernel 2.6.24 was too low, for criteria of the real-time systems. In modern kernels of Linux (> 2.6.24) provided support for timers of the high-precision resolution (HPET), what decreased the reaction time to a timer up to 100 microseconds, but that time stability is not guaranteed. To ensure stability of reaction to a timer on level 60 microseconds, and also series of the other criteria of real-time, at the moment you need to assemble a kernel with one real-time extension.

On the ALTLinux distributions observed the kernel 2.6.29-rt-up, which assembled with the real-time extension of real-time is XENOMAI. Into other distributions, for example OpenSuSE, observed also the solutions with it extension.

For now higher criteria of the real-time ensured of the extension The Real Time Preempt Patch, on enabling here full it's features by (CONFIG_PREEMPT_RT), The patch assembling to Linux kernels process and it's work result will trace into this section.

For the real-time level testing on different kernels will use the utility "Cyclictest", which typical call command line and it's arguments looks: "**$ cyclictest -t1 -c1 -p 80 -n -i 200 -l 100000**". Where:

- *-t1* ? single testing thread;
- *-?1* ? using the clock of real-time CLOCK_REALTIME;
- *-p 80* ? priority of the testing thread;
- *-n* ? using the function "clock_nanosleep";
- *-i 200* ? pulling interval of the testing thread: 200 us, as closer to the average value 50 us;
- *-l 100000* ? testing iterations number.

Pair of measurements for Linux kernels of the wide purpose:

- ALTLinux T6 distribution standard kernel (3.0.79-std-def, LP-8781):
  - *loading 0%*: Avg: 37 us; Max: 152 us
  - *loading 100%*: Avg: 37 us; Max: 191 us
- ALTLinux T6 distribution kernel (3.4.45-un-def, LP-8781):
  - *loading 0%*: Avg: 53 us; Max: 217 us
  - *loading 100%*: Avg: 40 us; Max: 183 us

3.1.8.1 kernel-image-rt-up-2.6.29

The kernel originally presents into the distribution ALTLinux 5.1, and it is also moved to the local repository of the project OpenSCADA for ALTLinux T6. The kernel assembled with the extension XENOMAI and AUFS, allowing it using into the firmwares and packed root FS, that was done for the PLC LP-8x81.

Results of the kernel testings:

- *LP-8781 (XENOMAI)*
  - *loading 0%*: Avg: 26 us; Max: 136 us (there observed jumps up to 1 ms, like to instability the source clock "tsc", at that the "acpi_pm" worse here)
  - *loading 100%*: Avg: 24 us; Max: 3655 us
- *LP-8781 (CONFIG_PREEMPT_NONE)*
  - *loading 0%*: Avg: 29 us; Max: 71 us
  - *loading 100%*: Avg: 28 us; Max: 117 us (there observed jumps to continuous the measured value rising)
- *LP-8781 (CONFIG_PREEMPT_RT)*
  - *loading 0%*: Avg: 29 us; Max: 64 us
  - *loading 100%*: Avg: 29 us; Max: 82 us
- *AMD Turion Neo X2 L625 (CONFIG_PREEMPT_RT)*
  - *loading 0%*: Avg: 57 us; Max: 74 us
  - *loading 100%*: Avg: 57 us; Max: 78 us

As you can see from the testing results the patch XENOMAI does not ensure proper level of the real-time on using the standard mechanisms of the POSIX real-time scheduling, at the same time as the kernel version 3 even without the specific real-time extensions ensures the clearly better result.

Necessity for assembling same that kernel with the patch/parameter CONFIG_PREEMPT_RT is actual by presence a number of binary modules from ICP_DAS, for "LP-8x81". Also here is actual a question of building the kernel 2.6.33 at the same reasons but for "LP-8x81 Atom". The preliminary assemblies of the kernels 2.6.29 and 2.6.33 reveals series of problems which will described here. Solving also a variant to build a modern kernel with CONFIG_PREEMPT_RT and next to request for building these binary modules from "ICP DAS".

Assembling and testing process:

1. Patches CONFIG_PREEMPT_RT and AUFS of days of 2.6.29 are conflict on the function "debug_mutex_set_owner()", into CONFIG_PREEMPT_RT it removed ? replaced to "mutex_set_owner()".
2. On the assembling here detected series of problems for "# typedef void irqreturn_t;" ? replaced to "#include <linux/irqreturn.h>".
3. The first start with CONFIG_PREEMPT_RT, but without AUFS, was successful ? the results above.
4. The starting with AUFS was reveal a problem into memory allocation by AUFS into "aufs_mmap()" ? the working code of AUFS was taken from the preliminary assembling "rt-up-2.6.29.alt2".
5. It starting with AUFS was reveal a problem of hanging on the FS root into AUFS, like to possibility of cycling/blocking a RT-task ? setup CONFIG_PREEMPT_NONE, on LP-8781 and "AMD Turion" any problem does not observed (possible the problem due the HPET missing on PLX8).
6. At the first look the kernel works fine, but jumps to continuous growing the measured value of the delay time observed.
7. There finished adapting of the kernel to a binary compatibility for the modules "slot" and "icp" from ICP_DAS. The module "8250_linpac" crashes at it's loading, and "icpdas_8250" has more unresolved symbols ? the modules need to reassemble or to try the interfaces > COM2 initiate through **setserial** ? the modules was reassembled by the help of Golden Wang (technical support of ICPDAS).
8. The new kernel set to high loading by the project configuration ACS ball drum mills:

   ◊ network with the driver "via_rhine" halted, after 4 days of successful working ? the halt expected, assembled the driver "rhinefet", testing continued.

- ◊ on the driver "rhinefet" the system on loading had work three weeks. But here observed the interrupt 11, on what hangs mostly all standard hardware (USB, Ethernet and may be something), had disabled and the network continued to work into "Pool" mode, which is slower. Possible that interruption disabling occurs also with "via_rhine", and it can not work in the mode "Pool", why packages into the network do not go. The problem reason linked to halt and generation the unhandled interruptions from one of hardware on the interruption 11.
- ◊ The problem fixed by preventing the interruption disabling at help the Linux kernel parameter "noirqdebug".
- ◊ The adapting successful finished and the firmwares based on the kernel ready to the production implementation!
- ◊ 01.03.2015: Instead function EnableWDT() used EnableSysWDT(), by limit to 30 seconds and cyclic reloading if the system was not loaded in 30 seconds (up to three reloadings).
- ◊ 17.03.2015: With assist of the ICP_DAS support service there was fixed a problem into the serial interfaces driver for more to COM2 which causes to Linux kernel "freeze" (like to interruptions block) after closing one port and activity on some other.
- ◊ 29.07.2015: Detected one more problem looks like by the symptoms to the interruption 11 disabling, but: the interruption 11 is not disabled and all other devices on it works. It reproduced only on configurations with using that both network interfaces, at that possible "braking" for one of its. The problem resolved only by reloading "the braked" network interface, by the command: **ifdown eth0; ifup eth0**. To detect it and the reloading performs we recommend on the OpenSCADA level append the traffic control and same reloading of the interface on the traffic lack.
- ◊ 21.11.2016: Driver "rhinefet" has been adapted to prevent the interrupts lock and the interrupt vector disable but SHARE mode using. For now it works but 19.12.2016 also there was observer the two adapters network slowing after about two week working.
   Then the hardware is broken for two adapters work and for the PLC you can use only one for stable work!
- ◊ 06.09.2017: Not full fixing into the serial driver of ICP-DAS, caused to malfunction on using more then two serial ports, seems already full fixed into last versions, which observed on LX-8x31.

 Result kernel, which renamed to "kernel-image-rt1-up-2.6.29.alt1", you can use for PLC with HPET or a high precision timer, and also into "LP-8x81" and "LP-8x81 Atom" (only single kernel)!

3.1.8.2 kernel-image-rt-up-2.6.33

Kernel version 2.6.33 assembling with the patch CONFIG_PREEMPT_RT needs for PLC LP-8x81 of firm "ICP DAS" and LP-8x81 Atom (main and original kernel) by reason of presence for it the binary drivers of "ICP DAS".

Testings results of the kernel:

- *AMD Turion Neo X2 L625*
  - ♦ *loading 0%*: Avg: 65 us; Max: 86 us
  - ♦ *loading 100%*: Avg: 57 us; Max: 72 us
- *LP-8781*
  - ♦ *loading 0%*: Avg: 37 us; Max: 88 us
  - ♦ *loading 100%*: Avg: 34 us; Max: 108 us
- *LP-8781-Atom (original assembling of the kernel 2.6.33.7-rt29-ICPDAS)*
  - ♦ *loading 0%*: Avg: 17 us; Max: 50 us
  - ♦ *loading 100%*: Avg: 12 us; Max: 32 us

Assembling and testing process:

1. Assembling of the kernel from sources of "ICP DAS" (2.6.33.7) and configuration it inherited from kernel 2.6.29 (the source code has suspiciously more of *.rej files, and also "staging/comedi" impossible to build) ? it started and mostly works; modules "ipic" and "slot" loaded; module "8250_linpac" crashes into function "platform_device_add"; series of programs hangs into FS operations, with the message: "task openscada:2153 blocked for more than 120 seconds".
2. AUFS replacing to the version from 2.6.29-rt1 ? crashes into rtmutex on starting; replacing to official version from git looks same result, at the begin used patch "aufs+sqfs4lzma-2.6.33.patch" from DLink.
3. Assembling the original kernel with patches CONFIG_PREEMPT and AUFS ? a problem again into AUFS, but now it at the finish can not look "/sbin/mingetty".
4. Assembling the original kernel 2.6.33.9 with patches CONFIG_PREEMPT and AUFS ? same problems.
5. Assembling from sources of "ICP DAS" (2.6.33.7) for SMP ? the module DAQ.JavaLikeCalc of OpenSCADA crashes at an unknown reason.
6. Assembling the original kernel with patches CONFIG_PREEMPT and AUFS for SMP ? same problem as without SMP but it is not immediately and about five thread.

 For now the kernel 2.6.33 links with CONFIG_PREEMPT_RT and AUFS is non-working. Then if you need to work on "LP-8x81 Atom" then we recommend to use the original Linux-environment, building and installing the OpenSCADA here.

3.2 Debian distributive, instruments for assembling the firmware environments with the compressed RFS

 Waiting to form

4 Architecture ARM, firmware and PLC program environment creation

Widespread in embedded solutions the ARM architecture obtained due to its relatively high productivity coupled with low power consumption and cost. In order to perform planed task to provide the hardware multiplatform OpenSCADA was adapted for the building and operation on the equipment of the ARM-architecture. Thus, the following projects were made Building the OpenSCADA project for the mobile devices of the Nokia company (N800, N900, N950) and Building OpenSCADA and firmware for the ARM-based controllers from ICP DAS (LP-5141). The purpose of this section is to systematize the procedures and track the problems of creating the OpenSCADA buildings and software environment firmwares as a whole for a variety of embedded ARM-architecture hardware.

Specific of the ARM architecture is the lack of a necessarily hardware-dependent software system of the basic initialization and configuration of equipment, which is characteristic for the x86 architecture ? BIOS, and the structure of hardware configuration typically includes: CPU, integrated operational and flash memory, as well as a number of built-in equipment on a standard system-level buses. The flash and RAM are placed in general address segment. Initialization of the such system with the software environment is made by downloading executable code directly on the built-in flash-memory.

To working of computing functions of OpenSCADA and other related libraries and software the performance of floating point calculations is very important. Specific of the ARM architecture processors is the ease of its core and availability of optional extensions such as math coprocessor. As a consequence, the performance on floating point

operations is highly dependent on the specific processor, and on the emulation type of the floating point coprocessor if it is absent at all. There are two formats of floating point in the ARM-architecture processors: FPA and VFP. FPA format is obsoleted and met as a hardware implementation in the ARM cores up to the StrongARM family (ARMv4). XScale ARM core families (ARMv5TE) did not have a math coprocessor at all. And the ARM core, starting with the ARM11 family (ARMv6) are equipped with VFP format math coprocessor. At the same time the ARM processors with the ARMv5 architecture are still widespread, and thus the question of performance of mathematical calculations for them comes down to the performance of the FPA or VFP format emulation. In the case of the Linux environment the emulation of FPA is usually done by the Linux kernel by the CPU exceptions handling when calling FPA commands. Software emulation in the math library is usually found with the VFP format which requires the rebuilding of all programs. The FPA emulation by means of exceptions is much worse than the performance of software VFP emulation. You can compare the performance of floating-point calculations on different architectures, processors and ways of emulation in the part "Processing systems productivity".

The typical software environment based on the Linux operating system for ARM based hardware is: Loader UBoot, Linux kernel and root file system (RFS). UBoot loader is loaded into the zero sector of flash memory, and its settings are stored in the first one. From the second sector the kernel code is loaded, and immediately after it ? the RFS. RFS is usually used as basis the JFFS2 or UbiFS file system, which are optimized to work on block devices ? flash memory with a limited resource of records. Examples of partitioning a block device (flash memory) for LP-5141 and TionPro270 are presented below:

```
# LP-5141
$ cat /proc/mtd
dev:    size   erasesize  name
mtd0: 00040000 00020000 "Bootloader"
mtd1: 00040000 00020000 "Bootloader Param"
mtd2: 00280000 00080000 "Kernel"
mtd3: 03c80000 00080000 "JFFS2 Filesystem"
# TionPro270
$ cat /proc/mtd
dev:    size   erasesize  name
mtd0: 00080000 00040000 "Bootloader"
mtd1: 00400000 00040000 "Kernel"
mtd2: 01b80000 00040000 "Filesystem"
```

The root file-system contains a typical UNIX-tree with work programs, libraries and other files. The basis of any program or library are the system libraries GLibC or UClibC. OpenSCADA is adapted for building and operating with "GLibC" version >= 2.3. "UClibC", created as a lightweight version of "GLibC" for embedded systems, contains a number of limitations and has not yet been implemented or has errors in the implementation of a number of functions.

RFS and software environments based on Linux can be supplied with the ARM-equipment and contain closed binary libraries, Linux kernel modules, etc. In this case, an independent building and replacement of the original software environment is impractical task because it leads to the loss of original functionality. However, it often happens of delivery of the ARM equipment without the source (original) software environment, or with an environment that does not contain closed code and which can be replaced. An example of the first case is the controller LP-5141 and similar of the "ICP DAS" company, which contain the binary building of the specialized equipment API library (libi8k) and Linux kernel modules for its initialization. An example of the second case is the single board computer Tion-Pro270, for which the software environment and the OpenSCADA build were created from scratch.

5 OpenWrt distributive

OpenWrt ?is a highly extensible ?GNU/?Linux ?distribution for embedded devices ?(typically wireless routers). Unlike many other distributions for these routers, OpenWrt ?is built from the ground up to be a full-featured, easily modifiable operating system for your router. In practice, this means that you can have all the features you need with none of the bloat, powered by a Linux kernel ?that's more recent than most other distributions.

Then OpenWrt mostly distributed in ready firmwares or pre-installed on the proper hardware. For build or rebuild the firmware in whole or the additional packages from the sources separately the project provides SDK based on the build system "BuildRoot" which slightly changed. For using the SDK you can build it from scratch with needed toolchain or get ready one for proper OpenWrt version and platform about that you can read here in the details.

OpenWrt uses the packages IPKG and the package manager utility "opkg" then you can install ready packages, previously download them to the hardware, in way:

```
$ opkg install openscada_0.9+r2517-1_mxs.ipk
```

Due the build system BuildRoot was changed for OpenWrt and mostly into the packages description and the feeds repositories appending the build rules of OpenSCADA needed and were fully rewritten and you can download their pack from here and unpack it to folder "package". For build OpenSCADA and its IPKG package from prepared and ready for it BuildRoot folder of the SDK with OpenSCADA rules into the folder "package" you must follow next instruction:

```
# Clean the package building, at first time you can omit the command
$ make package/openscada/clean
# Download and prepare OpenSCADA source archive, the commands call automatically at the next com
$ make package/openscada/download
$ make package/openscada/prepare
# Compile OpenSCADA, all dependent packages of the OpenSCADA building at the time and before mus
$ make package/openscada/compile
# Install and build the IPKG package, which you can find here "bin/{paltform}/packages/base"
$ make package/openscada/install
```

Due using by BuildRoot the "C" library "uCLibC" there was needed some OpenSCADA adaption to uCLibC version 0.9.33.2 after the last building with uCLibC version 0.9.32.1 in year 2011:

- "resourcesAllow" true checking for build without the resources (folders "data" and "doc"), mostly by problems at **automake** here execution;
- **librt** checking changed from the function clock_gettime() to clock_nanosleep();
- for **libcrypt** checking there used function "crypt";
- including <stdarg.h> to "src/tmess.h";
- checking for __UCLIBC__ into TUser::auth();
- **iconv** buffer set to "const char *".

OpenSCADA initially for OpenWrt was built and successfully executed and exploited on a 3G router "TELEOFIS RTU968" from AO "TELEOFIS" for the platform "Freeescale i.MX23/28".

All materials about OpenWrt for OpenSCADA you can find by the link.

6 Building tools for the Linux kernel and work environments of different target architectures

Linux RFS can be formed on the basis of ready packages of the existing binary distribution, source package of the current distribution, as well as to build from the original sources through the ToolChain in one of the building systems.

Building of the programs or of an entire RFS for architectures different than x86 and x86_64, is usually made using the Cross Compilation tools (ToolChain) for building, linking and debugging for the target ARM architecture. To automate this process a number of tools to build the ready RFS created.

6.1 BuildRoot

This building system is a part of the project for creation an alternative library of functions of "C" language UClibC, so basically aims to build environments with "UClibC", and with appropriate restrictions. BuildRoot is well in the work on the host systems of different versions, and allows to build the software environments based on Linux without too much troubles.

It is possible to get the BuildRoot archive of the correct version by the link http://buildroot.uclibc.org/downloads. Further it should be unpacked to the home directory of the simple user and the configuration, setup and building should be done:

```
# Initial configuration relative to the previous one, for example after changing the BuildRoot v
$ make oldconfig
# Configuration from the character graphics menu
$ make menuconfig
# Starting the building
$ make
# Entire cleaning the build environment
$ make distclean
```

The building process can cause following problems:

- Inability to download the programs archive.

  (+) This package should be downloaded separately and put to the directory "./dl" or "./output/dl".

- Programs' building errors.

  (+) There is no single solution to this problem you must to understand the reason of the building of individual programs. Building error may be linked, for example, with the lack of choice of the individual parameter during the configuration or problem building of the software in this environment. Patches and fixes of the building can be placed directly in the directory of the program description "./package/{package name}/"

6.2 PTXDist

Universal tool for building kernels, ToolChains and software environments based on Linux from the "Pengutronix" company. PTXDist is a powerful and flexible tool, but it's older versions have problems in the modern host

systems, which complicates the task of building the software environments for relatively old but still prevalent hardware platforms. For example, now (2012) can be found new hardware with the ARM XScale, ARM9 (ARMv5) processors of the year 2003. However, newer versions of PTXDist support the old platforms, what can be learned from the support table by the link: http://www.pengutronix.de/oselas/toolchain/index_en.html.

To build the software environment (RFS) using PTXDist it is necessary:

- to get the builder's tool archive (along with the next versions projects, http://www.pengutronix.de/software/ptxdist/download), compile and install it;
- to get archive of source files (the same or similar version as PTXDist, http://www.oselas.com/oselas/toolchain/download) and build ToolChain;
- to clone-create and build the RFS project.

Now detailed, in commands:

```
# Installation and building of the builder must be done from the simple user in his home directo
# The source archives are loaded to the ~/Downloads
$ mkdir ~/proj/ptxdist; cd ~/proj/ptxdist
$ tar xvjf ~/Downloads/ptxdist-2011.11.0.tar.bz2
$ tar xvzf ~/Downloads/ptxdist-2011.01.0-projects.tgz
# Transfer the contents of the projects' archive to the working version's directory, if the vers
$ cp -r ptxdist-2011.01.0/* ptxdist-2011.11.0/; rm -rf ptxdist-2011.01.0
# Building and installation of the toolkit
$ cd ptxdist-2011.11.0; ./configure --prefix=/home/roman/proj/ptxdist; make install
# Setting the environment variable "PATH" to call the toolkit file "ptxdist"
$ export PATH=$PATH:/home/roman/proj/ptxdist/bin

# Unpacking, configuration and building the Toolchain
$ cd ~/proj; tar xvjf OSELAS.Toolchain-2011.11.0.tar.bz2
# Select the desired configuration of toolchain
$ cd OSELAS.Toolchain-2011.11.0
$ ptxdist select ptxconfigs/arm-xscale-linux-gnueabi_gcc-4.6.2_glibc-2.14.1_binutils-2.21.1a_ker
# Start of building the ToolChain.
# The building result will be placed in the directory /opt/OSELAS.Toolchain-2011.11.0, you need
$ sudo chmod a+rwX /opt
$ ptxdist go

# Cloning-creation of the RFS project
# Presetting of the general configuration of ptxdist, for example, the path to a directory of pr
$ ptxdist setup
# Checking for availability and visibility of the projects to clone
$ ptxdist projects
# Cloning the one of the available projects
$ cd ~/proj; ptxdist clone OSELAS.BSP-Pengutronix-Generic New_RootFS
# Choice to a platform configuration and the previously built ToolChain for this project
$ cd ~/proj/New_RootFS
$ ptxdist platform configs/arm-qemu-2011.01.0/platformconfig
$ ptxdist toolchain /opt/OSELAS.Toolchain-2011.11.0/arm-xscale-linux-gnueabi/gcc-4.5.2-glibc-2.1
# Other settings of the architecture, programs selection and formation of the result are configu
$ ptxdist menuconfig
# Building of the RFS and the formation of images
$ ptxdist go
$ ptxdist images
# Additional programs configuration of the building should be placed in directory "rules" as two
# For a new program to be appeared in the menu of pseudographics configurator, it must be added
# to the file ~/proj/ptxdist/lib/ptxdist-2011.11.0/rules/Kconfig
```

7 Performance measurement

7.1 Processing systems

Hardware	Enter into JavaLikeCalc, us**	Operation sin(Pi) [into JavaLikeCalc], us	Operation pow(Pi,2) [into JavaLikeCalc], us	Model AGLKS [Vision, main mnemo], %(core)	Extra tests and notes
ARM					
Segnetics SMH2Gi (ARM926EJ-S, 400 MHz, 65nm SoftVFP, 199 BogoMIPS)	3.4	11.1 [14.9]	4.4 [7.9]	-	
Router 3G TELEOFIS RTU968 (ARM926EJ-S, 454 MHz, 65nm, OpenWrt, uCLibC, SoftVFP, 226 BogoMIPS)	2.45	7.2 [9.75]	2.02 [5.45]	-	
ICP DAS LP-5141 (PXA270, 520 MHz, FPA)		100 [200]*	51 [152]*		
ZAO ZEO TionPro270 (PXA270, 520 MHz, SoftVFP, uCLibC-0.9.32.1, -Os, 519.37 BogoMIPS)		22 [51]*	14 [41]*	-	Minimum power consumption: 1.27 W
ZAO ZEO TionPro270 (PXA270, 520 MHz, SoftVFP, GLibC-2.14.1, -O2, 519.37 BogoMIPS)		5.92 [8.26]	1.74 [4.08]	-	Last update: 30.10.2013
Nokia N800 (TI OMAP2420, ARMv6, 90nm, 400 MHz, 397 BogoMIPS)	2.32	2.93 [6.29]	2.11 [6.98]	-	
Raspberry Pi (BCM2708, ARMv6, 700 MHz)		1.15 [4.57]	1.28 [4.60]	-	
Nokia N900 (TI OMAP3430, CortexA8, 65nm, 600 MHz, 598.9 BogoMIPS)	1.23	1.55 [1.9]	0.932 [1.38]	>100	
Nokia N950 (TI OMAP3630, CortexA8, 45nm, 1 GHz)		0.90 [2.02]	0.552 [1.81]	>100	Last update (Turbo N900): 02.11.2013

Raspberry Pi 2 (BCM2836, ARMv7, 1 GHz, 4 Cores)	0.525	0.615 [0.955]	0.41 [0.875]	85 [154]	Minimum power consumption (on 600MHz): 1.02, 1.14 (+Eth), 1.33(+WIFI)
Orange Pi Zero (Allwinner H2(+), Cortex A7, 1.2 GHz, 4 Cores)	0.483	0.497 [0.757]	0.33 [0.627]	-	Minimum power consumption (on 240MHz): 0.89, 1.02(+Eth)
Raspberry Pi 3 (BCM2837, ARMv8, 1.2 GHz, 4 Cores)	0.379	0.43 [0.496]	0.295 [0.389]	75 [130]	Minimum power consumption (on 600MHz and indifferent to enabled WIFI or Bluetooth): 1.14, 1.39(+Eth)
HTC Desire 820G (MediaTek MT6592, Cortex-A7, 28nm, 1.7 GHz, 8 Cores)	0.416	0.237 [0.315]	0.236 [0.342]	-	
Asus Nexus7 II (Qualcomm Snapdragon APQ8064-1AA, Cortex-A15, 28nm, 1.5 GHz, 4 Cores)	0.497	0.161 [0.306]	0.126 [0.341]	54 [82]	armv7-a, Soft, VFP. Extra tests.
x86					
Cyrix Geode(TM) (232 MHz)		7 [44]*	11 [52]*	-	
VIA Nehemiah (400 MHz, 130nm)		2.9 [5.8]	2.4 [5.8]	-	
AMD K6-2 (504 MHz, 250nm, 1008 BogoMIPS, BUS 112 MHz)		1.136 [4.66]	1.602 [5.63]	-	
AMD Geode LX800, ICP-DAS LP-8x81 (500 MHz, 130nm, 1000 BogoMIPS)	1.04	1.27 [1.66]	2.03 [2.61]	-	
		2.7 [5.6]*	2.4 [6.1]*	-	

VIA Nehemiah (667 MHz, 130nm)				
RDC R3600, ICP-DAS LX-8x31 (1.0 GHz, 2 Cores)		0.72 (1.52)	1.14 (2.06)	-
Intel(R) Atom(TM) CPU Z520, ICP-DAS LP-8x81 Atom (1.33 GHz, 1[2] Cores, 45nm)		0.39 (1.14)	0.53 (1.12)	-
Intel Atom N270 (1.6 GHz, 1[2] Cores, 45nm, DDR2-533-1.6GB/s)	0.374	0.416 [0.613]	0.418 [0.686]	82 [151]
Intel(R) Celeron(R) CPU 847 (1.1 GHz, 2 Cores, 32nm)		0.23 [0.675]	0.25 [0.76]	50 [64]
AMD Phenom(tm) 9600 Quad-Core (2.3 GHz, 4 Cores, 65nm)		0.17 [0.45]	0.14 [0.35]	-
AMD Athlon 64 3000+ (2 GHz, 130nm)		0.15 [0.43]	0.16 [0.49]	23 [31]
AMD Athlon X2 3600+, (2 GHz, 2 Cores, 65nm)		0.145 [0.42]	0.153 [0.46]	25 [47]
Intel(R) Celeron(R) CPU N2840 (2.16GHz, 2 Cores, 22nm)		0.175 [0.389]	0.165 [0.385]	33 [60]
Intel(R) Pentium(R) 4 CPU (3 GHz, 1[2] Cores, DDR-400)	0.198	0.152 [0.206]	0.157 [0.253]	45 [77]
Intel(R) Core(TM)2 Duo CPU T5470 (1.6 GHz, 2 Cores, 65nm)	0.179	0.143 [0.18]	0.129 [0.197]	27.6 [54]
AMD Turion L625 (1.6GHz, 2 Cores, 65nm, DDR2-555-1.8GB/s)	0.096	0.125 [0.251]	0.096 [0.219]	28 [50]
Intel(R) Core(TM) i3-3217U CPU (1.8 GHz, 2[4] Cores, 22nm)		0.105 [0.277]	0.148 [0.305]	21 [26]
Intel(R) Core(TM) i3 CPU U 380 (1.33 GHz, 2[4] Cores, 32nm, DDR3-1333-4.8GB/s)		0.104 [0.257]	0.0985 [0.244]	36 [52]
Intel(R) Xeon(R) CPU E5-2603 (1.8 GHz, 4		0.074 [0.178]	0.068 [0.173]	-

Cores, 32nm)				
AMD Phenom(tm) II X4 900e (2.4 GHz, 4 Cores, 45nm, DDR2-800)	0.067	0.099 [0.1]	0.0567 [0.126]	17 [32]
Intel(R) Core(TM) i5-3610ME CPU (2.7 GHz, 2[4] Cores, 22nm)		0.05 [0.132]	0.0376 [0.122]	-
AMD A8-6500 APU (3.5 GHz, 4 Cores, 32nm, DDR3-1866-5.8GB/s)	0.0581	0.0425 [0.0572]	0.028 [0.0442]	14 [24]
Intel(R) Core(TM) i7-5600U (2.6->3.2 GHz, 2[4] Cores, 14nm, DDR3-1600-15GB/s)	0.0445	0.0288 [0.0326]	0.0266 [0.0306]	13.6 [20.5]
Intel(R) Core(TM) i3-4330 CPU (3.5 GHz, 2[4] Cores, 22nm)		0.03 [0.08]	0.023 [0.073]	-

* ? Includes double call of gettimeofday() function.
** ? Enter to procedure on the language JavaLikeCalc means also enter to the critical section and request to read of the RW lock then the time mostly show performance of the operation. The time was excluded from related values into columns with JavaLikeCalc.

 The difference in computation time for direct call of the mathematical operation and from the JavaLikeCalc virtual machine is connected with the influence of CPU core frequency (the frequency at which it operates) and who made the part of the command before the transfer of it to math co-processor and with the memory speed. Performance of the math co-processor is usually not directly connected with performance and frequency of the processor core or memory speed.

The measure methodology into the table above provides next:

1. Time estimation for operation generic lock of the critical section, "**sin(Pi)**" and "**pow(Pi,2)**", into second, third and forth columns. sin() and pow() operations selected to represent of estimation the co-processor performance and for overall real-numbers manipulations. Value into square brackets characterizes overhead charges on calculations into the virtual machine of OpenSCADA and performance for integer-numbers calculations about the exemplary operations. I.e. the main value characterizes performance of the processor into the float-point operations (mathematical co-processor or emulation), and into square brackets is the into the integer-operations (central processor), as subtraction of the real-numbers operations. The measure method:

    a) ensure the central processor frequency stability, in way of setting the policy of its management to PERFORMANCE;
    b) start OpenSCADA without load, the project by default or with an empty configuration, and within configurator UI.QTCfg, UI.WebCfg or UI.WebCfgD;

c) open the function's object "**sin()**", and next "**pow()**", by the module of mathematical functions library;
d) go to tab "Execute", set "Enable", enter the argument's value for "X" to 3.14159 and "Power" to 2 (for "**pow()**"), set number of executions to 1000 (to the representativeness rise you can increase the number, by degrees, to overall the operation execution time not more 10 seconds);
e) press "Execute" and get the execution time;
f) perform the execution into several tries, pressing "Execute", and achieve to minimal value;
g) fix the minimal value, which divide to 1000 (number of the executions) and get the main time value of the single execution into microseconds;
h) go to the module object of internal executions OpenSCADA (DAQ.JavaLikeCalc);
i) create here an object of library of functions "test", and into it a new function "test", turn it on;
j) into the tab "Program" enter the commands text zero, "**y=sin(3.14159)**", and next "**y=pow(3.14159, 2)**";
k) go to tab "Execute" and do the same things into the list items "d."-"g.";
l) the execution result consider for column two and as auxiliary, into the square brackets, for column three and four after them extraction from the column two.

2. Complex performance estimation, into fifth column, performs by execution the technological process (TP) model AGLKS on target architecture. The test can be executed only on computing systems with relatively high performance (or with more to one core), which capable for the model execute, and equipped by a graphical information output device (display), the visualization server execution case we will not consider. The main value of the processor loading characterizes only the dynamical model of TP execution, but addition value appends of forming and execution the graphical interface. The measure method:
a) ensure the central processor frequency stability, by way set the policy of the management to PERFORMANCE;
b) from desktop environment menu start the model AGLKS;
c) start a terminal emulator (e.g., "konsole"), where type "top", press Shift+H (for see the process at all) and Shift+P (sort by the processor load);
d) read the values in the column "%CPU" against to the process "openscada", select to typical value for several updates and fix it as main value;
e) return to the window OpenSCADA and start the visualization environment, and next the project "AGLKS" interface.
f) return to the terminal emulator and read addition value, like to the list item "d.".

The results you would send to [mail address] for appending to the table!

7.2 Storages (built-in, HDD, SSD, CF, SD, ...)

This section contains information about performance of different storages, from and on with was worked and working OpenSCADA solutions.

Hardware	Real size, GB/GiB	Read/Write, MB/s	Notes
SSD			
PATA SSD, Ver2.M0J (Acer Aspire One 110)		41.2/15.8	
SSD: GoodRAM Play 32GB 2.5" SATA2, MLC (SSD32G25S2MGYSM2244)		238/45	

SSD: GoodRAM C40 60GB 2.5" SATA3, MLC (SSDPR-C40-060)		498/467	
SSD: GoodRAM C40 120GB 2.5" SATA3, MLC (SSDPR-C40-120)		480/330	for EXT4
SSD: GoodRAM CX200 120GB 2.5" SATA3, TLC (SSDPR-CX200-120)	120/111	497/118	fast overheating on write 30+10 °?
SSD: GoodRAM IRIDIUM 120GB 2.5" SATA3, MLC (SSDPR-IRID-120)	120/111	499/201	heating 33+3/11 °?
SSD: TOSHIBA THNSNJ512GCSU 512GB 2.5" SATA3, MLC (JULA0101)	512/477	525/517	heating 32+4/10 °?
HDD. Typical environment: Read/Write: raw 50GB by blocks 1MiB, into end of the hard disk			
HDD 3.5": SAMSUNG SP2004C	200/186	39/38.5	heating 27+11/13.5/15°?
HDD 3.5": TOSHIBA DT01ACA050		85.4/84.4	+18°?
HDD 3.5": WDC WD15EARX-00PASB0		43/45.5	+11°?
HDD 3.5": WDC WD10EZRX-00D8PB0	1000/931	77.7/77.5	+9°?, noisely seek
HDD 3.5": Seagate ST1000VM002-1ET1	1000/931	87.1/86.9	heating 27+7/9/10 °?
HDD 2.5": WDC WD5000LPVX-22V0TT0	497/463	62.1/61.7	heating 33+5/9/11 °?
Internal flashes of the devices			
Flash: ICP_DAS LP-8x81 Internal 4GB		8/4	
Card flashes (CF)			
CF: ICP_DAS LP-8x81 8GB, MLC		27/(19...15)	
CF: ICP_DAS LP-8x81 8GB, MLC-N-233x		44/12.5	
CF: ICP_DAS LP-8x81 8GB, pSLC		47/44	
Flash Disk IDE44: Kontron chipDISK/1000-IDE		3/5.7	
SD, MiniSD, MicroSD, MMC. Typical environment. 30°C, Read/Write: raw 1GB by blocks 1MiB. Card Readers: [Grand X CRX45 USB 2.0 multicard reader; Realtec USB 2.0 multicard reader; Transcend Multi-Card Reader M3; GoodRam MicroSD USDRSGRBL10; Raspberry Pi2 MicroSD for EXT4]			
MicroSD: Transcend 8GB Class 2, 378010	7.94/7.40	[14.6/3.4; 9/3.5; 14.3/3.7; 14.3/3.7]	
MicroSD: EMTEC 8GB Class 4	7.96/7.41	[18.0/0.8; 12.3/1.8; 17.2/1.8; 17.0/1.9]	early and apparently overheat and hang in long time on write
MicroSD: Kingston 8GB Class 4, SDC4/8GB	7.74/7.21	[16.7/0.756; 15.1/1.8; 16.4/1.8; 16.5/1.9]	early and apparently overheat and hang in long time on write
MicroSD: Transcend 8GB Class 4, A31213	7.94/7.40	[17.8/4; 18.0/4.1; 18.8/4.2; 17.9/3.0]	
MicroSD: Transcend 32GB Class 4, 9161BA		19.7/6.7	
MicroSD: Transcend 4GB Class 6, 9153BA	4.03/3.75	[15.7/6.7; 15.9/7.6; 18.5/9.3; 17.9/9.5]	
	8.03/7.47		

SD: Team 8GB Class 10, CT8G02XTVCC1118N		[16.7/10.3; 14.1/12.2; 18.5/14.0]
MicroSDHC: Toshiba 16GB UHS-1 Class 10, SD-C016UHS1(BL5A)	15.7/14.6	[16.2/5.9; 16.7/7.4; 18.4/9.1; 17.6/8.8; 16.9/11.3]
MicroSDHC: Toshiba 16GB UHS-1 Class 10, SD-C016UHS1(6A)	15.5/14.5	[14.4/6.7; 17.3/8.1; 18.6/9.9; 17.9/10.0; 19.5/13.0]
MicroSDHC/MicroSDXC: Kingston 16GB UHS-1 Class 10, SDC10G2/16GB	15.5/14.4	[18.6/6.0; 16.3/6.7; 18.5/10.5; 18.1/10.1; 17.1/8.6]
MicroSDHC: Transcend 16GB UHS-1 Class 10, C93858	15.9/14.8	[17.2/10.6; 18.6/9.6; 18.5/11.1; 17.7/10.8; 22.5/11.1]
MicroSDHC: SP 32GB, SP032GBSTH010V10	31.1/28.9	[18.1/8.0; 14.8/7.3; 18.7/10.8; 17.9/9.8; 19.4/11.1]
MicroSDHC: SanDisk Ultra 32GB, SDSQUNB-032G-GN3MN	31.1/28.9	[17.1/7.0; 14.8/7.5; 18.4/11.0; 18.1/10.4; -]
MicroSDHC: Transcend 32GB UHS-1 Class 10, Premium 400x D24035	31.7/29.5	[17.6/14.7; 17.8/9.6; 18.8/9.9; 17.8/14.1; 22.4/10.5]

8 PLC solutions

This section contains information about PLC models actually built or planned to be so on the basis of the developed runtime and PLC firmware.

PLC components	Price (DDP), $	Notes
PC/104		
CPU: Kontron MOPSlcdLX	430	AMDGeodeLX800(i686)-500MHz, 0°-60°C, 5W, Video
CPU: Diamond ATHM800-256A	1229	VIA Mark(i686)-800MHz, 256Mb, -40°-85°C, 10W, Video, 16AI, 4AO, 24DIO
CPU: Diamond ATHM800-256N	842	VIA Mark(i686)-800MHz, 256Mb, -40°-85°C, 10W, Video
CPU: Rhodeus RDS800-LC	414	AMDGeodeLX800(i686)-500MHz, -20°-70°C, 5W, Video
CPU: Helios HLV800-256AV	772	Vortex86DX(i486)-800MHz, 256Mb, -40°-85°C, 5.4W, Video, 16AI, 4AO, 40DIO
CPU: Helios HLV800-256DV	387	Vortex86DX(i486)-800MHz, 256Mb, -40°-85°C, 4.5W, Video

CPU: Tri-M VSX104	380	Vortex86SX(i486sx)-300MHz, 128Mb, -40°-85°C, 2W
MEM: DDR-SODIM-256M	15	for Kontron MOPSlcdLX
Flash Disk: Kontron chipDISK/1000-IDE	100	1Gb, 0°-70°C, read=3MB/s, write=5.7MB/s
Flash Disk: M-Systems MD1171-D1024	42	1Gb, 0°-70°C
Flash Disk: M-Systems MD1171-D256	22	256Mb, 0°-70°C
Flash Disk: M-Systems MD1171-D128	18	128Mb, 0°-70°C
Flash Disk: Diamond systems FD-128R-XT	82	128Mb, -40°-85°C
Flash Disk: Diamond systems FD-1GR-XT	168	128Mb, -40°-85°C
Box: PB-300-K	108	
Box: PB-EAP-300-K	250	
Box: CT-4	156	
Power unit: MMEANWELL DR-4505	30	
IO: DMM-16-AT (16AI, 4AO, 16DIO)	581	-40°-85°C
IO: DMM-32X-AT (32AI, 4AO, 24DIO)	689	-40°-85°C
RS485: EMM-OPT4-XT	396	-40°-85°C
RS232->RS485	10	
ICP DAS *LP-8x81*		
CPU: LP-8381	974	AMDGeodeLX800(i686)-500MHz, -25°-75°C, 14W, 4GB flash (R/W: 8/4 MB/s), 8GB CF (R/W: 29/19 MB/s), 1GB SRAM, Video, 2xEthernet, 2xUSB, 3-slots, 2xRS-232, 1xRS-485, 1xRS-232/485
CPU: LP-8781	1025	AMDGeodeLX800(i686)-500MHz, -25°-75°C, 16W, 4GB flash (R/W: 8/4 MB/s), 8GB CF (R/W: 29/19 MB/s), 1GB SRAM, Video, 2xEthernet, 2xUSB, 7-slots, 2xRS-232, 1xRS-485, 1xRS-232/485
CPU: LP-8781-Atom	1438	IntelAtomZ520-1.3GHz, -25°-75°C, 18W, 8GB flash, 1GB DDR2, Video, 2xEthernet, 4xUSB, 7-slots, 2xRS-232, 1xRS-485, 1xRS-232/485
IO_BOX: I-87K9	155	IO box for 9 modules series I-87k accessible by DCON
IO: I-8017HW (8AI DE, 16AI SI)	230	Parallel bus, acquisition up to the 30 kHz
IO: I-8042W (16DI + 16DO)	121	Parallel bus.
IO: I-87017ZW (20/10AI)	209	Serial bus. Overvoltage support up to 240V.
IO: I-87019RW (8AI)	213	Serial bus. Additional surge protection, support for thermocouples and resistance thermometers.

IO: I-87024W (4AO)	204	Serial bus. Output of current and voltage.
IO: I-87026PW (6AI, 2AO, 2DI, 2DO)	215	Serial bus. Combined module.
IO: I-87040W (32DI)	121	Serial bus. Isolated.
IO: I-87041W (32DO)	109	Serial bus. Isolated. Watchdog function for communication.
IO: I-87057W (16DO)	82	Serial bus. Watchdog function for communication.
Segnetics *SMH 2Gi*		
CPU: SMH 2Gi-0020-31-2	335	ARM9-200MHz, LCD-display, 1xRS-485, 1xRS-232, 2xUSB, 1xEthernet, 3DO
IO: MC-0402-01-0 (8AI, 4AO, 9DI, 10DO)	176	Single MC RS-485 serial bus.
IO: MR-120-00 (12DI[opt])	92	Multiple MR RS-485 serial bus.
IO: MR-800-00 (8DO[rel])	103	Multiple MR RS-485 serial bus.
IO: MR-810-00 (8DI[opt,~])	82	Multiple MR RS-485 serial bus.
IO: MR-061-00 (6DO[sim,opt])	84	Multiple MR RS-485 serial bus.
IO: MR-602-00 (6DO[rel], 2AO[opt])	120	Multiple MR RS-485 serial bus.
IO: MR-504-00 (5DO[rel], 4AO[opt])	120	Multiple MR RS-485 serial bus.