

Exor PLC+HMI Development Kit User Manual

UM0012 (v1.8) – 12 Sep 2018

User Manual:

Overview

This document will guide user through the installation and usage of Linux Yocto BSP for Exor uSOM and JMobile portable runtime.

The reproduction, transmission or use of this document or its contents is not permitted without express written authority. Offenders will be liable for damages. All rights, including rights created by patent grant or registration of a utility model or design, are reserved. Technical data subject to change. All trademarks and trade names appearing in this document are property of their respective owners. Copyright © 2015 Exor International SpA-Verona-Italy, All Rights Reserved. **Disclaimer** Exor International SpA is providing this design, code, or information "as is." basis, without warranty of any kind, either expressed or implied, including, without limitation, warranties that the covered code is free of defects, merchantable, fit for a particular purpose or non-infringing. Each party bears the entire risk as to the quality and performance of the original code, upgraded code, and modifications, to the extent originating with and provided by such party. Should any covered code prove defective in any respect, you assume the cost of any resulting damages, necessary servicing, repair or correction. This disclaimer of warranty constitutes an essential part of this license. No use of any covered code is authorized hereunder except subject to this disclaimer.

Table of Contents

1.	Getting started	5
1.1	Running the VirtualBox VM.....	5
1.1.1	Setup a guest-host shared folder.....	6
1.2	Running the Docker image	7
1.2.1	Running the Docker image under Linux.....	7
1.2.2	Running the docker image under Windows	9
1.3	Using the development environment.....	12
1.3.1	Using QtCreator	13
1.3.2	Compiling the BSP with Yocto	13
2	The Sato desktop	14
2.1	Network configuration	14
2.2	Start JMobile from Sato	15
3	Compiling Yocto BSP from scratch.....	16
3.1	Setup the build environment	16
3.2	Optional customizations.....	16
3.3	Compiling Yocto BSP	17
3.3.1	Building core-image-exor-wayland for us01-kit	18
3.3.2	Creating the SDK (optional).....	18
4	BSP deploy on SD-card.....	19
4.1	Using a ready image.....	19
4.1.1	Under Linux.....	19
4.1.2	Under Windows	19
4.2	Using the SD-card installer (Linux users only)	20
4.3	Manually.....	20
5	BSP deploy on eMMC	22
6	Setup the workspace for building applications	24
6.1	Cross development environment setup.....	24
6.2	Connecting to the device	24
6.3	QtCreator setup	25
7	Using Expansion Plugins	31
7.1	Use PLCM01 plugin (Canbus)	31
7.1.1	Plugin connection	31
7.1.2	System configuration and Plugin use.....	31

7.1.3	Canbus connector (CN2).....	32
7.2	Use PLCM04 module (RS-422/485)	32
7.2.1	Plugin connection	32
7.2.2	System configuration and Plugin use.....	33
7.2.3	Exaple C code	33
7.2.4	RS485 connector (CN2)	35
7.3	Use PLCM05 module (Expansion module)	35
7.3.1	SPI Plugin connection	36
7.3.2	SPI System configuration and plugin use	36
7.3.3	SPI Example C code	36
7.3.4	CN4 Connector.....	38
8	Upgrade FPGA firmware (us02-kit only)	39
9	JMobile Portable runtime	40
9.1	JMobile portable runtime installation	40
9.2	JMobile Studio quick start guide	41
9.2.1	Creating a simple project.....	41
10	CODESYS V3.....	46
10.1	Installing CODESYS Devices.....	46
10.2	Creation of a new PLC project	46
10.3	Communication setup in the CODESYS software	47
11	Accessing PLC from JMobile	49
11.1	Codesys project creation	49
11.2	CDS3 protocol configuration on JMobile	52

Version History

Version	Release date	Changes
1.3	22/07/2015	<ul style="list-style-type: none"> Added section 1.3, briefly explaining how to start building from the Docker container. Section 3.3: solved an inconsistency between paths in build instructions.
1.4	4/04/2016	<ul style="list-style-type: none"> Updated references to new SD card image and JMobile 2.1 Chapter 4: add note on supported SD-cards. A capacity of at least 4GB is required. Chapter 6: add note clarifying the purpose of the serial port, serial protocols are not supported.
1.5	24/05/2016	<ul style="list-style-type: none"> Add references to usom01 and usom03 based development kits. Chapter 1: add VirtualBox virtual machine references. Chapter 3: update to use Yocto 2.1 recipes.
1.6	5/09/2016	<ul style="list-style-type: none"> Chapter 1: add instructions for installing VirtualBox VM from OVA. Installing Vagrant is no more required. Chapter 5: add instructions for BSP deploy on eMMC Chapter 6: add some more informations, section 6.3 has now more detailed instructions. Chapter 7: add instructions for expansion plugins
1.7	30/06/2017	<ul style="list-style-type: none"> Chapter 4: update instruction for partitioning the SD-card
1.8	27/09/2017	<ul style="list-style-type: none"> Add references to nsom01 based development kit. Chapter 3: update to use Yocto 2.3 recipes

1. Getting started

To work with the development kits a Linux operating system with a properly configured build environment is required. The simplest way to get started, especially for Windows users, may be using one of our development virtual machines. We provide a VirtualBox VM and a Docker container, both are preconfigured with:

- Yocto workspace for building the BSP
- Preinstalled SDKs to start building your own application for the development kit
- QtCreator IDE with preconfigured target toolchains (Qt 5.9)

If you are already working on a Linux machine or you already have a Linux VM you may consider configuring yourself the build environment instead. In this case skip this chapter and go to chapter 3 if you are interested in building the BSP or chapter 6 if you are interested in building your own applications for the target.

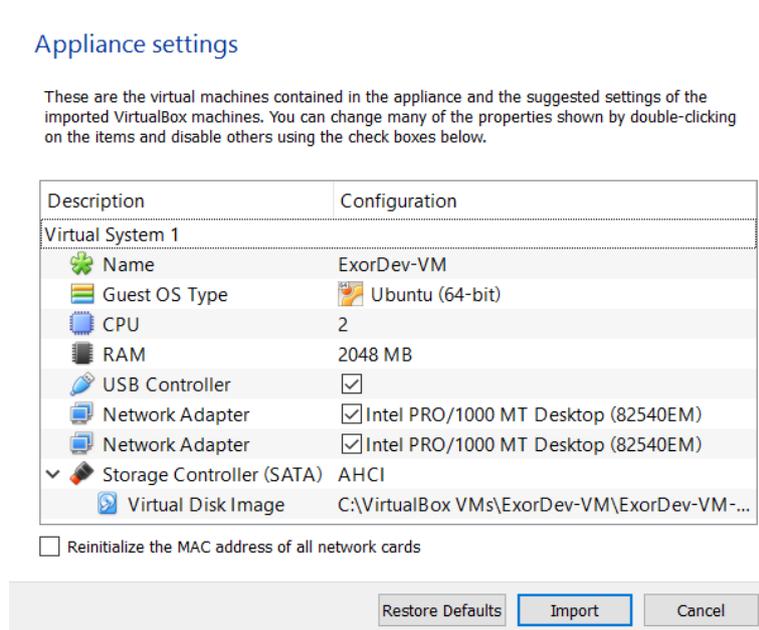
1.1 Running the VirtualBox VM

You can download the Exor’s VirtualBox development VM from here:

<http://download.exoreembedded.net:8080/Public/VirtualBoxVMs>

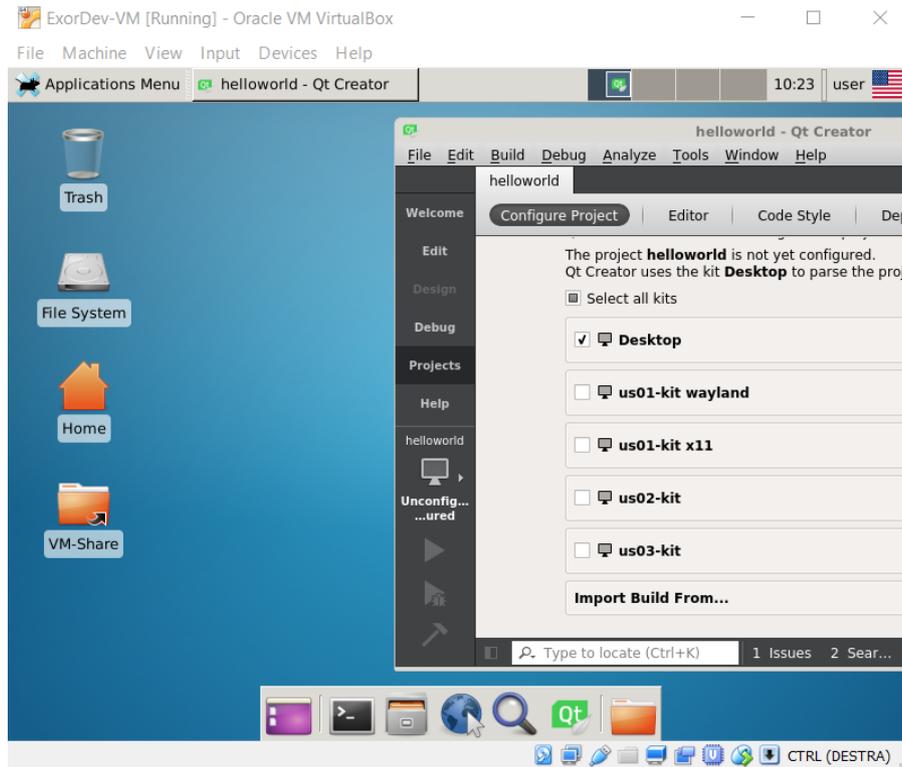
Instructions found on this document are compatible with versions 3.x of the VM. If you are about to use a greater version please consider looking for an updated version of this manual.

The virtual machine comes in the OVA (Open Virtualization Archive) format. To import it on VirtualBox got to “File” -> “Import Appliance...”, select the downloaded .ova file and then click “Import”. At this point VirtualBox will give you the opportunity to customize the VM, double-click on entries to edit them.



You will notice there are two network adapters, one is set to work in NAT mode while the second one works in bridged mode, the virtual machine will always use the bridged interface if possible and fall back to the other only if necessary. Adjust both adapters to work with the real network interface you use to have access to internet. Note that if the bridged adapter is not correctly configured you won't be able to resolve the Kit hostname, its IP address has to be used in this case.

The default amount of RAM is set to 2GB but if you plan working with Yocto we recommend to increase it to at least 4GB (suggested 6GB), adjusting the number of CPU cores is also a good idea. When you're done click on "Import". Once finished importing you will be able to change VM settings again.

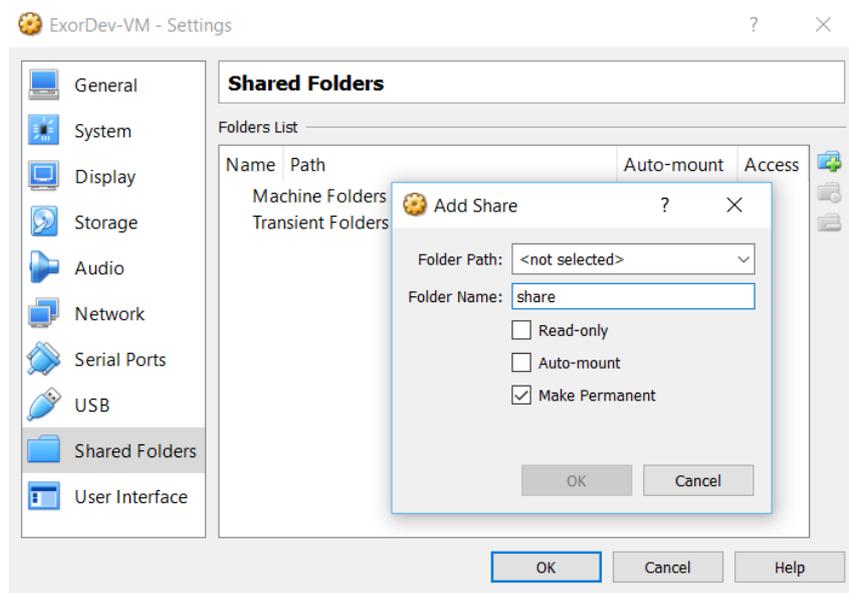


1.1.1 Setup a guest-host shared folder

We recommend configuring a shared folder between host and guest, it's the easiest way to move files from and to the VM. From VirtualBox right-click on Exor's VM and select "Settings...". Now go to "Shared Folders" and click on the add button to the right. Configure as follow:

- **Folder Path:** choose the host folder to share with the virtual machine
- **Folder Name:** must be `share`.
- **Read-only:** leave unchecked.
- **Auto mount:** leave unchecked.
- **Make Permanent:** set checked.

The chosen folder will be available inside the virtual machine from `/home/user/VM-Share`, a link to this location can be also found on the VM's desktop.



1.2 Running the Docker image

We also provide the Docker image `exoreembedded/docker-exor`, which is publicly available from DockerHub.

Docker supports many operating systems and Linux distributions. For more details or for installation on a different OS/Linux distribution, please refer to <https://docs.docker.com/installation/>.

1.2.1 Running the Docker image under Linux

Installing Docker on Linux can be done by using apt-get. On Ubuntu systems:

```
$ sudo apt-get install docker.io
```

In order to allow a non root user to run Docker add it to the "docker" group:

```
$ sudo usermod -aG docker <your_user_name>
```

You may need to logout and login again for the above changes to be applied.

Download the image from DockerHub using the following command:

```
$ docker pull exoreembedded/docker-exor
```

Finally create a new container named `exor` running the image:

To exit the Docker container press `Ctrl+D` from the container console or run `docker stop exor` from a host console. To re-attach to an existing container run:

```
$ docker start -ai exor
```

You can view container status with command `docker ps -a`. You can remove previous container using command `docker rm exor`.

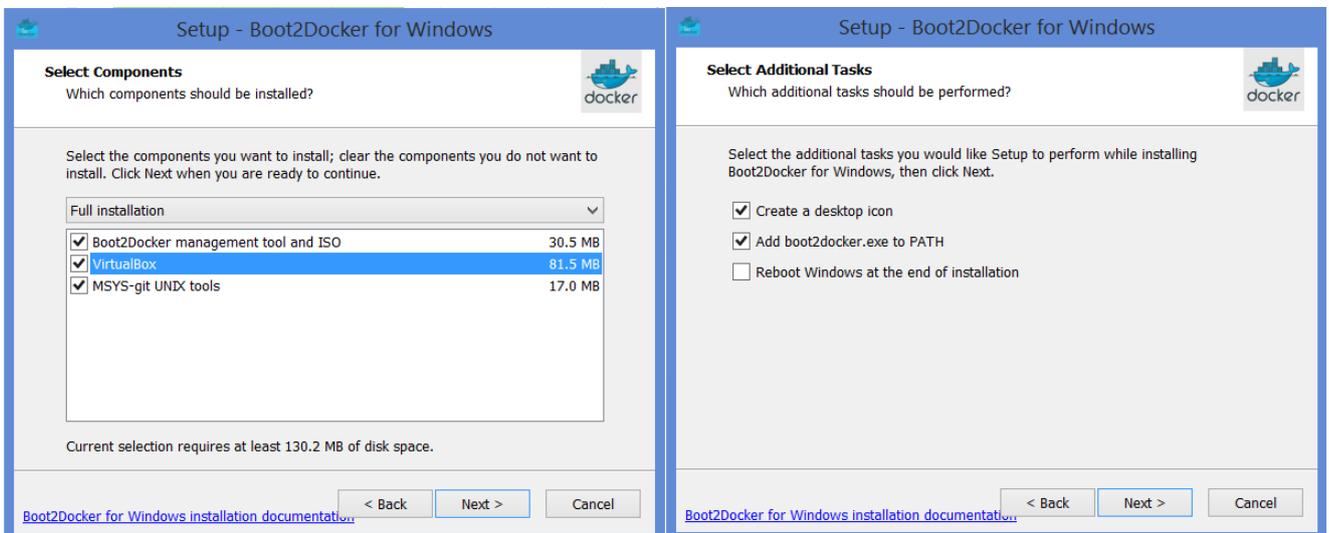
All data/settings not shared using an external volume with `-v` option will be removed when the container is removed.

If you want to expose to host a container folder, the home folder for instance, and make it persistent across container removal you can use the following command line to initialize the container : `docker run -v /home/user ...`. This way the shared folder can be accessed from the host environment, its location can be resolved by inspecting the container: `docker inspect us02 | grep /home/user/ | grep vfs`. The volume folders are typically put inside `/var/lib/docker/vfs/dir/`.

1.2.2 Running the docker image under Windows

Docker can run under Windows OS, provided that a small Linux virtual machine is set up to run the Docker daemon. Follow these steps for the installation:

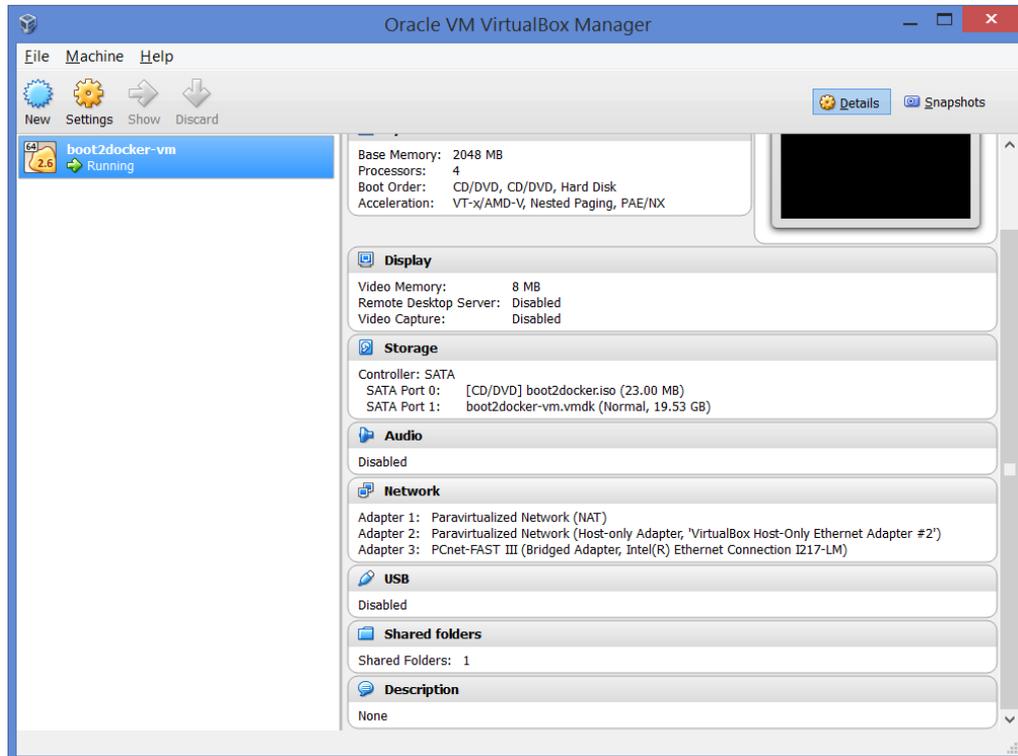
1. Download the latest release of boot2docker (<http://boot2docker.io>).
2. Run the installer, which will install Docker Client for Windows, VirtualBox, Git for Windows (MSYS-git), the boot2docker Linux ISO and the Boot2Docker management tool.



3. It is recommended to initialize the boot2docker-vm with at least 150000 MB of space in the hard drive. To do this run the following command from Git bash:

```
$ boot2docker --disksize=150000 init
```

A “boot2docker-vm” virtual machine is created, you are free to customize it to your needs. For instance, in order to benefit the IP resolution service provided by the development kit you have to add to the boot2docker virtual machine a network interface bridged to the network interface you are using to connect to the panel, as shown in figure below:

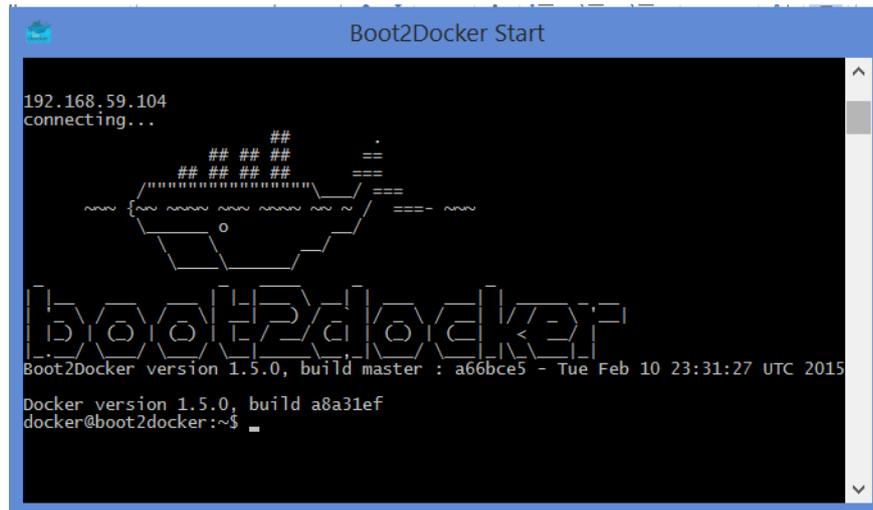


The bridged network interface must be able to obtain a valid IP address from a DHCP service.

To take advantage of the E17 desktop environment running on the Docker container you need to start boot2docker from Git bash by using the following command:

```
$ boot2docker.exe ssh -L 5555:localhost:5555
```

Where `-L 5555:localhost:5555` is needed to forward the Boot2Docker virtual machine port 5555 to the host machine 5555 port. This will expose the VNC service to the host.



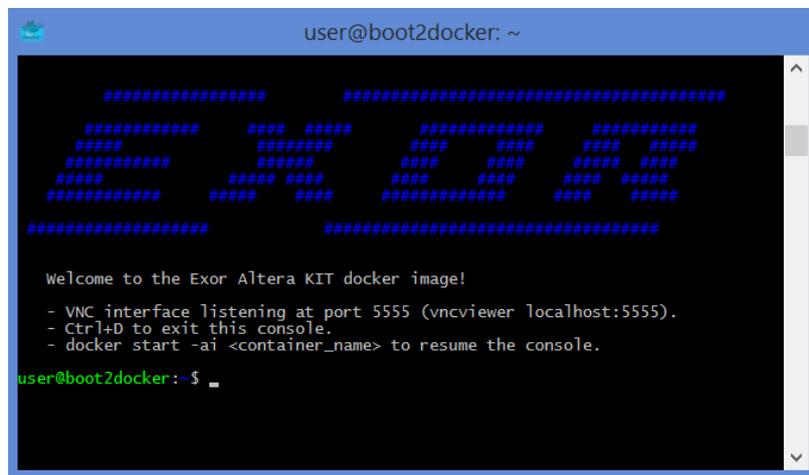
Inside the boot2docker environment you can use Docker just like explained in section 1.2.1.

```
$ docker run --name exor --net host -v /c/Users/Default:/home/user/share -ti
exoreembedded/docker-exor
```

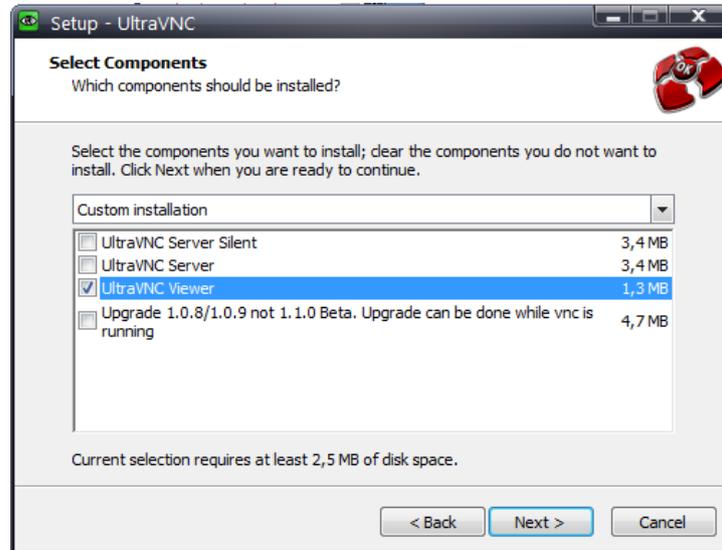
The boot2docker virtual machine by default shares the `/c/Users/Default` with the host, so you can bind that folder to the container's folder `/home/user/share`. Also in this case you can manually edit VirtualBox settings to share a different folder.

This step can take some time to download all the required container data

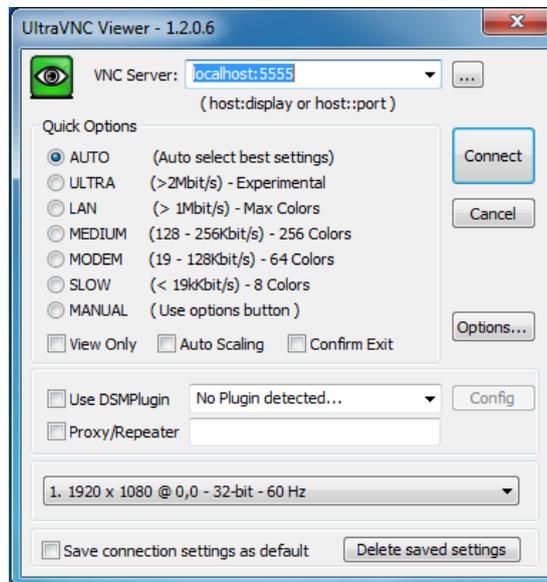
Once finished you will see the welcome screen from the container's terminal.



Now, to connect to the VNC server, Windows users can use a simple client as UltraVNC (http://www.uvnc.com). As shown in the following figure, for our purpose, only the “Viewer” is needed.



Finally, open UltraVNC Viewer and connect to localhost:5555:



1.3 Using the development environment

If you chose to use our VirtualBox or Docker image here are some information on how to use the environment. The Linux system used is based on Ubuntu 14.04, the default user is:

- username: **user**
- password: **password**

To run a command with root privileges you can use `sudo`, entering the password is not required.

1.3.1 Using QtCreator

The QtCreator IDE is already installed and configured to deploy and debug applications for each development kit. When creating a new project make sure to select the kit configuration for your device. For the us01-kit there are two configurations (x11 and wayland), the development kit is shipped with the x11 image preloaded on the SD-card, if you have not changed it select x11. There's also a "Desktop" kit configuration which you can use to build your application and run it on the virtual machine instead of deploying it, useful for fast testing and heavy profiling.

You will find a helloworld sample project in `/home/user/helloworld`, open it with QtCreator, compile it for your platform and press `Ctrl+R`, a window will pop up in the development kit.

You can find more details about configuring QtCreator in section 6.3, in particular how to change the hostname or IP address of the target device.

1.3.2 Compiling the BSP with Yocto

Inside `/home/user/exor-yocto-3.0` you will find the preconfigured Yocto workspace for building the BSP for your development kit. As we do not update our development virtual machines as often we do with our Yocto recipes you may want to update the meta-exor layer before starting the build:

```
$ cd /home/user/exor-yocto-3.0/git/meta-exor
$ git pull
```

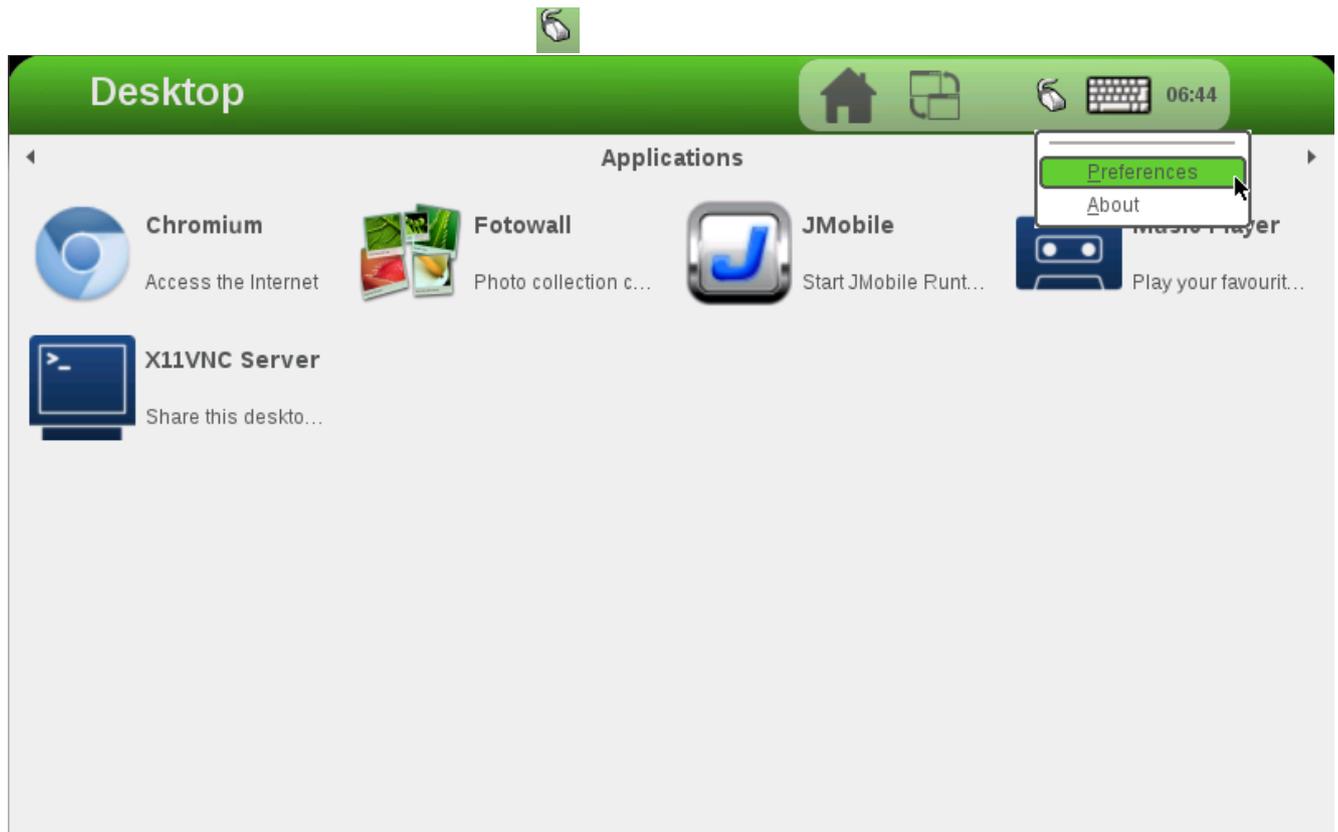
Go to chapter 3 to go ahead compiling the BSP.

2 The Sato desktop

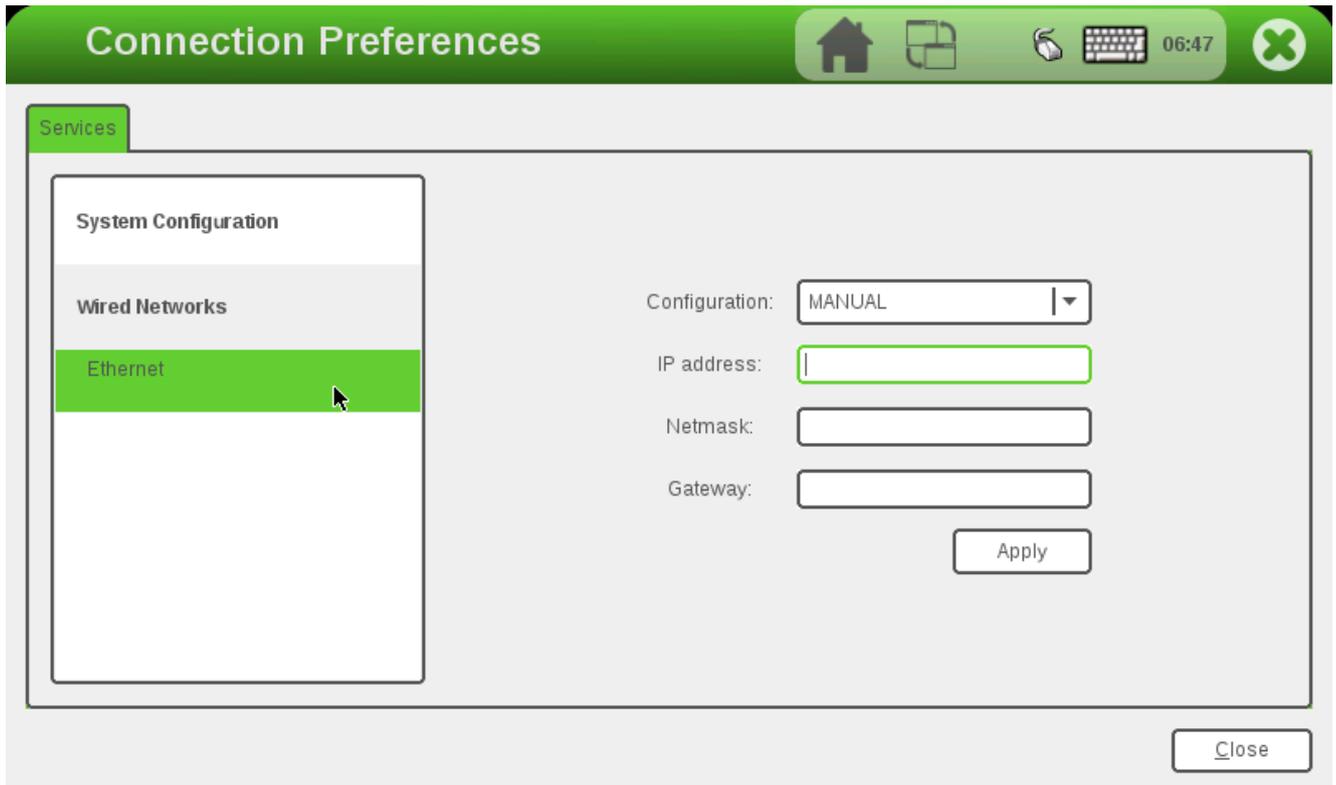
The development kit will boot with the default Yocto SATO interface. The machine is configured to run as **root** user and an empty password

2.1 Network configuration

By default network configuration is done using a DHCP service. To change this and set a static IP click on the Ethernet icon on the right top of the screen and choose "Preferences".



Here, in "Connection Preferences", choose "Ethernet" from the Services list and select "MANUAL" under "Configuration". Now you can fill in your network configuration. To do this you can toggle the on-screen keyboard or just plug-in a real USB keyboard. Remember to click on "Apply" when you are done.



2.2 Start JMobile from Sato

By default, among other applications, a portable version of JMobile Runtime is installed. To launch the HMI just click on the JMobile icon you will find in the “Applications” menu in Sato. A demo project is already loaded for evaluation purpose. As the Runtime is meant to run by his own on the system the Sato user interface will be terminated.

To close JMobile and return to Sato you can both reboot the board or kill the HMI by issuing the following command from an ssh session:

```
$ killall xinit
```

3 Compiling Yocto BSP from scratch.

3.1 Setup the build environment

If you are using Exor's Docker container or VirtualBox VM you can skip the first two steps: you will find the `exor-yocto-3.0` folder already in the user's home (`/home/user/exor-yocto-3.0`).

1. Create a workspace directory structure:

```
$ mkdir -p exor-yocto-3.0
$ cd exor-yocto-3.0/
```

2. Get the source code from github repositories:

```
$ curl http://commondatastorage.googleapis.com/git-repo-downloads/repo > repo
$ chmod a+x repo
$ ./repo init -u https://github.com/ExorEmbedded/exor-bsp-platform -b pyro
$ ./repo sync
```

3. Setup the Yocto environment. From the `exor-yocto-3.0` folder execute:

```
$ source git/yocto-poky/oe-init-build-env build
```

You should now find yourself in a newly created "build" directory located in `exor-yocto-3.0/build`. The source command above

4. Configure Yocto by copying the provided sample configuration files. From the the `build` directory:

```
$ cp ../git/meta-exor/conf/bblayers.conf.sample conf/bblayers.conf
$ cp ../git/meta-exor/conf/local.conf.sample conf/local.conf
```

5. Now edit your `conf/local.conf` and set the `MACHINE` variable to `us01-kit`, `us02-kit` (AlteraKit) or `us03-kit`. For example:

```
MACHINE = "us02-kit"
```

You are now ready to build the BSP.

3.2 Optional customizations

Here are some customizations you may be interested in:

- You can force Yocto to build a 32-bit SDK uncommenting the following line in the `build/conf/local.conf` file:

```
#SDKMACHINE ?= "i686"
```

- Uncomment following lines in the `build/conf/local.conf` file to be able to set the number of threads and CPU cores you want to use for the build process:

```
#BB_NUMBER_THREADS ?= "4"
#PARALLEL_MAKE ?= "-j 4"
```

- To compile chromium web browser add the following line to `build/conf/local.conf`:

```
LICENSE_FLAGS_WHITELIST="commercial"
```

Then uncomment this line in `git/meta-exor/recipes-images/images/core-image-exor.inc`

```
#CORE_IMAGE_EXTRA_INSTALL += "chromium"
```

3.3 Compiling Yocto BSP

Make sure to run following commands from your “build” folder:

1. Compile the bootloader:

```
$ bitbake bootloader
```

2. The xloader (for the us01 kit only):

```
$ bitbake xloader
```

3. The Linux kernel:

```
$ bitbake virtual/kernel
```

4. And finally the rootfs:

```
$ bitbake core-image-exor-x11
```

This will build the classic x11 sato image, the one that can be found in the SD-card included with the development kit.

At the end of these operations you will find build output files in `build/tmp/deploy/images/usom0X`:

<code>us0X-kit-uboot.tar.gz</code>	Contains the U-Boot raw image
<code>us0X-kit-xloader.tar.gz</code>	Contains the xloader raw image (usom01 only)
<code>us0X-kit-kernel.tar.gz</code>	Contains the kernel zImage and the dtb
<code>core-image-exor-[...]-us0X-kit.tar.gz</code>	Contains the rootfs
<code>us0X-kit-xloader.tar.gz</code>	Contains the xloader raw image (usom01 only)
<code>us0X-kit-kernel.tar.gz</code>	Contains the kernel zImage and the dtb
<code>core-image-exor-[...]-us0X-kit.tar.gz</code>	Contains the rootfs

3.3.1 Building core-image-exor-wayland for us01-kit

A Wayland image called `core-image-exor-wayland` is also available for the us01 kit. To build it, change Yocto `DISTRO_FEATURE` variable for the current distribution by making following changes to `git/meta-exor/conf/exorintos.conf`. Comment or remove this line:

```
DISTRO_FEATURES_remove = "wayland"
```

And uncomment this other one:

```
#DISTRO_FEATURES_remove = "x11"
```

To enable hardware acceleration, also these few more lines needs to be uncommented from the kit's configuration file `git/meta-exor/conf/machine/us01-kit.conf` :

```
#PREFERRED_PROVIDER_virtual/egl = "ti-sgx-ddk-um"
#PREFERRED_PROVIDER_virtual/libgles1 = "ti-sgx-ddk-um"
#PREFERRED_PROVIDER_virtual/libgles2 = "ti-sgx-ddk-um"
#PREFERRED_PROVIDER_virtual/libgl = "mesa-gl"
#PREFERRED_PROVIDER_virtual/mesa = "mesa-gl"
#PREFERRED_PROVIDER_libgbm = "libgbm"
```

Launch bitbake from the build folder:

```
$ bitbake core-image-exor-wayland
```

At the moment hardware acceleration on the us01 kit is available only with the Wayland image.

3.3.2 Creating the SDK (optional)

Start the SDK build for the x11 image:

```
$ bitbake -c populate_sdk core-image-exor-x11
```

While if working with the wayland image:

```
$ bitbake -c populate_sdk core-image-exor-wayland
```

The SDK installer can be found in `build/tmp/deploy/sdk/exor-evm-qt5-sdk.sh`.

4 BSP deploy on SD-card

This section describes how to prepare a bootable SD-card for the evaluation kit, for this remember that only SD-cards with at least 4GB of space are supported.

Also note that following operations can be dangerous, harm your system or cause loss of data. Do not blindly execute these operations if you don't know what they actually do.

For Linux users we will assume below the SD-card device is named `/dev/sdb` and its partitions `/dev/sdbX`, change these to the actual names.

4.1 Using a ready image

We provide a fully working 4GB image containing the x11-sato environment to let you start using the kit in no time. Note that by using this option, even with a more capable SD, only ~4GB of space will be available to the system.

Download the latest disk image for your evaluation kit:

```

US01kit images: http://download.exoreembedded.net:8080/Public/usom01/sdcard-images/
US02Kit images: http://download.exoreembedded.net:8080/Public/usom02/sdcard-images/
US03Kit images: http://download.exoreembedded.net:8080/Public/usom03/sdcard-images/
US01Kit images: http://download.exoreembedded.net:8080/Public/nsom01/sdcard-images/
  
```

4.1.1 Under Linux

From a Linux shell:

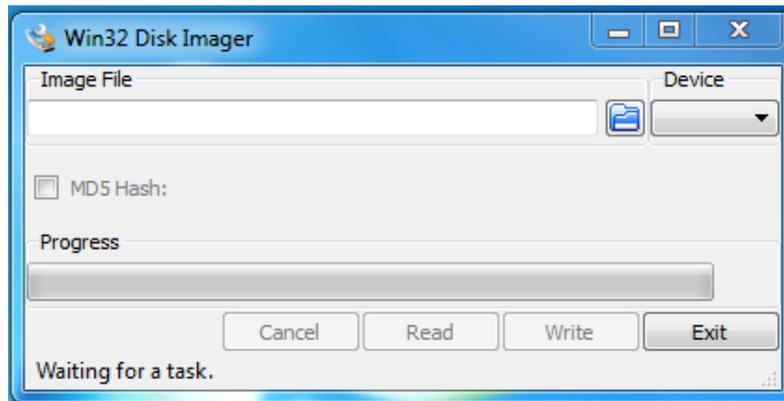
```

# unzip SDcard-image-4gb.zip
# dd if=SDcard-image-4gb.img of=/dev/sdb bs=64k
# sync
  
```

Your SD-card is now ready to be used on the development kit.

4.1.2 Under Windows

Download Win32DiskImager from <http://sourceforge.net/projects/win32diskimager/>. From the user interface of Win32DiskImager select the extracted .img image file and the SD-card drive and press "Write".



4.2 Using the SD-card installer (Linux users only)

If you want to take advantage of all your SD-card space or you have built by your own some components you want to deploy, it's also possible to use a SD-card installer script:

```
$ wget http://download.exoreembedded.net:8080/Public/Utils/mkSDTool/mkSDTool-v2.0.sh -O mkSDTool.sh
$ sudo chmod +x mkSDTool.sh
$ sudo ./mkSDTool.sh --machine [us01-x11|us01-wayland|us02|us03] --device /dev/sdb
```

By default the script will deploy on your SD-card all the necessary files by downloading the needed components from following locations:

US01kit:	http://download.exoreembedded.net:8080/Public/usom01/
US02kit (AlteraKit) :	http://download.exoreembedded.net:8080/Public/usom02/
US03kit:	http://download.exoreembedded.net:8080/Public/usom03/
NS01kit:	http://download.exoreembedded.net:8080/Public/nsom01/

If you want to provide yourself one or more of these components you can take advantage of following options supported by the mkSDTool script:

- --rootfs <path/to/my/rootfs.tar.gz>
- --kernel <path/to/my/kernel.tar.gz>
- --uboot <path/to/my/u-boot.tar.gz>
- --xloader <path/to/my/xloader.tar.gz> (only for us01)

4.3 Manually

1. Create the SD-card partition layout :

```
# umount /dev/sdb*
# SIZE=`fdisk -l /dev/sdb | grep Disk | awk '{print $5}'`
# CYLINDERS=$(( (($SIZE)) / 255 / 63 / 512 ))
# sfdisk --force -D -H 255 -S 63 -C $CYLINDERS /dev/sdb << EOF
1,5
6,$(($CYLINDERS - 10 ))
$(($CYLINDERS - 4)),,a2
EOF
# mkfs.vfat -n BOOT /dev/sdb1
```

```
# mkfs.ext4 -L ROOT /dev/sdb2
```

2. Mount partitions. Execute following operations:

```
// Mount partitions if not already mounted

# mkdir /media/BOOT
# mount /dev/sdb1 /media/BOOT
# mkdir /media/ROOT
# mount /dev/sdb2 /media/ROOT
```

Now the actual deploy phase depends on the specific board. Make sure to follow the appropriate steps:

```
// Deploy files to SD-card for us01-Kit
# mkdir /media/BOOT/boot
# tar xzvf us01-kit-kernel.tar.gz --no-same-owner -C /media/BOOT/boot
# tar xzvf us01-kit-bootloader.tar.gz --no-same-owner -C /media/BOOT
# tar xzvf us01-kit-xloader.tar.gz --no-same-owner -C /media/BOOT
# tar xzvf core-image-exor-[...]tar.gz -C /media/ROOT
# sync

// Deploy files to SD-card for us02-kit (AlteraKit)
# tar xzvf us02-kit-kernel.tar.gz -C /media/BOOT
# tar xzvf core-image-exor-[...]tar.gz -C /media/ROOT
# tar xzvf us02-kit-uboot.tar.gz
# dd if=u-boot.img of=/dev/sdb3 bs=64k seek=4
# sync

// Deploy files to SD-card for us03-kit
# tar xzvf us03-kit-kernel.tar.gz -C /media/BOOT/boot
# tar xzvf core-image-exor-[...]tar.gz -C /media/ROOT
# tar xzvf us03-kit-uboot.tar.gz
# dd if=u-boot.imx of=/dev/sdb bs=1k seek=1
# sync

// Deploy files to SD-card for ns01-kit
# tar xzvf ns01-kit-kernel.tar.gz -C /media/BOOT/boot
# tar xzvf core-image-exor-[...]tar.gz -C /media/ROOT
# tar xzvf ns01-kit-uboot.tar.gz
# dd if=u-boot.imx of=/dev/sdb bs=1k seek=1
# sync
```

5 BSP deploy on eMMC

To deploy the BSP to the internal eMMC it is required to define a partition layout and then modify the bootloader environment in order to inform the u-boot on where to look for all the necessary files. Here, for demonstration purposes, we will use the simplest layout, a single ext4 partition. Following instructions needs to be executed on the development kit via ssh, it requires you have a working SD-card and these files available on it:

- The bootloader image, `u-boot.img`.
- The rootfs, `core-image-exor.tar.gz`.
- Kernel and dtb or a `kernel.tar.gz` containing both.

Here are the steps to follow:

- 1) Reformat the eMMC device to have a single partition and create the ext4 filesystem. The eMMC device is defined as `build/dev/mmcblk1` on the us01-kit and us03-kit while it's `build/dev/mmcblk0` on the alterakit, for this reason the operation is slightly different between platforms.

```
// Format eMMC and mount rootfs partition for us01-kit and us03-kit
# umount /dev/mmcblk1p*
# echo -e "o\nn\np\n1\n\n\nw" | fdisk /dev/mmcblk1
# mkfs.ext4 /dev/mmcblk1p1
# mkdir emmc
# mount /dev/mmcblk1p1 emmc

// Format eMMC and mount rootfs partition for us02-kit (AlteraKit)
# umount /dev/mmcblk0p*
# echo -e "o\nn\np\n1\n\n\nw" | fdisk /dev/mmcblk0
# mkfs.ext4 /dev/mmcblk0p1
# mkdir emmc
# mount /dev/mmcblk0p1 emmc
```

- 2) Deploy rootfs and kernel. Make sure at the end `emmc/boot` contains both a zImage and a dtb.

```
# tar xzvf core-image-exor.tar.gz -C emmc
# tar xzvf kernel.tar.gz -C emmc/boot // Or just copy zImage and dtb to emmc/boot
# sync
```

- 3) Deploy the bootloader. Again, this is platform dependent.

```
// Deploy bootloader on eMMC for us01-kit
# echo 0 > /sys/block/mmcblk1boot0/force_ro
# dd if=u-boot.img of=/dev/mmcblk1boot0 bs=512 seek=0

// Deploy bootloader on eMMC for us02-kit (AlteraKit)
# echo 0 > /sys/block/mmcblk0boot0/force_ro
# dd if=u-boot.img of=/dev/mmcblk0boot0 bs=512 seek=0

// Deploy bootloader on eMMC for us03-kit and ns01-kit
# dd if=u-boot.img of=/dev/mmcblk1 bs=512 seek=2
```

Now if you remove the SD-card the bootloader written to the eMMC will be executed but the system won't boot because the u-boot will still look for files inside the SD-card.

To make it work the bootloader environment must be changed. To do this remove the SD-card, connect to the kit's serial port using a client like putty and while continuously pressing Ctrl+C on the console power off and then on the device. A prompt should appear.

From here execute these commands:

```
// U-boot environment changes for us01-kit
# setenv mmcroot /dev/mmcblk1p1
# setenv mmcloadcmd ext2load
# setenv bootimage /boot/zImage
# setenv fdtimage /boot/usom_eco.dtb
# setenv mmcload "mmc rescan;${mmcloadcmd} mmc 1:${mmcloadpart} ${loadaddr}
    ${bootimage};${mmcloadcmd} mmc 1:${mmcloadpart} ${fdtaddr} ${fdtimage}"
# saveenv

// U-boot environment changes for us02-kit (AlteraKit)
# setenv mmcroot /dev/mmcblk0p1
# setenv mmcloadcmd ext2load
# setenv bootimage /boot/zImage
# setenv fdtimage /boot/socfpga.dtb
# setenv mmcload "mmc rescan;${mmcloadcmd} mmc 0:${mmcloadpart} ${loadaddr}
    ${bootimage};${mmcloadcmd} mmc 0:${mmcloadpart} ${fdtaddr} ${fdtimage}"
# saveenv

// U-boot environment changes for us03-kit
# setenv mmcroot /dev/mmcblk1p1
# setenv mmcloadcmd ext2load
# setenv bootimage /boot/zImage
# setenv fdtimage /boot/usom_evkitq.dtb
# setenv mmcload "mmc rescan;${mmcloadcmd} mmc 1:${mmcloadpart} ${loadaddr}
    ${bootimage};${mmcloadcmd} mmc 1:${mmcloadpart} ${fdtaddr} ${fdtimage}"
# saveenv
```

To restore the bootloader's environment and boot again from SD-card stop the machine at the u-boot's prompt again and type:

```
# env default -a
# saveenv
```

6 Setup the workspace for building applications

This section describes how to setup a 64bit Linux PC or virtual machine to be able to build applications for the target development kit. Our virtual machine and our Docker image are already preconfigured and ready to use, these steps can be skipped when using one of these solutions.

6.1 Cross development environment setup

Download the latest SDK from here:

US01kit:	http://download.exoreembedded.net:8080/Public/usom01/SDK
US02kit (AlteraKit) :	http://download.exoreembedded.net:8080/Public/usom02/SDK
US03kit:	http://download.exoreembedded.net:8080/Public/usom03/SDK
NS01kit:	http://download.exoreembedded.net:8080/Public/nsom01/SDK

For the us01kit there's a x11 SDK and a wayland SDK, make sure to download the one matching the rootfs currently in use.

Execute the SDK installation file `exor-evm-qt5-sdk.sh` (requires admin privileges):

```
$ cd /opt
$ sudo chmod a+x ./ exor-evm-qt5-sdk.sh
$ sudo ./exor-evm-qt5-sdk.sh
```

You will be asked for the installation directory, press enter to use the default, `/opt/exorintos/2.3.2`. To setup the cross development environment for the current shell run this command (correct the path if you have changed the default installation directory):

```
// Environment setup for us01-kit
$ source /opt/exorintos/2.3.2/environment-setup-cortexa8hf-vfp-neon-poky-linux-gnueabi

// Environment setup for us02-kit and us03-kit
$ source /opt/exorintos/2.3.2/environment-setup-cortexa9hf-vfp-neon-poky-linux-gnueabi

// Environment setup for ns01-kit
$ source /opt/exorintos/2.3.2/environment-setup-cortexa7hf-vfp-neon-poky-linux-gnueabi
```

To build a simple hello world application use the arm cross compiler that should now be reachable from your PATH:

```
$ arm-poky-linux-gnueabi-gcc main.c -o hello_world
```

6.2 Connecting to the device

On each device a console is active over serial port for debugging purposes. An ssh server is also running, useful for having a shell over ethernet or transferring files via sftp. In both cases the username to use is `root`, no password is required.

If your system has an avahi client installed the kit can also be addressed by its hostname:

```

US01kit:          exorUS01kit.local
US02kit (AlteraKit): exoralterakit.local
US03kit:          exorUS03kit.local
NS03kit:          exorNS01kit.local
  
```

6.3 QtCreator setup

When developing Qt applications it may be useful to have the Qt IDE preconfigured to use the toolchain. You can get latest QtCreator package from DIGIA here:

```

http://download.qt.io/official\_releases/qtcreator/3.3/3.3.2/qt-creator-opensource-linux-x86-3.3.2.run
  
```

Install it in your machine:

```

$ sudo chmod a+x ./qt-creator-opensource-linux-x86-3.3.2.run
$ ./qt-creator-opensource-linux-x86-3.3.2.run
  
```

You will find qtcreator installed in `~/qtcreator-3.3.2`. Start it:

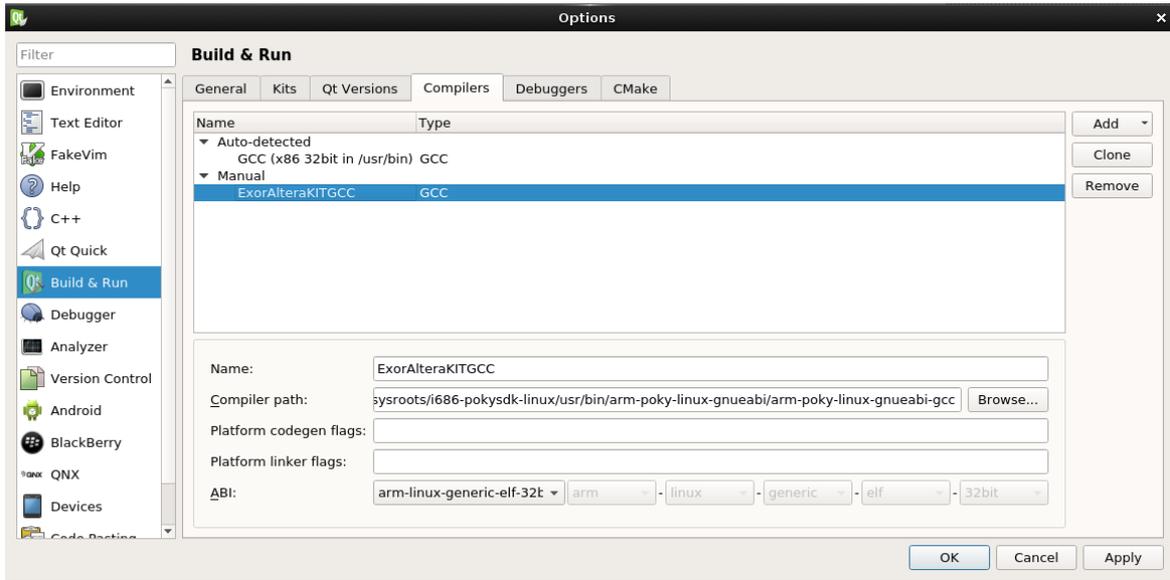
```

$ ~/qtcreator-3.3.2/bin/qtcreator
  
```

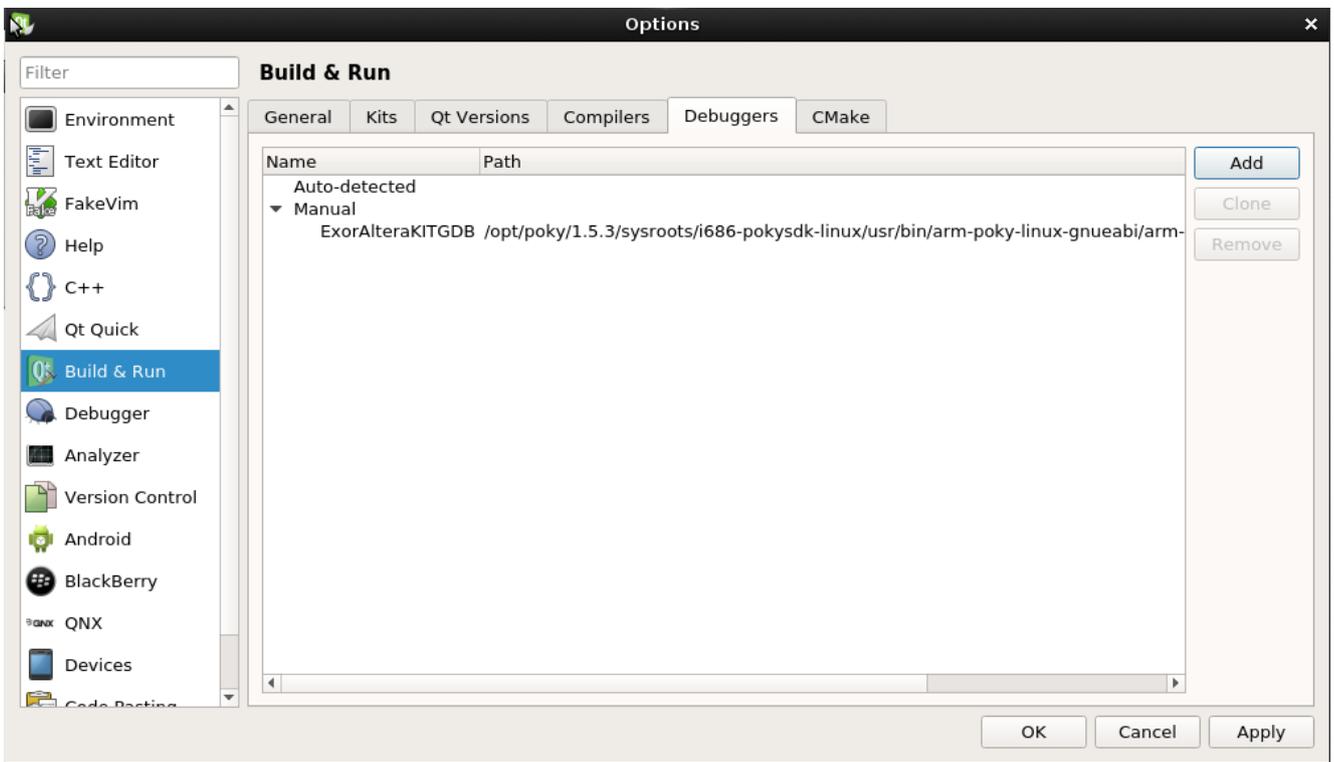
We are now going to setup the QtCreator build kit for the target.

From Tools menu select “Options...” -> “Build & Run”, then follow these steps:

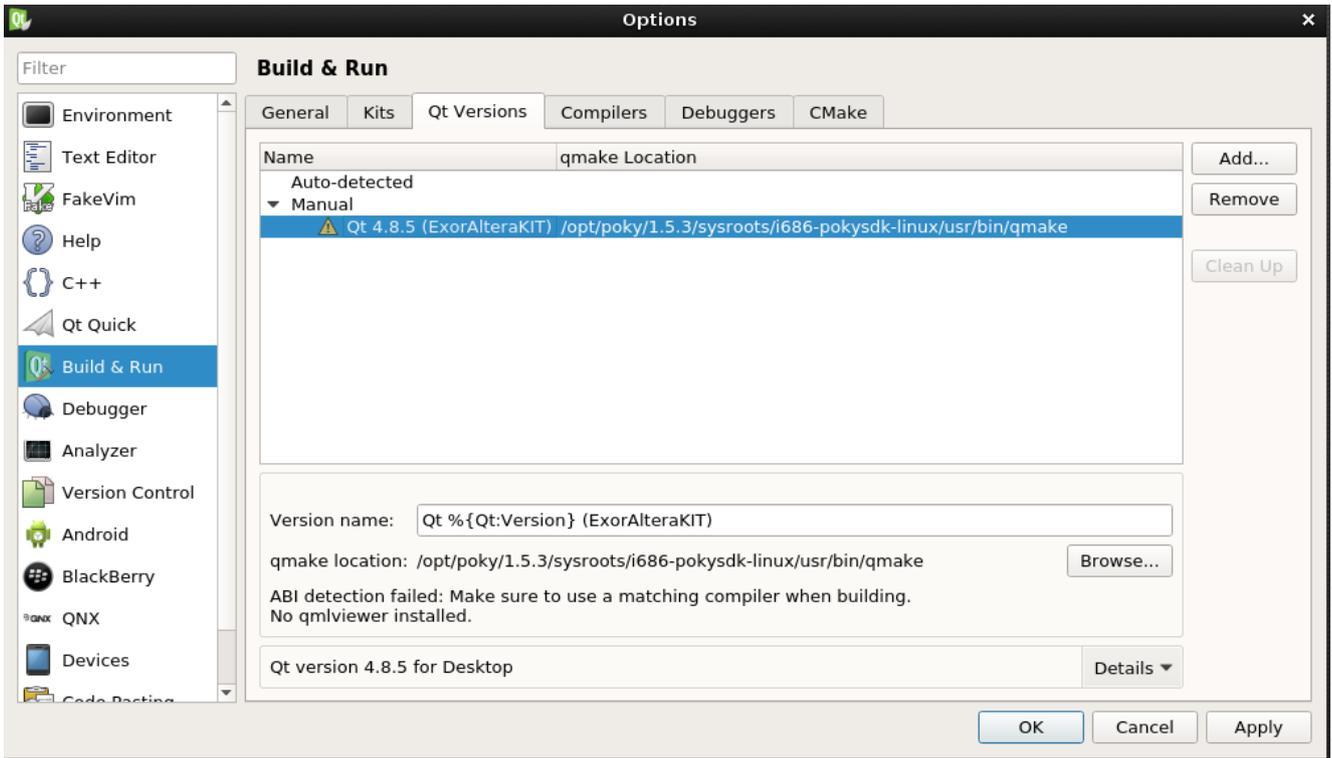
- 1) In the “Compilers” tab click on “Add” -> “GCC” and select the cross compiler picking it from the SDK installation folder. If the SDK has been installed in the default location the correct path is: `/opt/exorintos/2.3.2/sysroots/x86-64-pokysdk-linux/usr/bin/arm-poky-linux-gnueabi/arm-poky-linux-gnueabi-gcc`.
Optionally edit “Name” to give a more meaningful name for the entry, then click “Apply”.



- From “Debuggers” tab press “Add” and select gdb from the same directory. The default location is: `/opt/exorintos/2.3.2/sysroots/x86-64-pokysdk-linux/usr/bin/arm-poky-linux-gnueabi/arm-poky-linux-gnueabi-gdb` . Optionally edit “Name”, then click “Apply”.



- 3) From “Qt Versions” tab, press “Add..”. The default path to select is: `/opt/exorintos/2.3.2/sysroots/x86-64-pokysdk-linux/usr/bin/qmake`. QtCreator should automatically recognize the qt version selected. Press “Apply”.



4) Now move from “Build & Run” section to “Devices”. Click “Add..”, select “Generic Linux Device” and press “Start Wizard”. Fill in these informations:

- **Name:** the device name, for example, `us01-kit`.
- **Host name:**

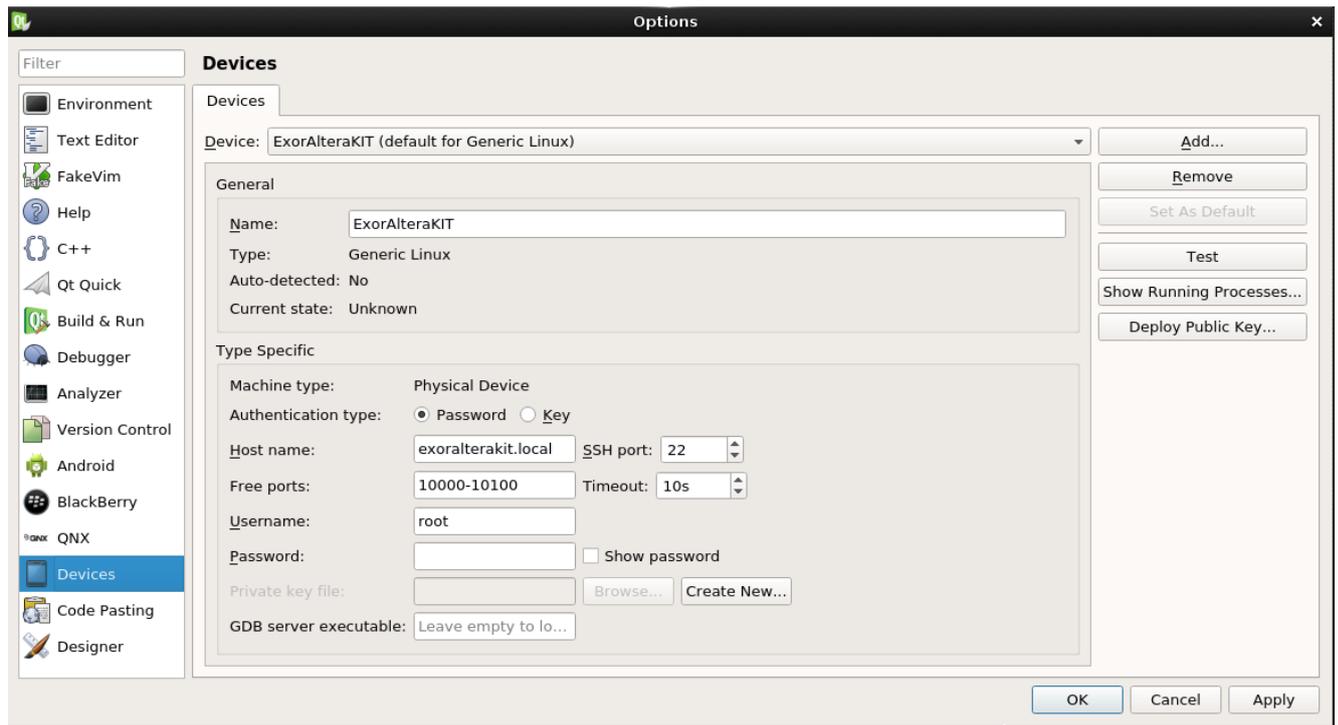
```

US01kit:          exorUS01kit.local
US02kit (AlteraKit) : exoralterakit.local
US03kit:          exorUS03kit.local
NS01kit:          exorNS01kit.local
    
```

- **Username:** `root`.
- **Authentication type:** set to “Password”.
- **User’s password:** leave empty, no password is needed.

Click “Next” and then “Finish”. Qt Creator will attempt a test connection, if the device is already powered on and reachable everything should be ok.

If for any reason you cannot reach the target by its hostname make sure avahi is installed on your system or edit “Host name” to set the actual board IP address instead. Press on “Test” button to check the connection again.



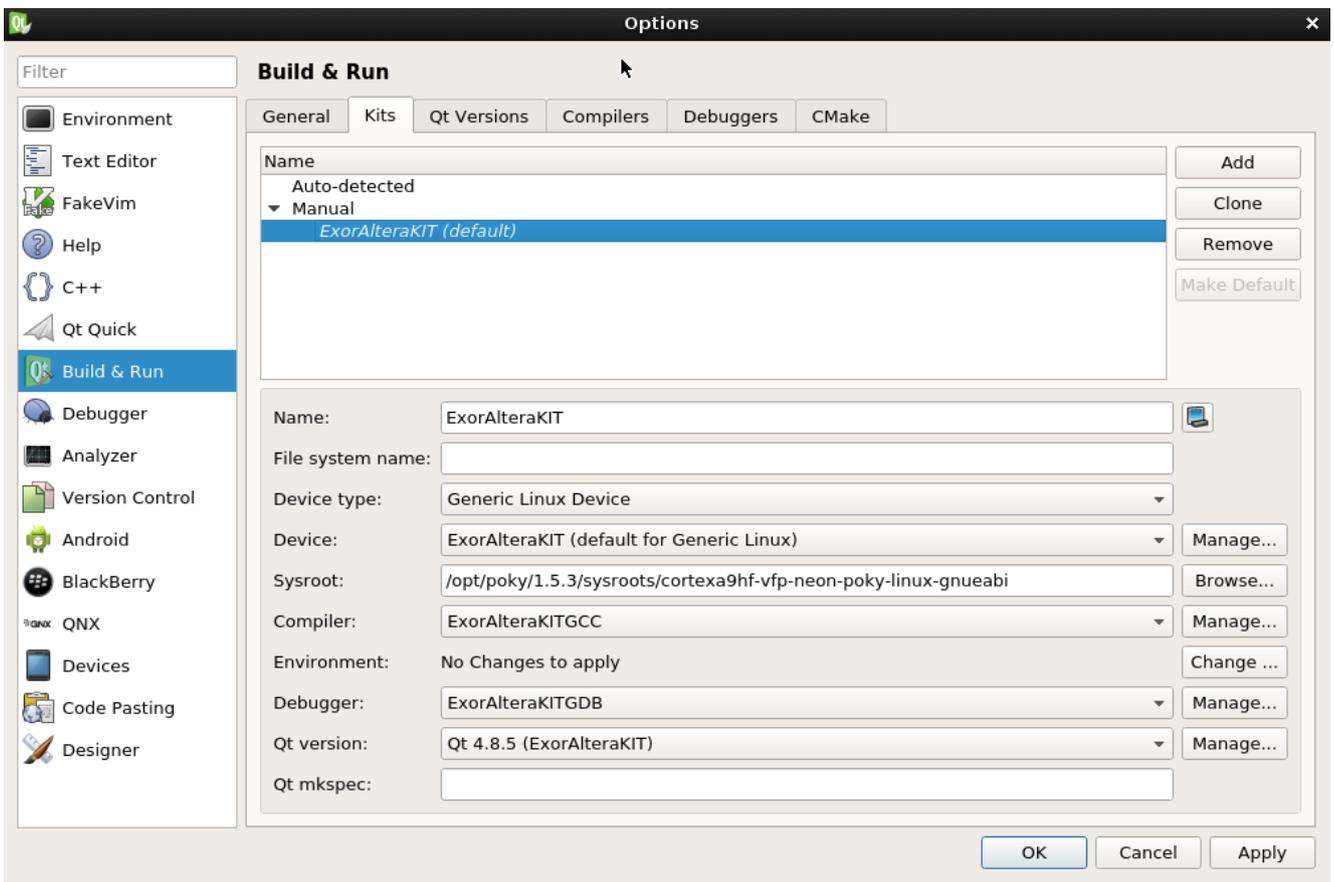
5) Finally move again to “Build & Run” section, “Kits” tab. Combine all pieces together in a new kit. Click “Add” and fill in as follows:

- **Name:** choose a name for the kit.
- **Device Type:** select “Generic Linux Device”.
- **Device:** select the device configured in 4).
- **Sysroot:** if the SDK is installed in the default location, these are the paths to select:

```

US01kit: /opt/exorintos/2.3.2/sysroots/cortexa8hf-neon-poky-linux-gnueabi
US02kit: /opt/exorintos/2.3.2/sysroots/cortexa9hf-neon-poky-linux-gnueabi
US03kit: /opt/exorintos/2.3.2/sysroots/cortexa9hf-neon-poky-linux-gnueabi
NS01kit: /opt/exorintos/2.3.2/sysroots/cortexa7hf-neon-poky-linux-gnueabi
    
```

- **Compiler:** select compiler by name as configured in 1).
- **Debugger:** select debugger by name as configured in 2).
- **Qt version:** select qt version added in 3).



7 Using Expansion Plugins

7.1 Use PLCM01 plugin (Canbus)

7.1.1 Plugin connection

The Plcm01 can be plugged in every plugin connector.

If you connect the Plcm01 on the connector “Plugin 1” the system provide the Can0 interface.
If you connect the Plcm01 on the connector “Plugin 2” the system provide the Can1 interface.



7.1.2 System configuration and Plugin use

Once connected the plugin you can power-up the development kit and wait the booting process.
With the following system command you can:

- Set the can interface

```
# ip link set can0 up type can bitrate 250000
```

- Enable the can interface

```
# ifconfig can0 up
```

- Send can packet

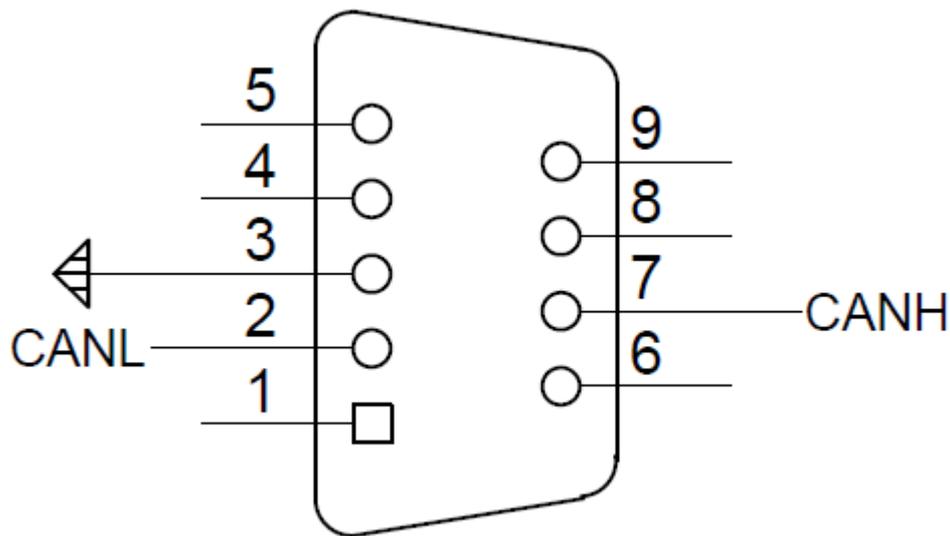
```
# cansend can0 -i 0x1A5 0x01 0x23 0x45 0x67 0x89 0xAB 0xCD 0xEF
```

- Received can packet

```
# candump can0
```

7.1.3 Canbus connector (CN2)

Plcm01 is equipped with a standard male DB9 canbus connector.



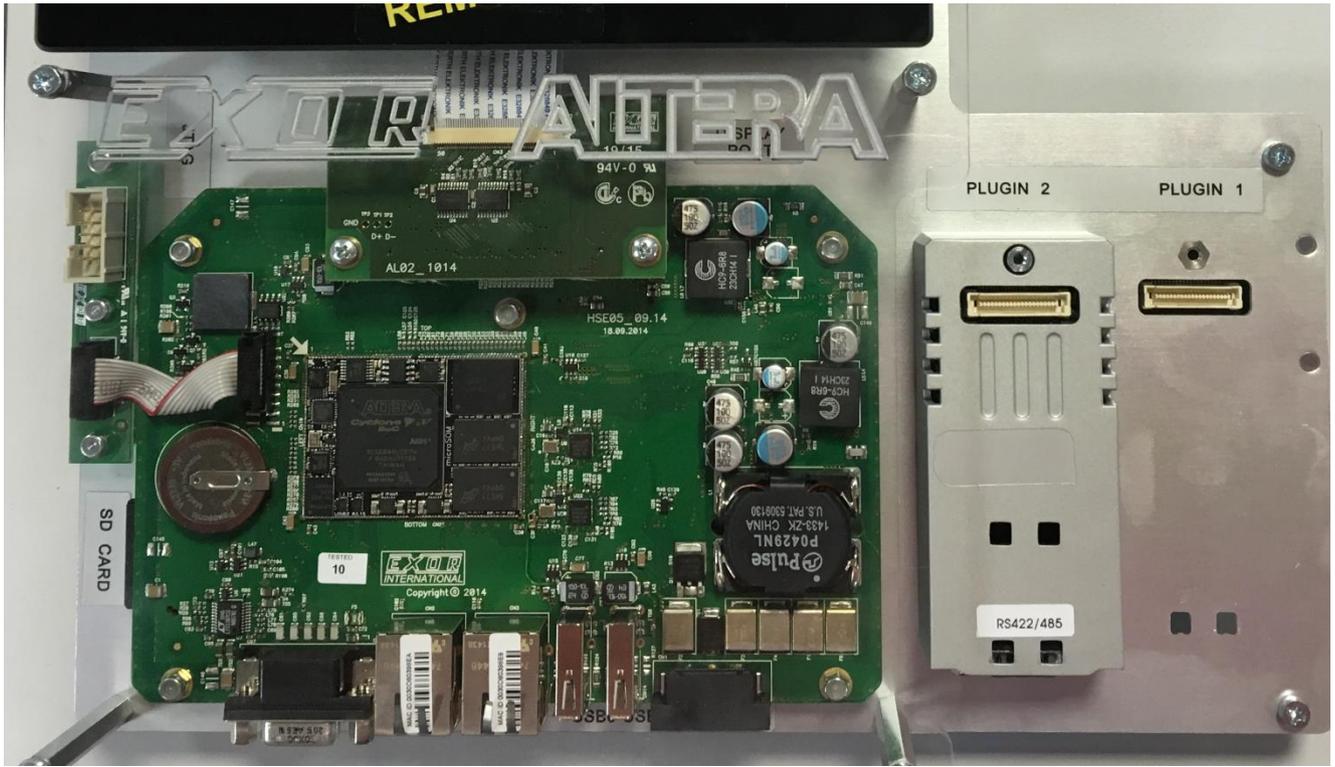
7.2 Use PLCM04 module (RS-422/485)

7.2.1 Plugin connection

The Plcm04 can be plugged in every kit plugin connector.

If you connect the Plcm04 on the connector "Plugin 1" the system provide the ttyS1 interface.

If you connect the Plcm04 on the connector "Plugin 2" the system provide the ttyS2 interface.



7.2.2 System configuration and Plugin use

Once connected the plugin you can power-up the development kit and wait the booting process. The system is just configured to use this module, and you can read/write on the serial port.

7.2.3 Exaple C code

Here a simple example written in C:

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/ioctl.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <linux/serial.h>
#include <asm-generic/termbits.h>
#include <string.h>
#include <signal.h>

/* Driver-specific ioctls: */
#define TIOCGRS485 0x542E
#define TIOCRRS485 0x542F

#define MSG_LENGTH 255
#define HELLO_WORLD "Hello from Exor us02 kit\n"

#define SERIAL_PORT_PLUGIN_1 "/dev/ttyS1"
#define SERIAL_PORT_PLUGIN_2 "/dev/ttyS2"
```

```

/*
 * SELECT USED PORT
 */
#define SERIAL_PORT SERIAL_PORT_PLUGIN_2

int main(int argc, char const *argv[])
{
    int i, fd, ret=0;
    struct serial_rs485 rs485conf;
    unsigned char b[MSG_LENGTH], c[MSG_LENGTH];

    fprintf(stdout, "Start!\n");
    fprintf(stdout, "Open open port %s...!", SERIAL_PORT);
    fd = open(SERIAL_PORT, O_RDWR);
    if (fd < 0) {
        perror ("Open device failure");
        return -1;
    }
    fprintf(stdout, " done!\n");

    fprintf(stdout, "Enable RS485 mode...");
    if (ioctl(fd, TIOCGRS485, &rs485conf) < 0) {
        perror ("ioctl failure");
        return -2;
    }
    rs485conf.flags = SER_RS485_ENABLED | SER_RS485_RTS_ON_SEND;
    if (ioctl(fd, TIOCSR485, &rs485conf) < 0) {
        perror ("ioctl failure");
        return -3;
    }
    fprintf(stdout, " done!\n");

    //Set custom or std baudrate
    struct termios2 tio;
    ioctl(fd, TCGETS2, &tio);
    tio.c_cflag &= ~CBAUD;
    tio.c_cflag |= BOTHER;
    tio.c_ispeed = 115200;
    tio.c_ospeed = 115200;
    ioctl(fd, TCSETS2, &tio);

    write(fd, HELLO_WORLD, strlen(HELLO_WORLD) );
    while( strncmp(b, "exit", 4) )
    {
        fprintf(stdout, "Waiting data on %s \n", SERIAL_PORT);
        memset( b, 0, sizeof(b) );
        ret = read(fd, b, sizeof(b) );
        printf("Data received: %s\n", b );

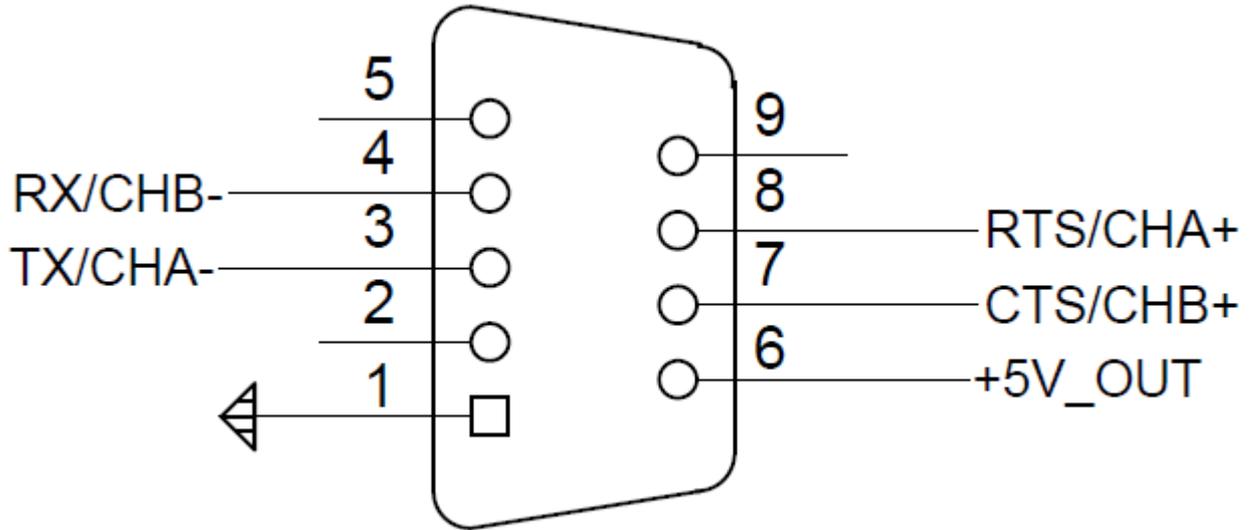
        strcpy(c, "uS02 send: \t");
        strcat(c, b);
        write(fd, c, strlen(c) );
        printf("Data sendend: %s\n", c );
    }

    fprintf(stdout, "Close fd...");
    if (close (fd) < 0) {
        perror ("Close device failure");
        return -4;
    }
    fprintf(stdout, "done!\n");
    fprintf(stdout, "Stop!\n");
    return 0;
}

```

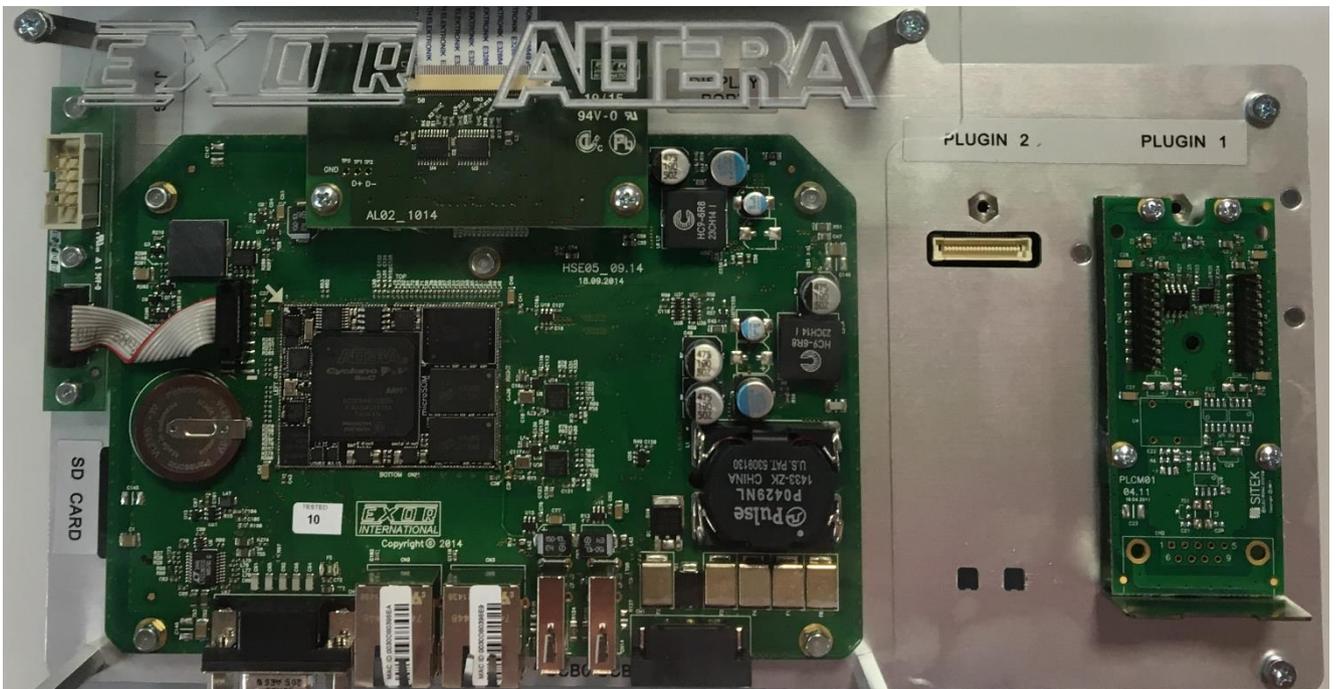
7.2.4 RS485 connector (CN2)

Plcm04 is equipped with a standard male DB9 connector.



7.3 Use PLCM05 module (Expansion module)

The Plcm05 is a simple plugin build to simplify the connections, and use various interfaces.



7.3.1 SPI Plugin connection

The Plcm05 can be plugged in every development kit plugin connector.

If you connect the Plcm05 on the connector "Plugin 1" the system provide the spidev0.1 interface.

If you connect the Plcm05 on the connector "Plugin 2" the system provide the spidev1.1 interface.

7.3.2 SPI System configuration and plugin use

Once connected the plugin you can power-up the development kit and wait the booting process. The system is just configured to use this module, and you can read/write on the SPI port.

7.3.3 SPI Example C code

Here a simple example written in C:

```
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <string.h>
#include <errno.h>

#include <fcntl.h>
#include <linux/spi/spidev.h>
#include <stdio.h>
#include <sys/ioctl.h>

#define MSG_LENGTH 2
#define SPI_PORT_PLUGIN_1 "/dev/spidev0.1"
#define SPI_PORT_PLUGIN_2 "/dev/spidev1.1"

/*
 * SELECT USED PORT
 */
#define SPI_PORT SPI_PORT_PLUGIN_1

static void writeSPI(unsigned char buf[2])
{
    int fd;
    unsigned char swap_buf[2];
    char name[20];
    struct spi_ioc_transfer xfer[2];

    sprintf( name, SPI_PORT );
    //fprintf(stdout,"writeSPI on %s \n ", name );
    fd = open(name, O_RDWR);
    if (fd < 0) {
        perror("Open");
        return;
    }
    memset(xfer, 0, sizeof xfer);
    memset(swap_buf, 0, sizeof swap_buf);

    swap_buf[0] = buf[1];
    swap_buf[1] = buf[0];
    xfer[0].tx_buf = (unsigned long)swap_buf;
    xfer[0].len = 2;
    ioctl(fd, SPI_IOC_MESSAGE(2), xfer);
    close(fd);
}
```

```
static void readSPI(unsigned char buf[2])
{
    int fd;
    unsigned char swap_buf[2];
    char name[20];
    struct spi_ioc_transfer xfer[2];

    sprintf( name, SPI_PORT );
    //fprintf(stdout,"readSPI on %s \n ", name );
    fd = open(name, O_RDWR);
    if (fd < 0) {
        perror("Open");
        return;
    }
    memset(xfer, 0, sizeof xfer);
    memset(swap_buf, 0, sizeof swap_buf);

    swap_buf[0] = buf[1];
    swap_buf[1] = buf[0];
    xfer[0].tx_buf = (unsigned long)swap_buf;
    xfer[0].len = 1;
    xfer[1].rx_buf = (unsigned long) buf;
    xfer[1].len = 1;
    ioctl(fd, SPI_IOC_MESSAGE(2), xfer);
    close(fd);
}

int main(int argc, char const *argv[])
{
    int i;
    unsigned char b[MSG_LENGTH];

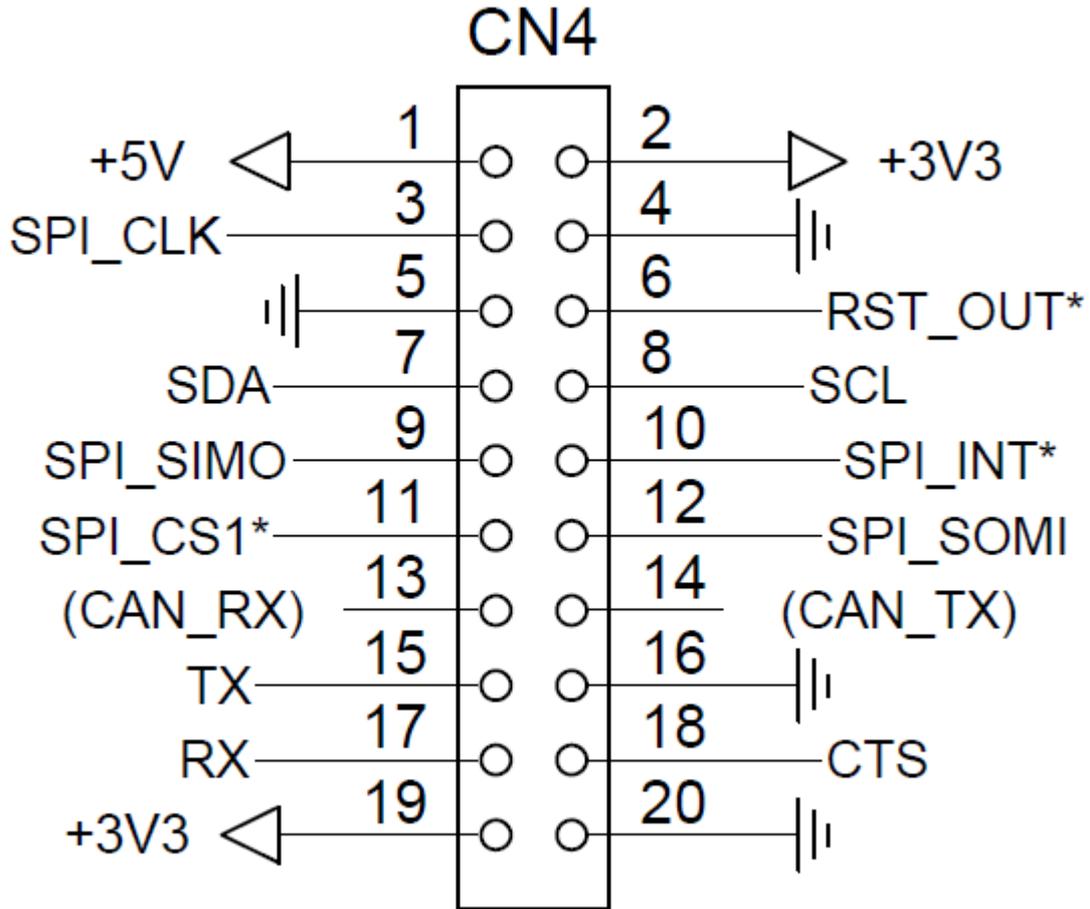
    for(i=0; i<0xFFFF; i++)
    {
        memset( b, 0, sizeof(b) );
        b[1] = ((unsigned char) ((i & 0xFF00) >> 8));
        b[0] = ((unsigned char) ((i & 0x00FF) >> 0));

        writeSPI( b );

        usleep(100000);
    }

    exit(0);
}
```

7.3.4 CN4 Connector



8 Upgrade FPGA firmware (us02-kit only)

Power up the uS02 kit and block the boot process during u-boot countdown by pressing CTRL-C.

```

U-Boot 2015.02.17 (Aug 21 2016 - 12:28:46)

CPU   : Altera SOCFPGA Platform
BOARD : Altera SOCFPGA Cyclone V Board
I2C:   ready
DRAM:  1 GiB
MMC:   initializing dwmmc...
        initializing ultisdc...
ALTERA DWMMC: 0, ultisdc: 1
In:    serial
Out:   serial
Err:   serial
Loading configuration from I2C SEEPROM
bootargs environment variable set to "setenv bootargs console=ttyS0,115200 ro
Hit any key to stop autoboot:  0
SOCFPGA_CYCLONE5 # █
    
```

Type the following commands:

```
# mw.l ff210010 7fe
# run bootcmd
```

Now wait the booting process and in Linux terminal type:

```
# dd if=path_to_new_fpga_image.bin of=/dev/mtdblock0 bs=1M
```

After few minutes the command ends, for use new FPGA firmware power-off and power-on the board.

9 JMobile Portable runtime

JMobile is a software suite designed to offer a complete HMI solution with client-server architecture. It is made of several software components, integrated into a unique application. JMobile applies the latest available technology developed for HMI in industrial automation to every situation where a user interface is required. The suite includes commissioning tools, to allow easy maintenance and configuration of multiple remote units, and both desktop and runtime engineering software for application development.

The portable version of JMobile is a standard Linux JMobile runtime provided as a chroot-based container designed to run under Linux 32bit ARM Cortex A8 and above platforms.

The portable JMobile runtime is provided for rapid prototyping and evaluation purposes and contains a subset (Codesys V2/, Modbus and the internal variables protocol) of the available protocols. In particular serial protocols are not supported, the serial port on the evaluation kits is only meant for debugging purpose.

A closer integration with the final target system and access to the complete set of protocols can be achieved on demand during the product engineering phase.

9.1 JMobile portable runtime installation

By default JMobile is preinstalled on both the standard SD image and the rootfs generated by our standard yocto recipes. However here is how to get started with the JMobile runtime on the evaluation kit from scratch:

1. Get the JMobile portable package for your target:

```

US01Kit:          http://download.exoreembedded.net:8080/Public/usom01/JMobile/
US02Kit (AlteraKit): http://download.exoreembedded.net:8080/Public/usom02/JMobile/
US03Kit:          http://download.exoreembedded.net:8080/Public/usom03/JMobile/
NS01Kit:          http://download.exoreembedded.net:8080/Public/nsom01/JMobile/
  
```

2. Copy it into the kit.

```
$ scp jmobile[...].tar.gz root@[hostname]:~
```

3. Connect to the kit:

```
$ ssh root@[hostname]
```

4. Now, from the remote shell, untar the package in a folder with write permissions (e.g. /home/root)

```

$ cd /home/root/
$ tar xzpf jmobile[...].tar.gz root/
$ rm -rf jmobile[...].tar.gz
  
```

5. Make sure X11 is not running:

```
# /etc/init.d/xserver-nodm stop
```

6. Start JMobile:

```
# /home/root/portable/run.sh
```

If you wish JMobile runtime to start automatically at boot, do the following. Remove the script xserver-nodm:

```
# update-rc.d -f xserver-nodm remove
```

and add a new script to the init sequence:

```
# echo "/home/root/portable/run.sh &" > /etc/init.d/jmobile  
# chmod a+x /etc/init.d/jmobile  
# update-rc.d jmobile defaults 99
```

9.2 JMobile Studio quick start guide

To download a free trial of JMobile Suite go to our web page dedicated to development kits on [exoreembedded.net](http://www.exoreembedded.net):

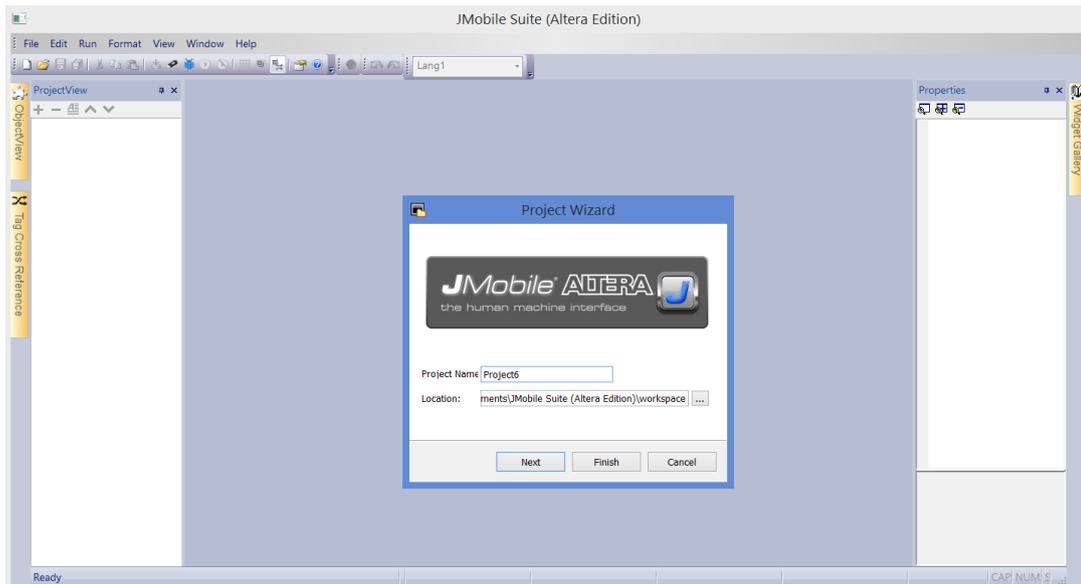
```
http://www.exoreembedded.net/webpage?ReadForm&wPageName=products&c=Development%20Kits&t=Products
```

Select the device you are working with then, from the “Download” section, download the version of JMobile Suite that matches the runtime currently installed on the machine. If unsure, the runtime version can be found as follows:

1. Start the JMobile runtime by using its icon on the sato desktop.
2. Once fully started, touch the screen and hold, a context menu will appear.
3. Move the finger over “About...” then release

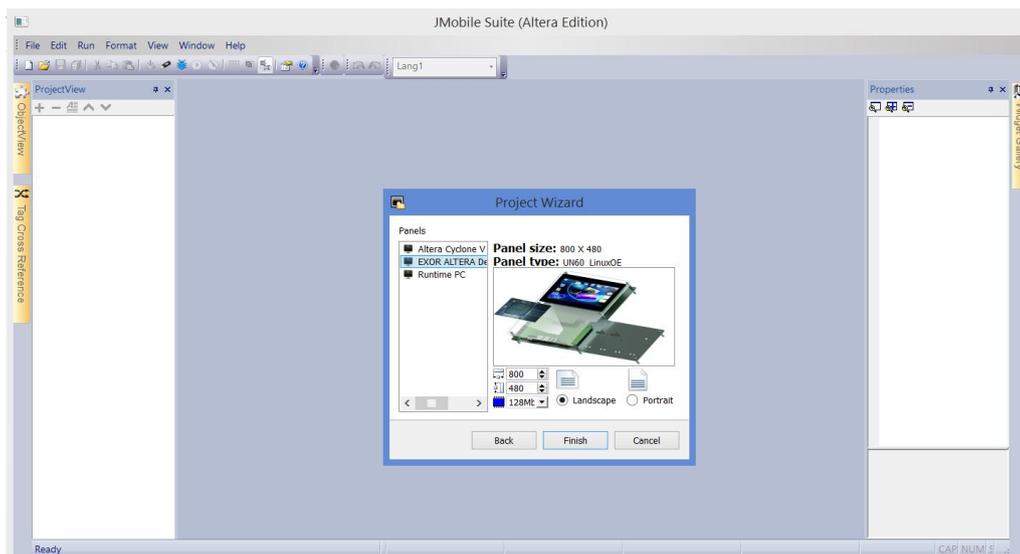
9.2.1 Creating a simple project

After installation, start JMobile Studio and create a new project from “File” -> “New..”:



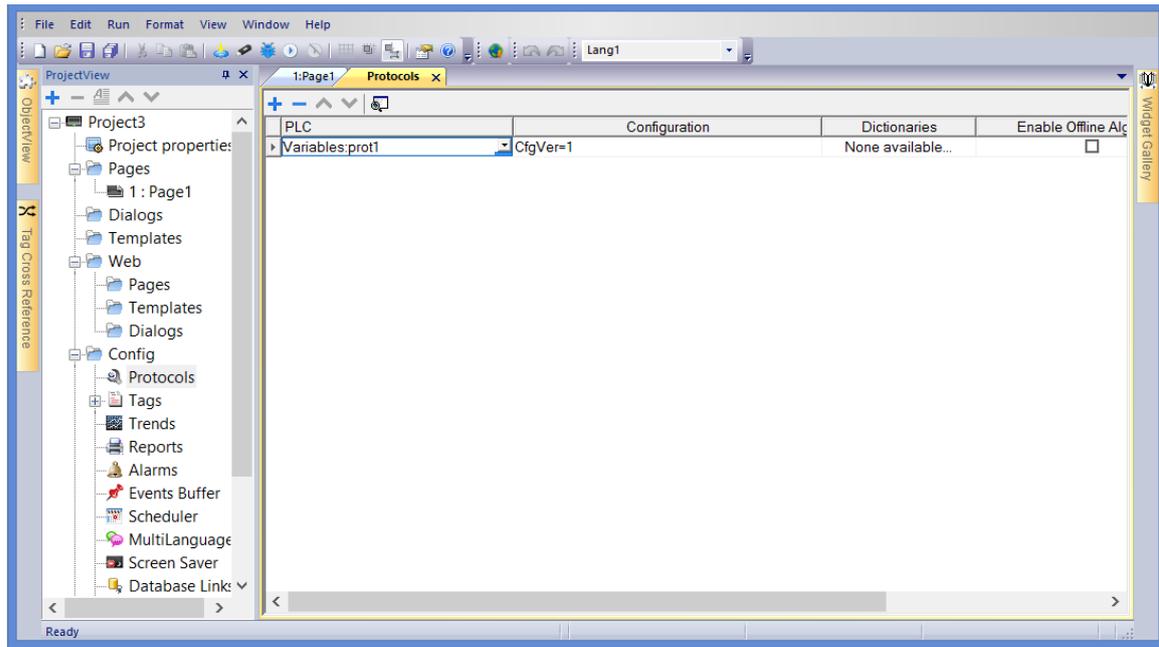
Chose a project name, select a location folder and click on “Next”. Select now the correct target corresponding to the board:

US01Kit: eSMART07M
US02Kit (AlteraKit) : EXOR ALTERA Development Kit
US03Kit : eTOP707

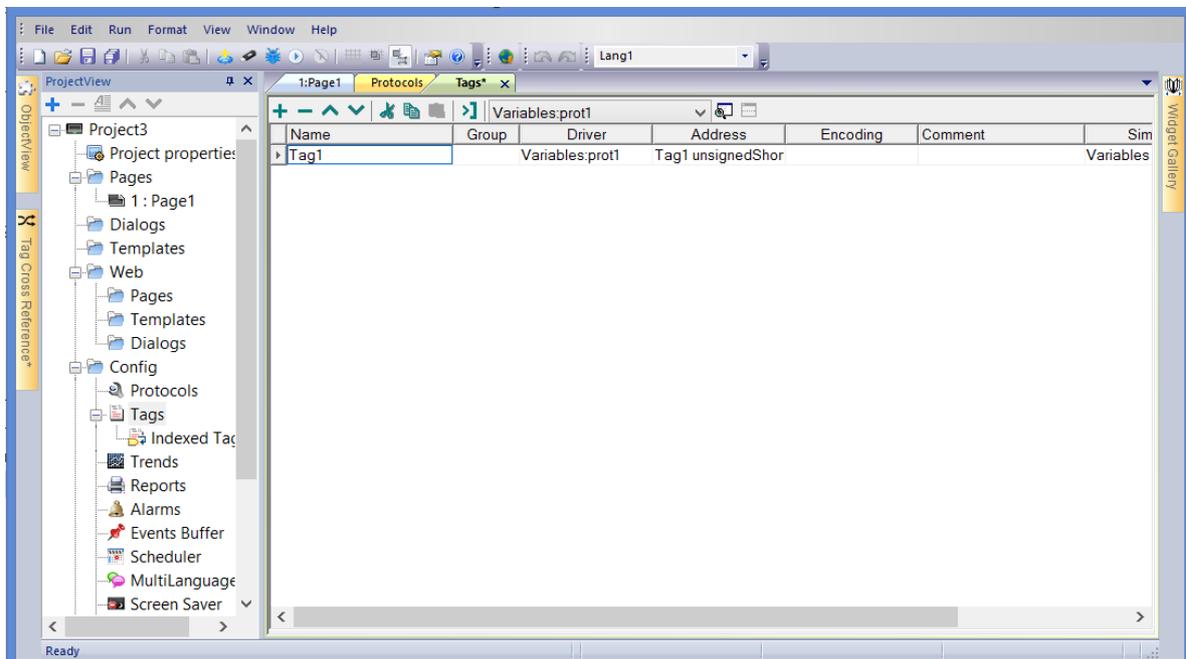


The goal is to create a project simply consisting of an increasing numerical counter. Although at the end it won't do very much, this example project will introduce you to some of the basic mechanics JMobile uses to combine protocols, data and visualization.

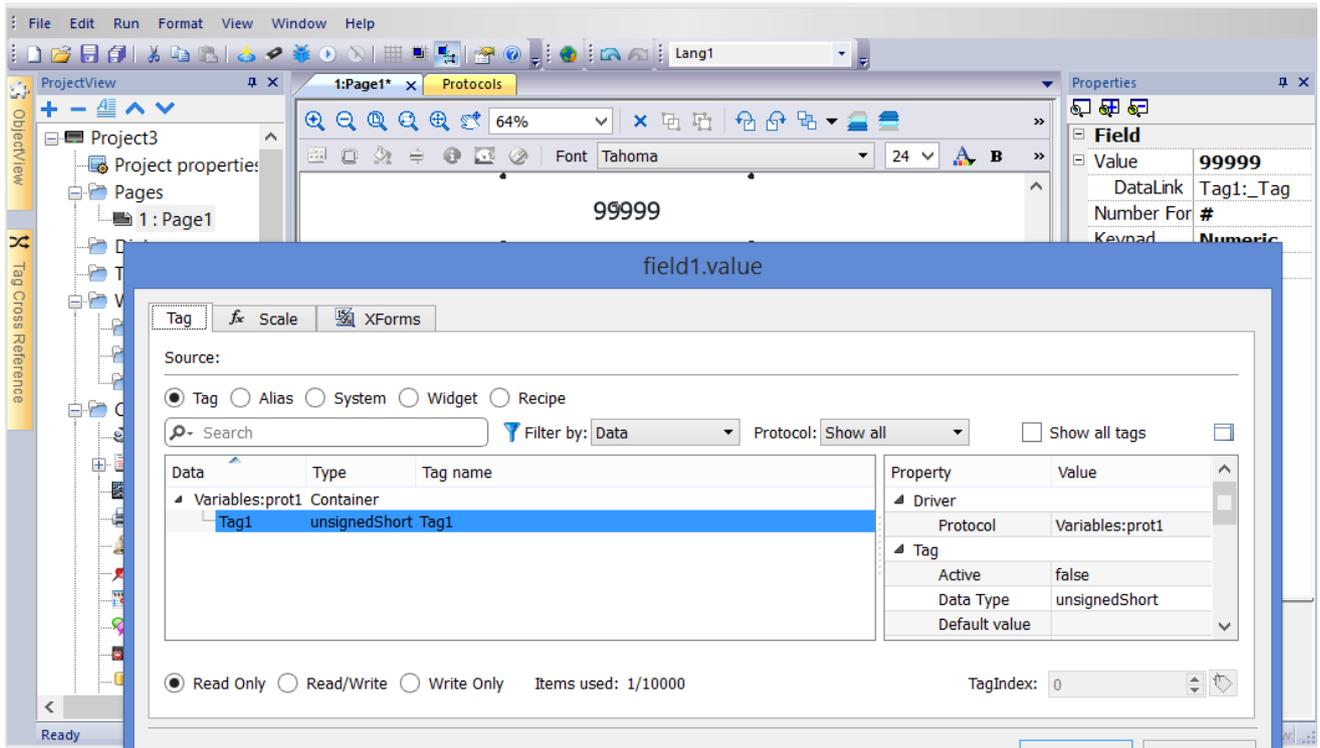
To begin with, from the “ProjectView” on the left panel, click on “Protocols”. Here click “+” to add a new protocol and select “Variables” as shown in the figure below:



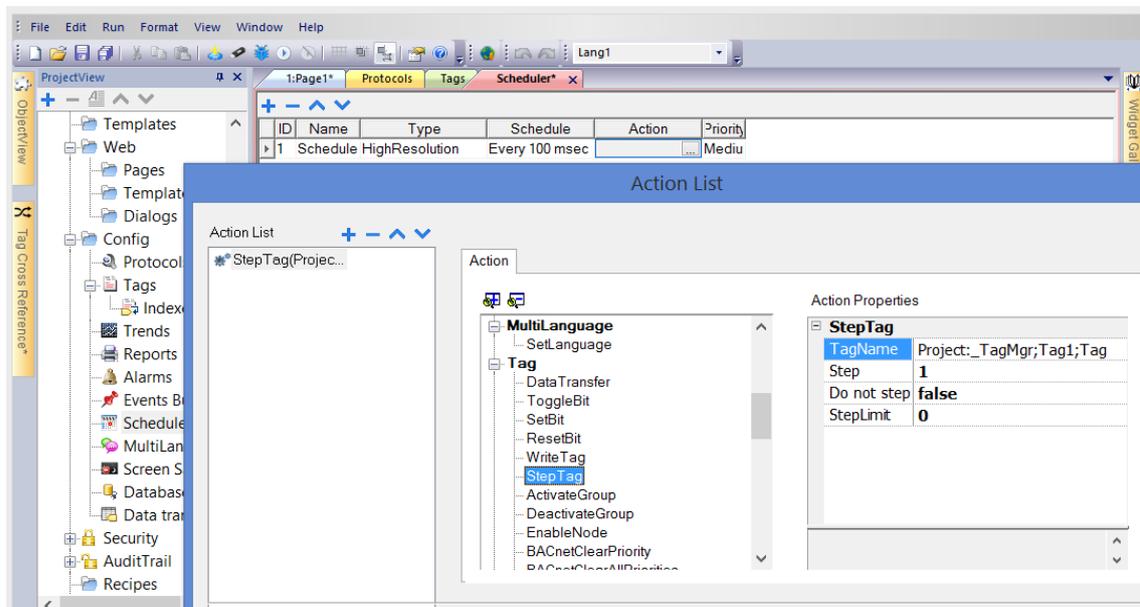
Use the left panel to move to “Tags”. Press “+” and add an unsignedShort tag named “Tag1” representing our counter.



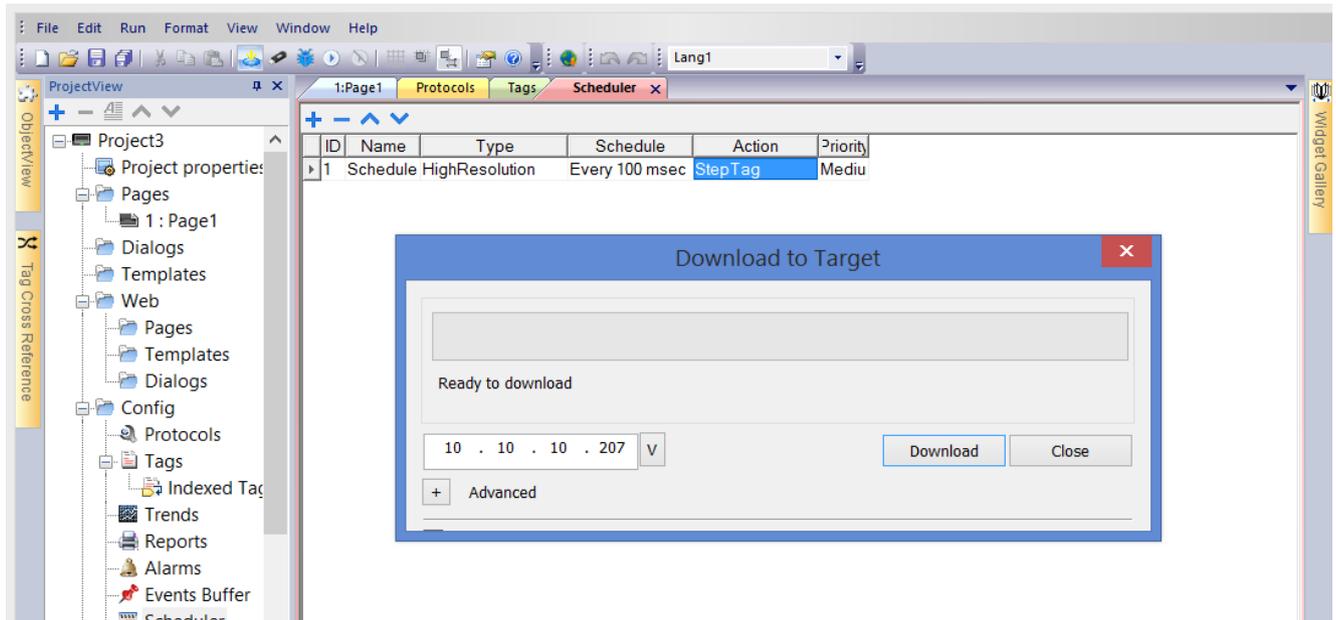
Add a numeric field widget to the project’s page by dragging it from the Widget Gallery. Double click on the numeric filed and select “Tag1” to bind the widget to the tag’s value:



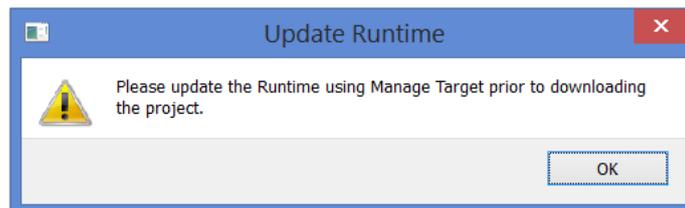
Now move to “Scheduler”. Add a HighResolution scheduler with a “StepTag” action to increment the counter. Note in the figure below that our tag has been selected for “TagName” under “Action Properties”:



Click on the “Download to Target” icon in the toolbar or simply press Ctrl+D. Make sure the development kit is powered on, that you can reach it over the network and that JMobile’s runtime is running. Select the target IP and click “Download” to deploy the project:



If JMobile Studio complains with the following message when downloading:



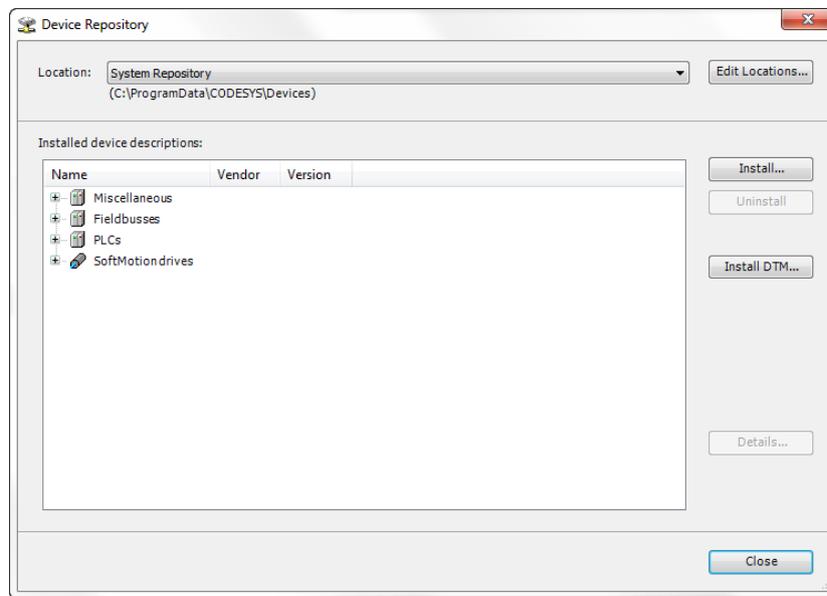
go to “Run” -> “Manage Target...”, select the target IP and press “Update Runtime”. The development kit will reboot after update. If it has not been configured to automatically start at boot execute JMobile’s runtime from its icon on the desktop, then try again downloading the project.

10 CODESYS V3

The CODESYS V3 programming software can be downloaded for free from the CODESYS web site at www.codesys.com/download.html
You need to register before you can download the software.

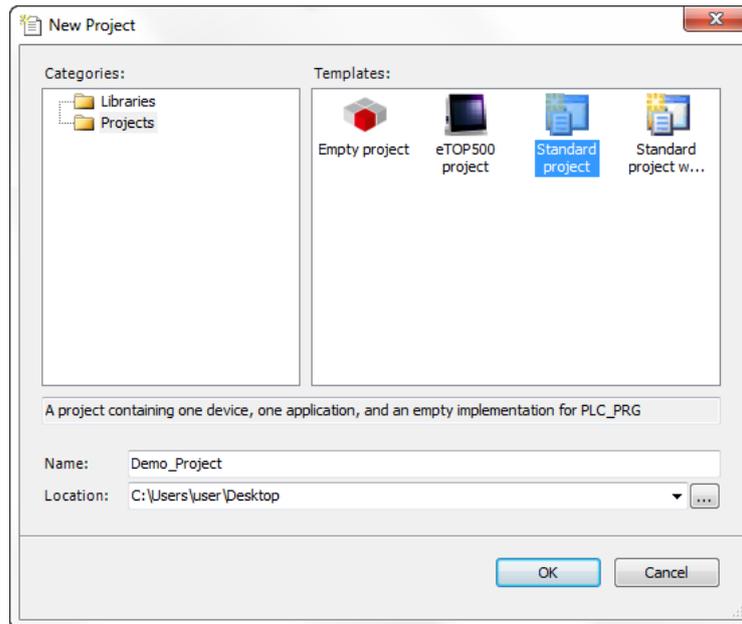
10.1 Installing CODESYS Devices

A device descriptor is required to allow the standard CODESYS V3 work with the evaluation kit. This file can be downloaded from exoreembedded.net (Products > Development Kits), select the evaluation kit you are working with and go to the “Downloads” section.
The CODESYS programming software includes a tool called “Device Repository” for the installation of new device descriptors. The Device Repository tool can be launched from CODESYS Tools menu, selecting the proper menu entry. To install the descriptor, once opened the Device Repository click on the “Install...” button and browse for the xml file. The choice can be confirmed with “Open”. The Device Repository dialog is showed in the below figure:

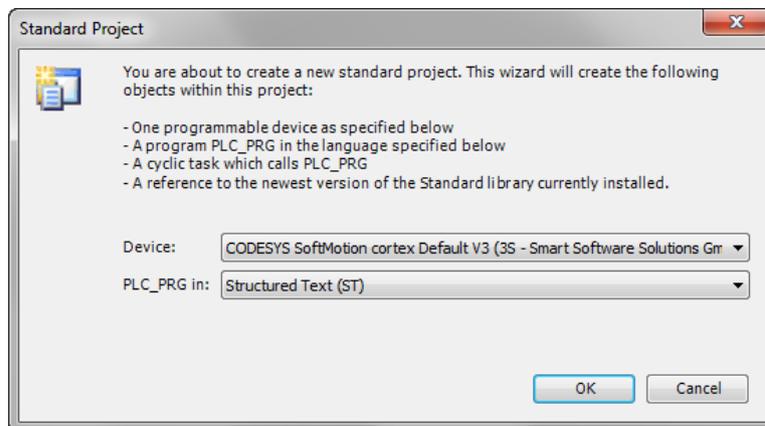


10.2 Creation of a new PLC project

To create a new CODESYS V3 project select “File” > “New Project” or click on the corresponding icon from the upper tool bar. The “New Project” dialog will be displayed, here, among the available templates, select the “Standard project” template. Choose then a project name and a location, confirm with “OK”.



Finally, to complete the project creation, select the “CODESYS SoftMotion cortex” device and the preferred programming language of choice.



10.3 Communication setup in the CODESYS software

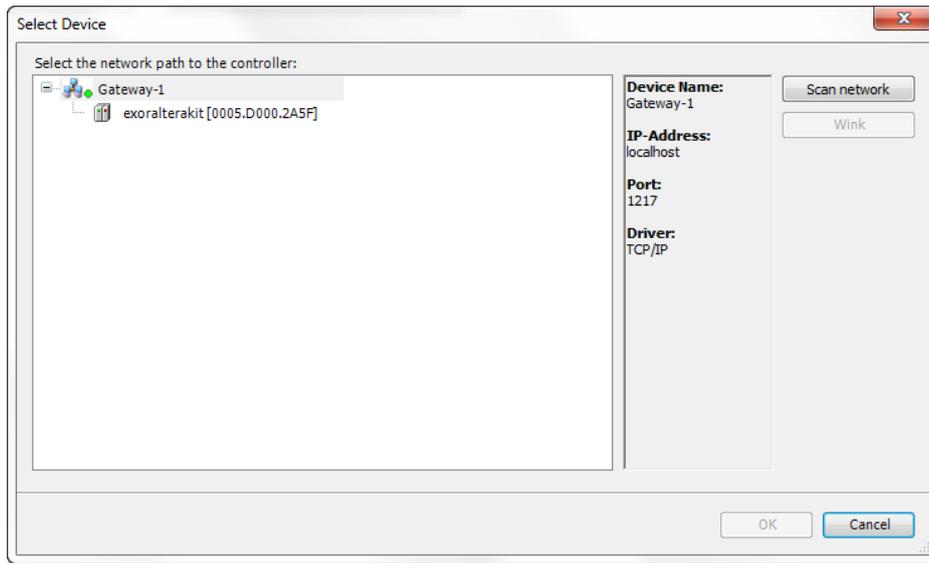
The selection of the PLC where to download the project must be done from Device communication settings tab before proceeding with the download operation.

Double click on the “Device (CODESYS SoftMotion cortex)” voice available in the project tree to display the Device properties in the work area, select the “Communication Settings” tab then click on the “Scan Network...” button.

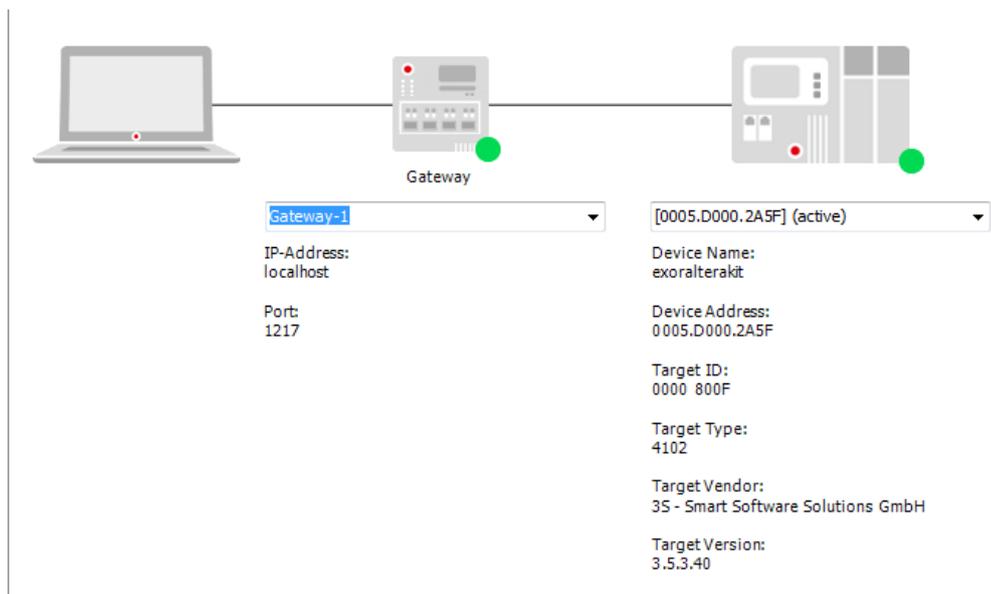
The Select Device dialog will be displayed, this dialog lists all the compatible devices available in the network, select here your device then press on “OK”.

In case more panels using CODESYS V3 PLC runtime are present into the network each panel is recognized by a different string between square brackets reported after Device name. In figure, the

string is "0005.D000.2A5F", the last part of the string "2A5F" corresponds to the last 2 bytes of the panel IP Address in Hex format so, in this case, the corresponding operator panel is the one with IP address xxx.xxx.42.95 as 2AHex corresponds to 42 Dec and 5F Hex corresponds to 95 Dec.



The selected device is then listed in the Communication Settings as shown below. the device properties are listed on screen. A green dot over the device graphical representation informs that the device is correctly recognized and available into the network.



Communication with the available devices is established through a Gateway, a default Gateway is available, and it is generally not needed to change the standard Gateway settings. For more information about the Gateway set-up please refer to CODESYS V3 documentation.

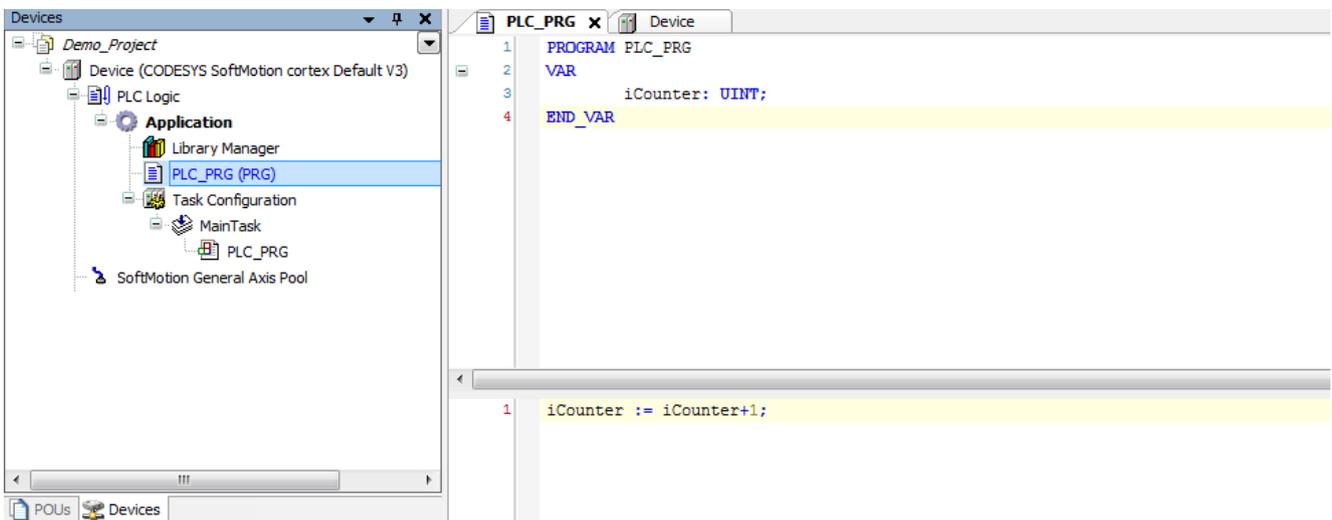
11 Accessing PLC from JMobile

JMobile and CODESYS projects discussed in this section are included in the “JMobile_CDS_demo” folder from the demo projects package “Demo_Projects.zip”, downloadable from exoreembedded.net (Products > Development Kits).

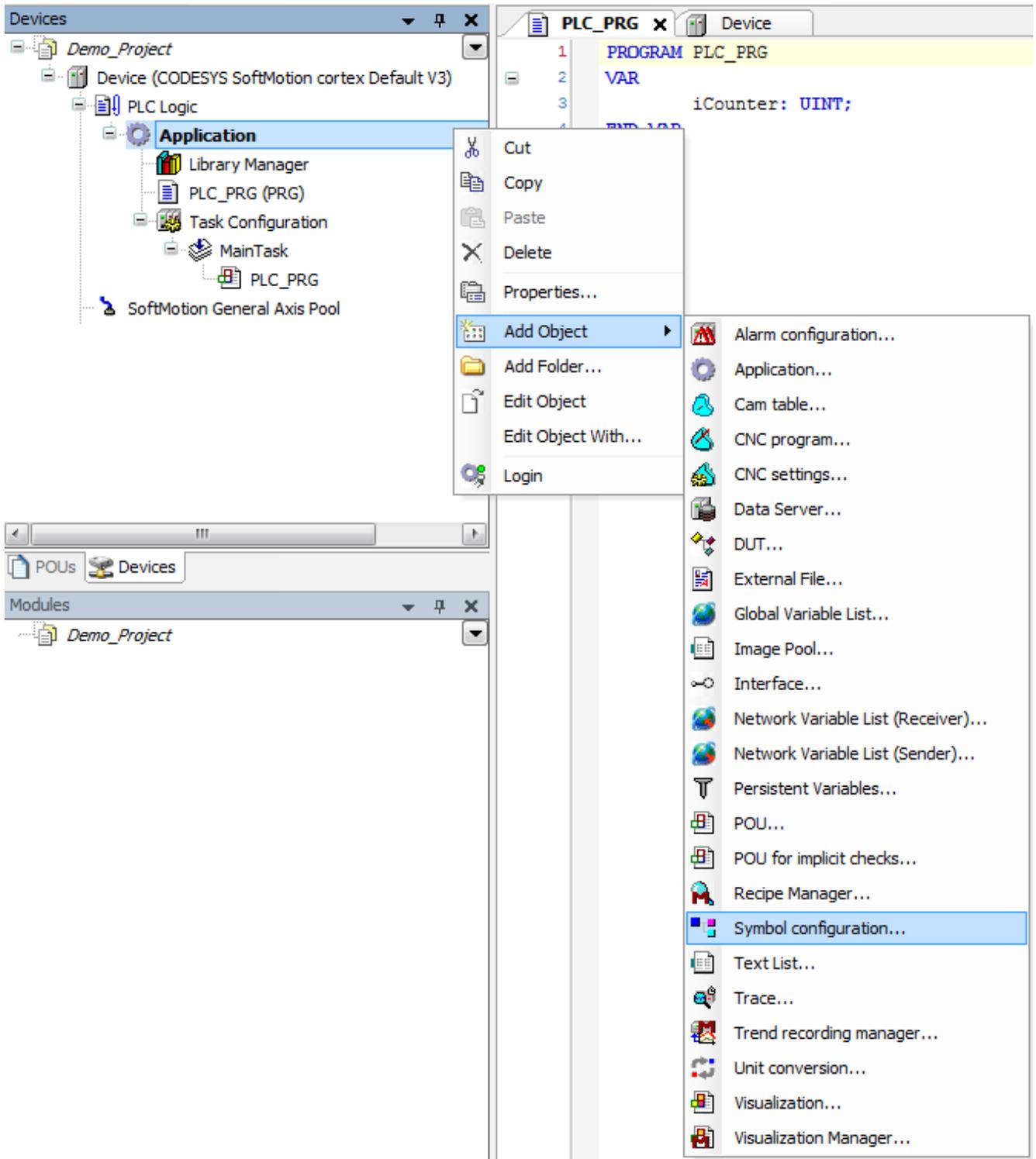
The JMobile Portable Altera edition contains also a demo version of CODESYS v3 runtime which is started along with JMobile runtime. Here is presented an example in which JMobile will be able to access variable values from the PLC.

11.1 Codesys project creation

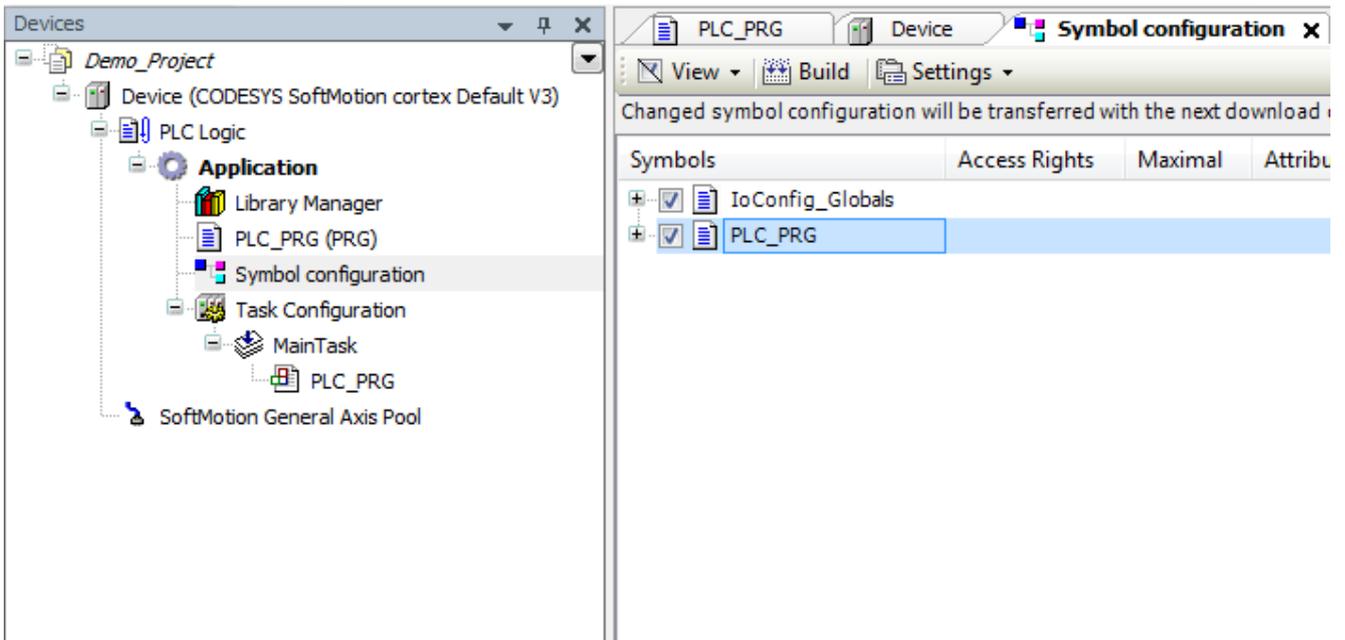
First of all we need to create a simple Codesys program. Assuming a configured project has already been created on CODESYS v3 and that the Altera kit is properly connected we can write these few lines of code inside the PLC_PRG file:



Now right click on “Application” and select “Add Object > System configuration”:



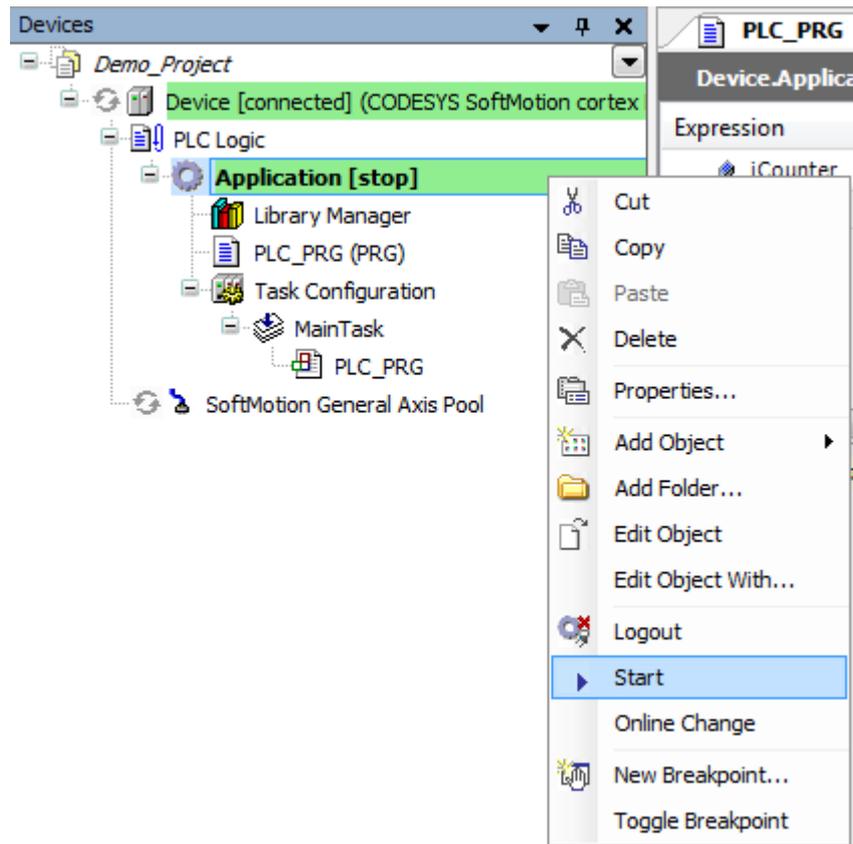
Double click on the newly created “Symbol configuration” object and from the opening tab click “Build” and then make sure to check the symbols in the list below:



In the main toolbar click on “Build > Generate code” to create, among other files, an xml file that we will later use to import PLC variables on JMobile.

Now from the main toolbar choose Online > Login to deploy the program on the device. If you get a warning about an existing program on the PLC click OK to overwrite it with the new one.

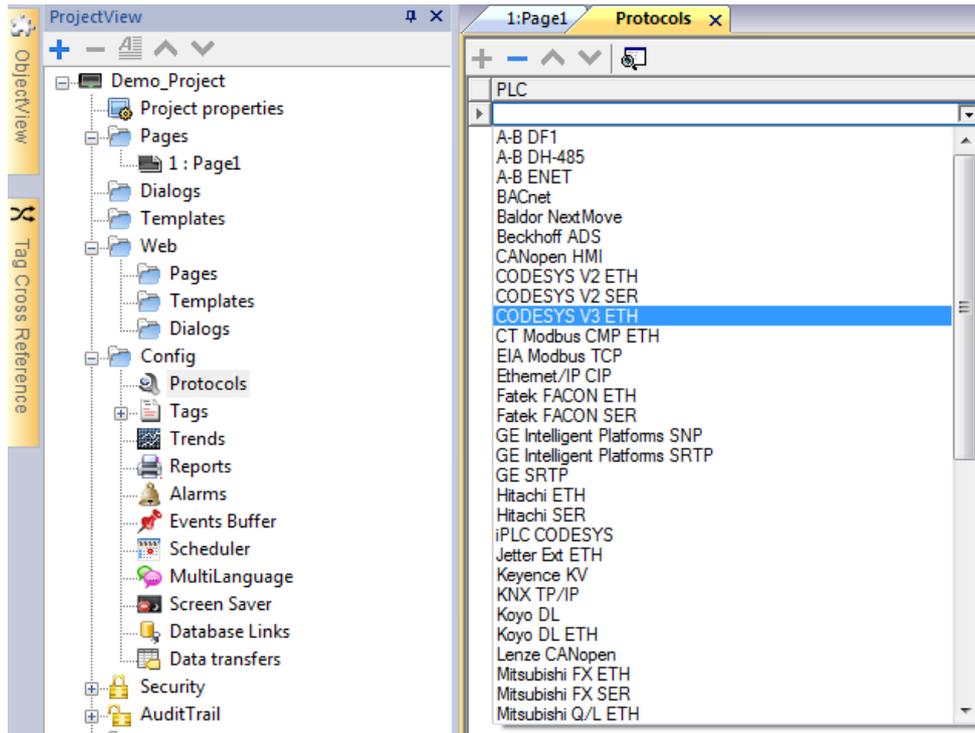
Lastly, to run the program on the PLC, right click on Application and choose “Start”:



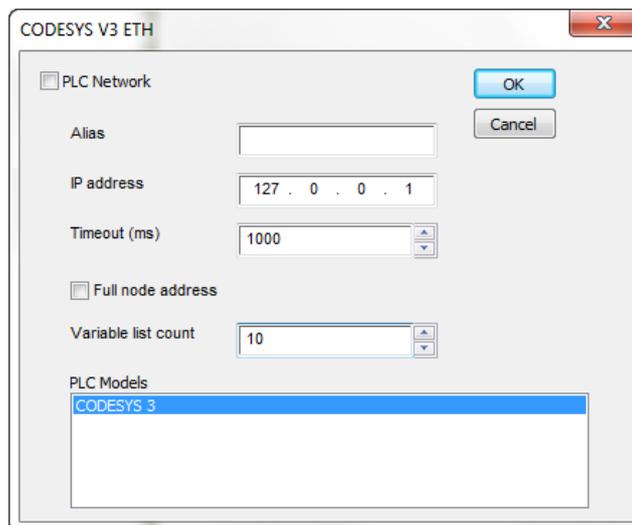
To keep PLC project on CPU also after a reboot, once the device is online, click on "Online > Create boot project".

11.2 CDS3 protocol configuration on JMobile

On JMobile, from a Altera Cyclone V project, select "Protocols" from the Project View on the left. Click on the "+" button and select "CODESYS V3 ETH".



Configure the protocol as shown in the figure below, then click “OK”.

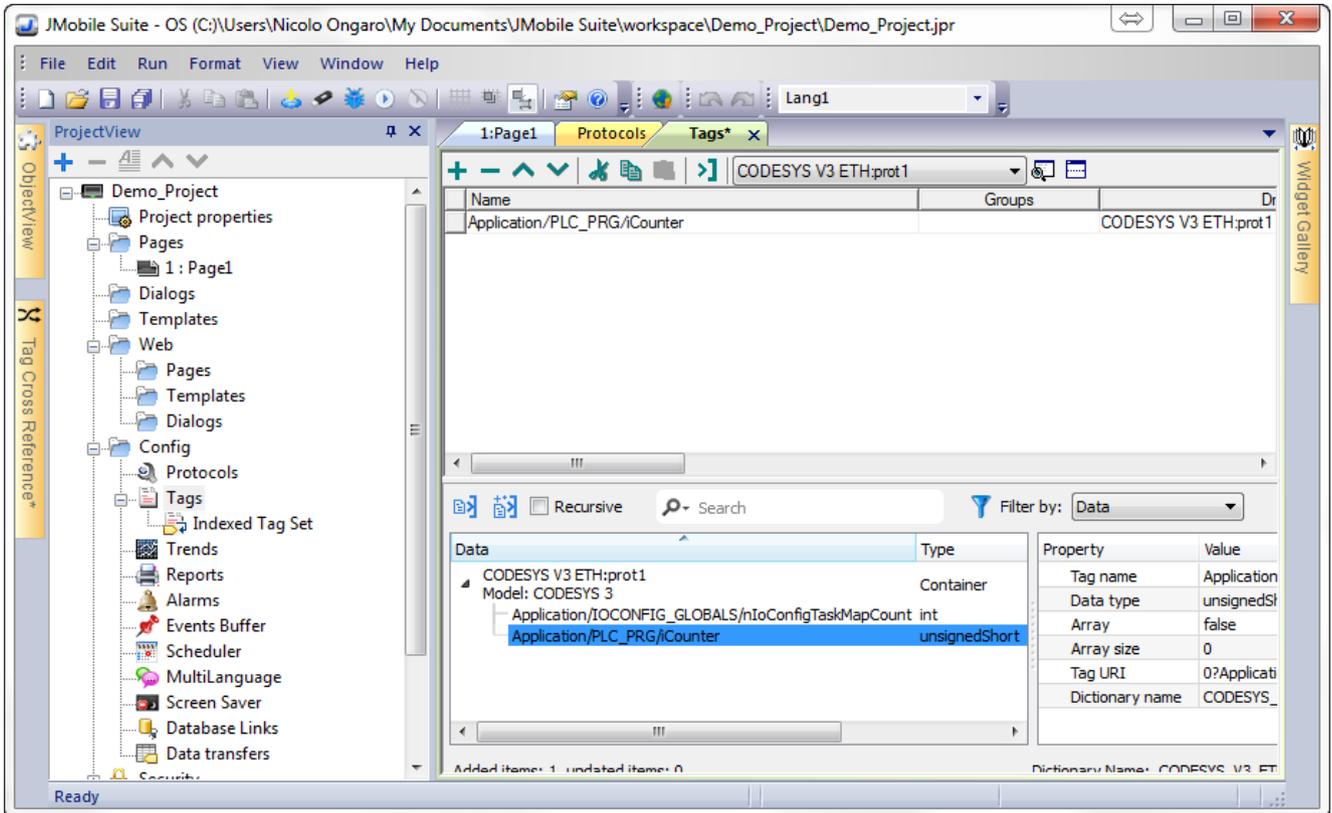


Note that with the above configuration JMobile Runtime will look for Codesys connection on the localhost. This will work only when the project is running on the Altera kit. To avoid this behavior you can select the Altera kit network interface IP instead of the one in the figure.

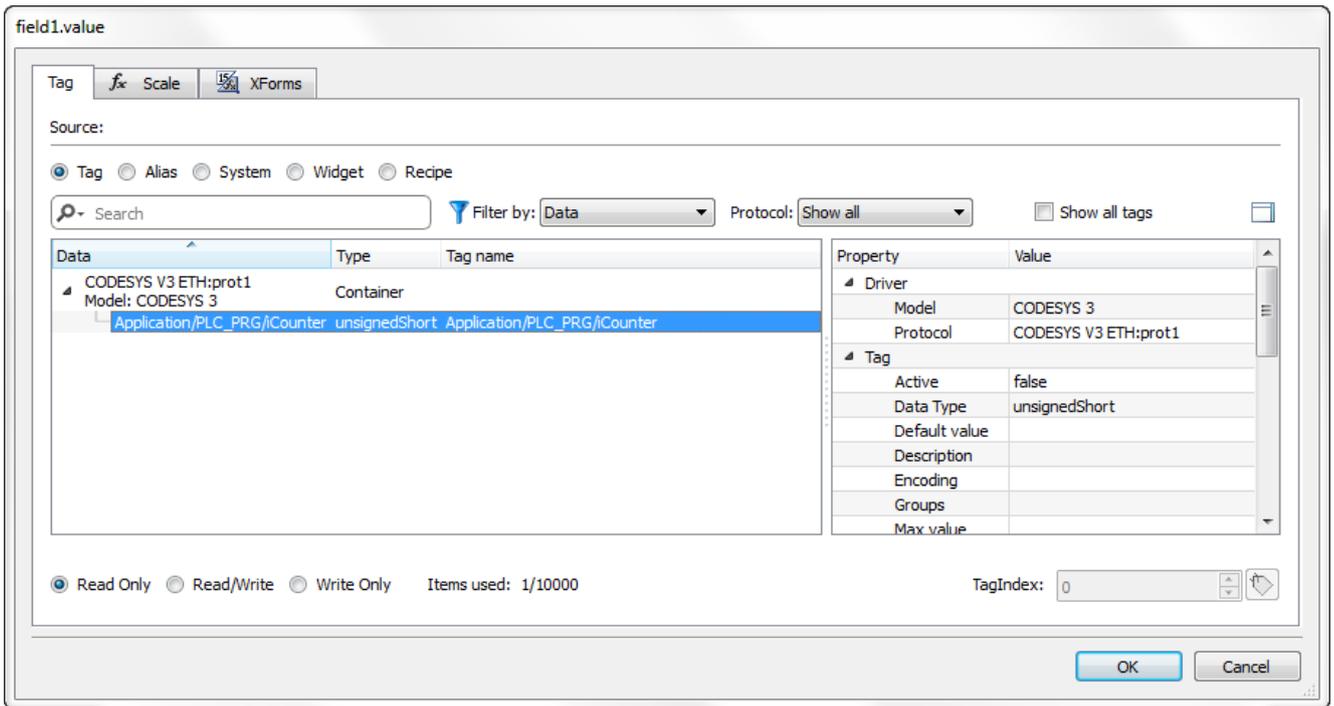
Now, to import tags from our Codesys project, select “Tags” from the “Project View”. Here you have to:

1. Select “CODESYS V3 ETH:prot1” as protocol.

2. Click on the “Import Tags” button . Choose “CodeSys3 v1.0” as tag importer, click OK and then browse and select the xml file you will find in the Codesys project folder we have created before. The name of this file should be something like “<project_name>.Device.Application.xml”.
3. Now, from the lower section of the “Tags” tab select the variable “Application/PLC_PRG/iCounter” and click on the Import Tag(s) button .



Lastly, on a project page, add a numeric label widget, right click on it and choose “Attach to..”. Here select the Codesys tag and click “OK”.



Once you have downloaded the project to the Altera kit, if the PLC program is running, you should see the Codesys "iCounter" variable being incremented.