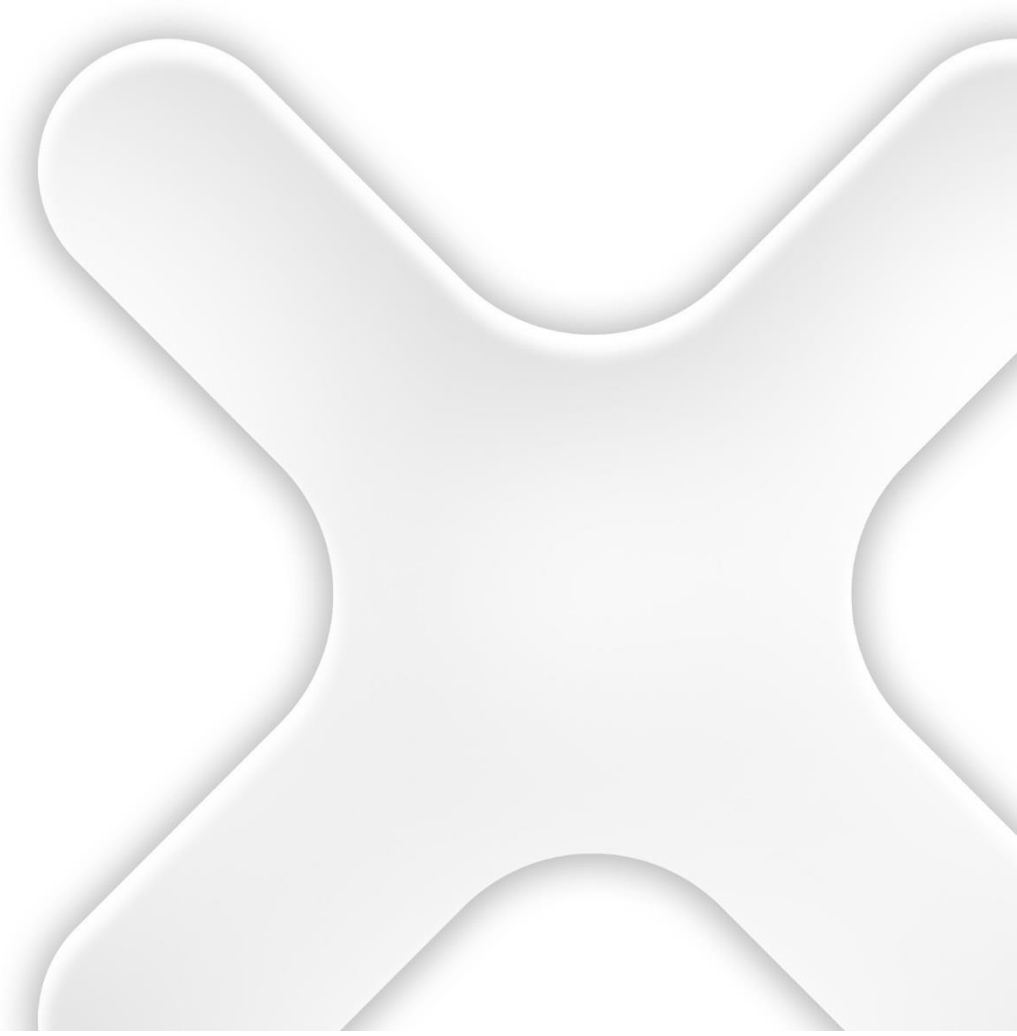
Kairos development kit

Software user manual



Kairos development kit

History			
Rev	Date	Description	By
1.0	18/Jul/2019	Initial version	AG
1.1	27/Aug/2019	Updated screenshots Added details about command line utilities Added sections about OpenHMI	AG

Reference		
Cross Reference	Filename	Description
[1]		

TABLE OF CONTENTS

1 Overview

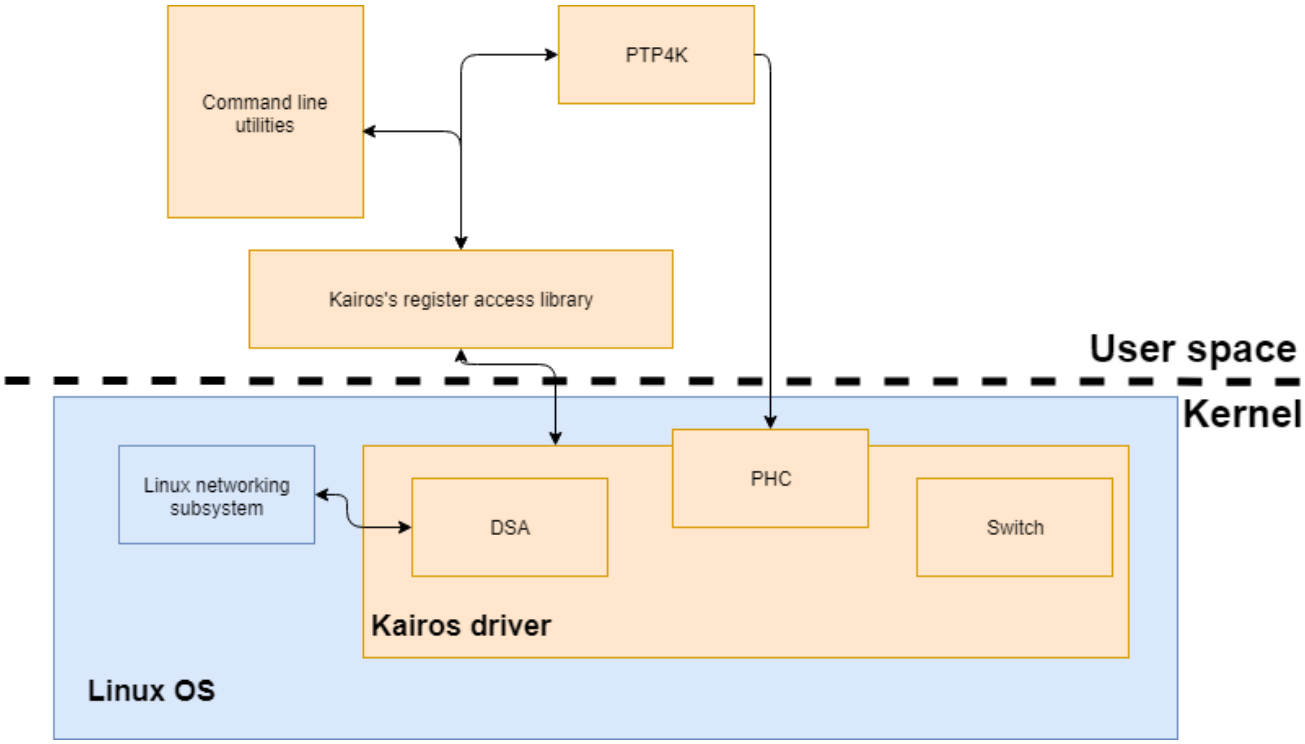
This manual provides a getting started guide to the Kairos development kit. This document will guide you through the following steps

1. Understand the software architecture of the Kairos-based solution
2. Go through the implementation details of the software provided
3. Create the environment to build test applications
4. Build the all the system software (BSP) from scratch
5. Deploy the system software to an SD card

2 Software overview

Kairos kit has been developed to give you an easy-to-use platform for experimenting with the Kairos chip. Kairos chip implements a TSN (Time Sensitive Network) stack to simplify the implementation of devices that can leverage TSN technologies to provide reliable and deterministic packet delivery

The Kairos development kit software architecture is shown in picture below



In light orange are the components specific to the Kairos solution, in light blue are the components that are part of a standard Linux distribution

2.1 Kairos driver

Kairos driver implements all the functionalities required to integrate Kairos features in the standard Linux kernel. It exposes interfaces that make use of Kairos chip features as easy and seamless as possible from a typical user application. Three major components are built inside the Kairos driver

2.1.1 DSA-compliant switch

The Kairos chip has been integrated in the standard Linux kernel by leveraging the DSA (Distributed Switch Architecture) framework. Thanks to this approach, the two Kairos native Ethernet ports installed on the development kit are visible as standard network card (named lan0 and lan1) and can be accessed and configured by means of standard tools like ifconfig, ethtool, etc.
More details about DSA architecture can be found at this link

https://www.kernel.org/doc/Documentation/networking/dsa/dsa.txt

2.1.2 PHC

Kairos PHC (PTP Hardware Clock) device complies with the standard method for developing PTP user space applications (namely ptp4l).
A new class driver exports a kernel interface for specific clock drivers and a user space interface. The infrastructure supports a complete set of PTP hardware clock functionality.
Basic PHC operations include

- Set time
- Get time
- Shift the clock by a given offset atomically
- Adjust clock frequency

Ancillary clock features include

- Time stamp external events
- Period output signals configurable from user space
- Synchronization of the Linux system time via the PPS subsystem

More details about the PTP Hardware Clock drivers can be found at this link

https://www.kernel.org/doc/Documentation/ptp/ptp.txt

2.1.3 Switch

This submodule implements the interface to the Kairos chip specific features, like the Qbv scheduler, the integrated bridge capabilities on so on

FDB entries can be added to the Kairos switch using standard bridge command. For example, you can add a new FDB entry to lan0 port by typing the following command

```
bridge fdb add 01:1b:19:00:00:00 dev lan0
```

2.2 Kairos registers access library

This is a library provided as redistributable source code that implements all the routines required by a user application to access the Kairos register through a standard SPI interface.
This library exposes these main functions

2.2.1 Read Kairos register

This function reads a 16-bits register from the Kairos module

```
int kairos_read(const char* spidev,
       KAIROS_MODULES module
       uint8_t addr,
       uint16_t* rvalue);
```

where

- `spidev` is the SPI device the Kairos chip is connected to (e.g. /dev/spidev0.0)
- `module` is the desired Kairos module. Valid values are
    - KAIROS_MODULE_GENERAL: general module
    - KAIROS_MODULE_TSN
    - KAIROS_MODULE_PTP
- `addr` is the address of the register to read

- `rvalue`: pointer to the variable where the value being read will be stored

2.2.2 Write Kairos register

This function writes a 16-bits value into a Kairos module's register

```
int kairos_write(const char* spidev,
       KAIROS_MODULES module
       uint8_t addr,
       uint16_t wvalue);
```

where
- `spidev` is the SPI device the Kairos chip is connected to (e.g. /dev/spidev0.0)
- `module` is the desired Kairos module. Valid values are
    - KAIROS_MODULE_GENERAL: general module
    - KAIROS_MODULE_TSN
    - KAIROS_MODULE_PTP
- `addr` is the address of the register to read
- `wvalue`: is the values to write

2.3 Command line utilities

Built upon the Kairos register access library, command line utilities provides a ready-to-use solution for setting up the Kairos chip

2.3.1 kspi

This utility provides read and write access to any register in the Kairos chip (please refer to Register document for details about Kairos chip registers)
To use of this utility is straightforward. To read a register, the syntax of the command line is

```
kspi rd <module> <addr> <num regs>
```

where
- `<module>` is the Kairos chip module number, namely
  1: general module
  2: TSN module
  3: PTP module
- `<addr>` is the register address (hexadecimal format)
- `<num>` *regs* is the number off register to read

The syntax to write a register is

```
kspi wr <module> <addr> <num regs> <values>
```

where
- `<module>` is the Kairos chip module number, namely
  1: general module
  2: TSN module
  3: PTP module
- `<addr>` is the register address (hexadecimal format)
- `<num>` *regs* is the number off register to read
- `<value>` is the value to write (decimal format)

2.3.2 ktsn

`ktsn` is another tool that provides high-level functionalities to configure the Qbv scheduler of the Kairos chi. Supported functions include

- load a scheduler configuration file
- configure scheduler cycle time and base time (either relative or absolute)
- set PVID (Port VLAN Identifier)
- show TSN statistics
- show PTP statistics

Here are some details about supported command line syntaxes

2.3.2.1 Configure scheduler

The command line syntax to set scheduler cycle time and base time is

```
ktsn config <cycletime> <basetime> <gatefile>
```

where

- `<cycletime>` is the period (in nanoseconds) in which the sequence of gate operations is performed. As the name suggests, the cycle is repeated periodically. Once a cycle begins, the gate sequence is always restarted from the beginning. A cycle should at least be able to accommodate one (or more) packet transmission times to be useful. Note that the current version of the IP core does support CycleTimes with nanosecond granularity, so fractional values are not possible.
- `<basetime>` is required to achieve synchronization among all the devices on the network. Qbv gate operations are taking place inside of cycles. For useful inter-device operation, these cycles should always start at defined points in time, even after a device (temporarily) reboots. It might be required by an application that all devices in a network start their cycle at the same point in time. To achieve such synchronization, a parameter called AdminBaseTime is configured into all devices participating in the TSN network (the same value would be used if the cycles should start at the same time on all devices). When a device begins or resumes TSN operation, it must determine the time of the first cycle it starts. This is based on the precise network time base (e.g. PTP), so the device needs to be synchronized. If the configured AdminBaseTime is still in the future, the device will simply wait until that time comes and start the first cycle at that time. On the other hand, if AdminBaseTime lies in the past, the device needs to calculate the next feasible cycle start using the AdminBaseTime value.
  The principle of this calculation is to quantize time based on the given cycle times, with AdminBaseTime as the starting point. In other words, the intended cycle start will be at an integer multiple of cycle times added to AdminBaseTime. To calculate the actual value, the TSN driver could simply keep adding cycle time values to the admin base time until the accumulated value is larger than the present time.
  The supported formats for these parameters are
  1. Absolute time: an absolute time can be expressed using the ISO8601 format
     . `<YYYY>-<MM>-<DD>T<hh>:<mm>:<ss>.<mmm>`
     For example:
     . `2018-01-03T01:02:03.456`
     Absolute time is assumed to be expressed in UTC time
  2. Relative time: a relative time can be expressed using the following format
     . `[+|-]<seconds>.<milliseconds>`
     For example
     . `+10.123`
     will apply the Qbv schedule starting from 10 seconds and 123 milliseconds from current time

- `<gatefile>` is a file with a sequence of scheduler gates status. There are 8 scheduler gates (one for each available VLAN priority). The gatefile is made up of one or more lines with the following format
  . `<command> <duration> <gates status>`

  where
  - `<command>` is the command to execute. Currently, only the `sgs` command (Set Gates Status) is supported
  - <duration> is the period of time (expressed in nanoseconds) the gates must stay in this state
  - `<gate status>` is the status the gates must be put on. This is value in hexadecimal format ranging from 0x00 to 0xFF. This value is treated as a bitmask. Bit 0 corresponds to the status of gate 0 (i.e. the gate through which untagged packets are sent). If the bit is 0, the gate is disabled (no packets are allowed to be sent through that gate). If bit is 1, the gate is enabled, and packets are allowed to be sent out

    An example of the content of a valid `<gatefile>` is as follow
    . `sgs 500000 0x0F`
    . `sgs 500000 0xF0`

    In this configuration, gates 0 to 3 stays open for 500 microseconds of the cycle, then gates 4 to 7 opens for 500 microseconds

2.3.2.2 Show TSN statistics

The command line syntax to show TSN statistics is

```
ktsn show tsn
```

Example of the output of this command is shown below

```
Build date: 19/1/24 16:18:45
Release: 4.173.0 - 206
Current counter values
Counter         Interlink             Front A         FrontB
Rx filtered               0                    0              0
Rx octets                 -                    0              0
Rx tagged                 -                    0              0
Rx errors                 -                    0              0
Rx overload               0                    0              0
Rx unicast                -                    0              0
Rx multicast              -                    0              0
Rx broadcast              -                    0              0
Rx < 64 bytes             -                    0              0
Rx 64 bytes               -                    0              0
Rx < 128 bytes            -                    0              0
Rx < 256 bytes            -                    0              0
Rx < 512 bytes            -             53347118              0
Rx < 1024 bytes           -                    0              0
Rx <=1518 bytes           -                    0              0
Rx > 1518 bytes           -                    0              0
Tx 0 dropped              0                    0              0
Tx 1 dropped              0                    0              0
Tx 2 dropped              0                    0              0
Tx 3 dropped              0                    0              0
Tx 4 dropped       53347118             16056565       69403683
Tx 5 dropped              0                    0              0
Tx 6 dropped              0                    0              0
Tx 7 dropped              0                    0              0
Rx 0 frames               0                    0              0
Rx 1 frames               0                    0              0
Rx 2 frames               0                    0              0
Rx 3 frames               0                    0              0
Rx 4 frames               0                    0              0
Rx 5 frames               0                    0              0
Rx 6 frames               0                    0              0
```

```
Rx 7 frames              0              67437573              0
Tx octets                -              44958382              0
Tx tagged                -                     0              0
Tx errors                -                     0              0
Tx unicast               -                     0              0
Tx multicast             -                     0              0
Tx broadcast             -                     0              0
Tx < 64 bytes            -                     0              0
Tx 64 bytes              -                     0              0
Tx < 128 bytes           -                     0              0
Tx < 256 bytes           -                     0              0
Tx < 512 bytes           -                     0              0
Tx < 1024 bytes          -                     0              0
Tx <=1518 bytes          -                     0              0
Tx > 1518 bytes          -                     0              0
Tx 0 overrun             -                     0              0
Tx 1 overrun             -                     0              0
Tx 2 overrun             -                     0              0
Tx 3 overrun             -                     0              0
Tx 4 overrun             -                     0              0
Tx 5 overrun             -                     0              0
Tx 6 overrun             -                     0              0
Tx 7 frames              0                     0              0
Tx 0 frames              0                     0              0
Tx 1 frames              0                     0              0
Tx 2 frames              0                     0              0
Tx 3 frames              0               6291552              0
Tx 4 frames              0                     0              0
Tx 5 frames              0                     0              0
Tx 6 frames              0                     0              0
Tx 7 frames              0                     0              0
```

2.3.2.3 Show PTP statistics

The command line syntax to show PTP statistics is

```
ktsn show ptp
```

NOTE: this command shows the content of some internally-used PTP registers and is useful for debug purposes. From a user point-of-view, it's more useful to get PTP statistics from standard Linux tool (e.g. pcm)

Example of the output of this command is shown below

```
Build date: 19/1/24 16:18:45
Release: 0.0.0 - 206
Clock         0.000000000
Free running  1.045946744
Syntonized    1.091274195
Synchronized  21.165095085
Drift accum.  0.007696582
Port A Rx     21.147697227
Port A Tx     0.000000000
Port B Rx     0.000000000
Port B Tx     21.159090705
```

2.3.3 kfdb

`kfdb` is a command line utility to create, delete and query Kairos FDB entries

2.3.3.1 Add FDB entry

The command line syntax to create a new FDB entry is

```
kfdb add <MAC address> <port mask> <priority>
```

where

- `<MAC address>` is the MAC address to add to the Kairos FDB database. The MAC address is made of six hexadecimal values separated by ':' (column). A valid MAC address string is `01:1b:19:00:00:00`
- `<port mask>` is a hexadecimal value where a bit corresponds to a port of the Kairos chip. If a bit is set to 1, a packet that matches the FDB entry will be sent to the corresponding port
    - Bit 0: cascading port
    - Bit 2: front port 1
    - Bit 3: front port 2
- `<priority>` is the traffic class (0..7) to assign to the packet that matches the FDB entry

NOTE: Kairos FDB entries can be managed also using the standard Linux tool `bridge`. However, due to the limitations in the options supported by `bridge`, the FDB entries are created with `<port mask>` set to "Cascading"

2.3.3.2 Delete an FDB entry

The command line syntax to delete an FDB entry is

```
kfdb del <MAC address>
```

where

- `<MAC address>` is the MAC address to add to the Kairos FDB database. The MAC address is made of six hexadecimal values separated by ':' (column). A valid MAC address string is `01:1b:19:00:00:00`

2.3.3.3 Query FDB entries

To show the current FDB entries, please type the following command

```
kfdb show
```

2.3.4 kvlan

`kvlan` is a command line utility to configure and delete VLAN membership

2.3.4.1 Configure VLAN membership

The command line syntax to add a port to a VLAN is

```
kfdb add <vid> <port> [untagged] [pvid]
```

where

- `<vid>` is the VLAN ID
- `<port>` is the port to add as a member to the given VLAN. Valid values are
    - 0: cascading port
    - 2: front port 1
    - 3: front port 2
- `untagged` if this keyword is added to the command line, the port will send untagged packets
- `pvid` if this keyword is added to the command line, the PVID will be set

2.3.4.2 Delete VLAN membership

The command line syntax to remove a port from a VLAN is

```
kvlan del <vid> <port>
```

where

- `<vid>` is the VLAN ID
- `<port>` is the port to add as a member to the given VLAN. Valid values are
    - 0: cascading port
    - 2: front port 1
    - 3: front port 2

2.4 ptp4k

The standard implementation of ptp4l (PTP For Linux) has been modified to use Kairos hardware timestamping registers. This new version has a new timestamping option (-K) to activate the Kairos timestamping

A complete documentation of the standard ptp4l command line options can be found at this link

https://www.mankier.com/8/ptp4l

These options are all supported by ptp4k as well. The only think to note is that the only timestamping option is "-K". This means that other timestamping options (-H, -S and -L - respectively hardware, software and legacy) can not be used with ptp4k

2.5 Control script

A helper script has been included in the software provided with the Kairos development kit. The script has options to

1. Start all the application required to run the board in master mode, which means
    a. The board runs as PTP master
    b. The board runs the demo application in master mode (i.e. publishes OPC UA data - see section below for details about the demo application)
    c. The board can generate packets to simulated network load conditions
2. Start all the application required to run the board in slave mode, which means
    a. The board runs as PTP slave
    b. The board runs the demo application in slave mode (i.e. subscribes to OPC UA data - see section below for details about the demo application)

Control script can be invoked with the following options

1. To start the board in master mode
   `demo/scripts/control.sh master on`

2. To stop a board running in master mode
   `demo/scripts/control.sh master off`

3. To start the board in slave mode
   `demo/scripts/control.sh slave on`

4. To stop a board running in slave mode
   `demo/scripts/control.sh slave off`

5. To start network traffic simulation
   `demo/scripts/control.sh netload on`

6. To stop network traffic simulation
   `demo/scripts/control.sh netload off`

2.6 Software folders

All the applications and scripts can be found on the board in the folder

```
/home/user/demo
```

2.6.1 Bin folder

All the executables are in this folder. In particular, the following files can be found here
- `ptp4k`: customized PTP stack
- `kspi`: command line utility to read and write Kairos registers
- `ktsn`: command line utility to configure Kairos Qbv scheduler
- `kfdb`: command line utility to configure Kairos FDB entries
- `ethtool`: command line utility to access Network card statistics
- `udpsend`: demo application that sends UDP packets
- `udprecv`: demo application that receives and decodes UDP packets
- `rawsend`: network traffic generator

2.6.2 Config folder

This folder stores the PTP configuration files, namely
- `slave.cfg`: configuration file to run ptp4k as a slave clock
- `master.cfg`: configuration file to run ptp4k as a master clock
- `boundary.cfg`: configuration file to run ptp4k as a boundary clock
- `qbv.txt`: Qbv scheduler configuration file

2.6.3 Scripts folder

This folder includes some bash scripts that invoke executables in the proper way. In particular, here you can find

- `control.sh`: script that starts PTP master/slave clock, configures VLANs, enables Qbv scheduler, starts network traffic simulation, etc
- `ptp-master.sh`: script that starts PTP stack as master clock
- `ptp-slave.sh`: script that starts PTP stack as slave clock
- `ptp-boundary.sh`: script that starts PTP stack as a boundary clock

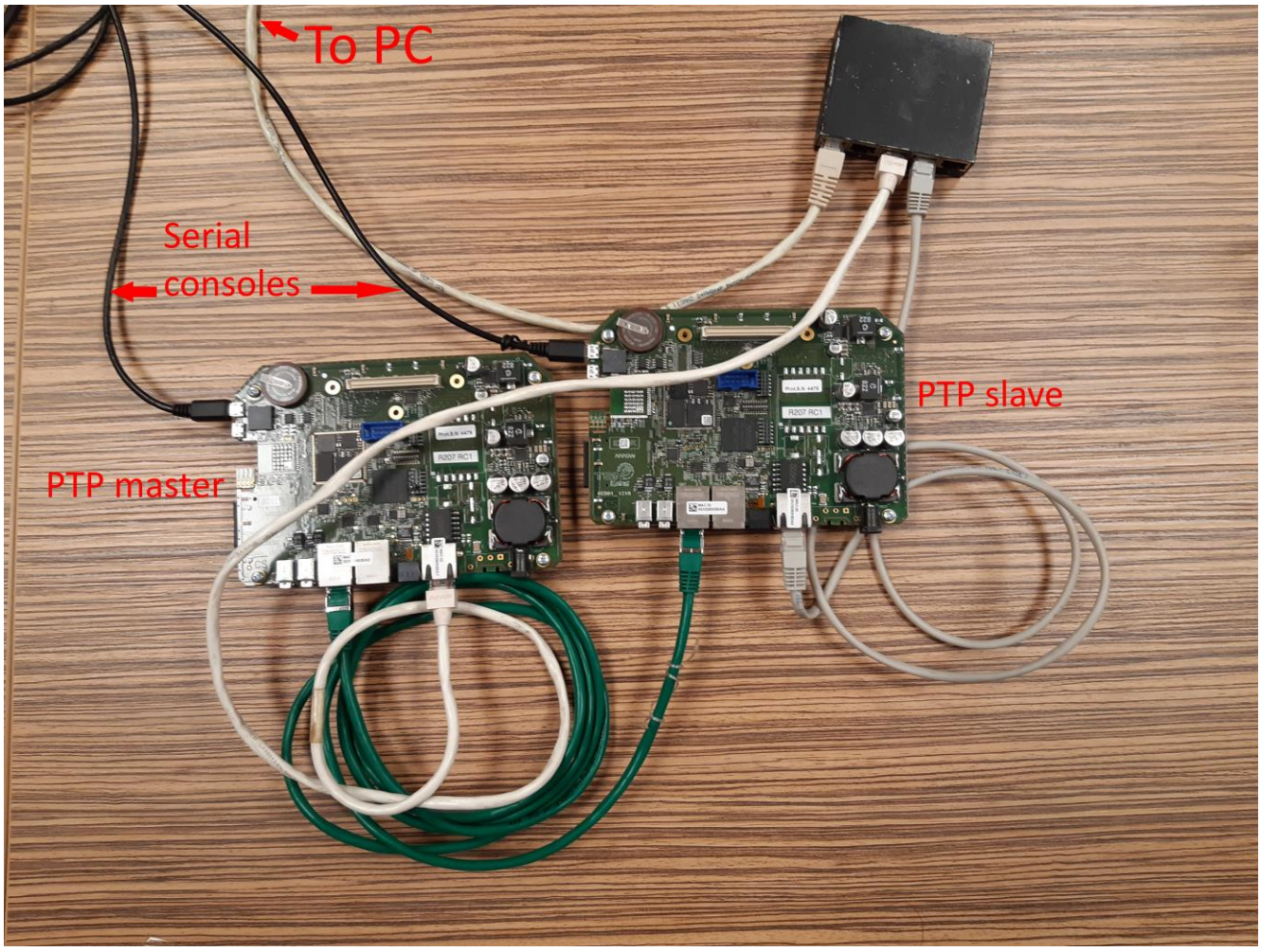
3 Demo application

The Kairos development kit can run a demo application to show highlight the capabilities of a TSN-capable switch. To run the demo application, you need two development kits: the first one running in master mode (i.e. sending out packets) and the second one running as slave (i.e. receiving packets and gathering statistics)
Before taking an in-depth look at how demo application works, we need to introduce the physical and logical connections required to run the application itself

3.1 Hardware connection

To setup the demo bench, you need to

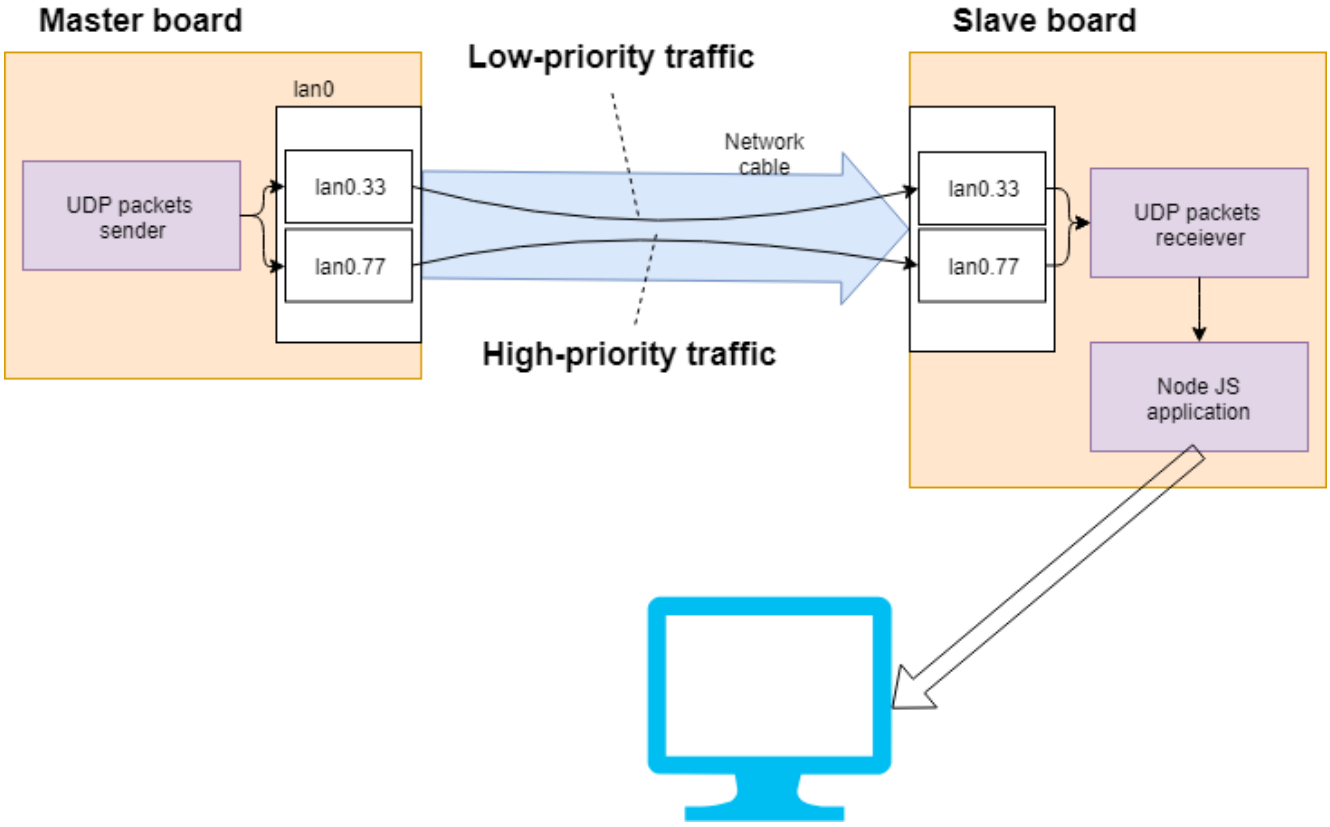
1. Connect Kairos' ETHA port on the master board to the Kairos' ETHA port of the slave board. Packets whose delivery time is measured are sent over this link
2. Connect the eth0 port of the master board to the switch device
3. Connect the eth0 port of the slave board to the switch device
4. Connect a PC to the switch device



3.2 Logical connection

The demo application uses two logical connections that share the physical link between the ETHA ports of the two boards

1. A best-effort connection on port lan0. PTP packets are sent through this network interface
2. Two Qbv scheduler-controlled connection on lan0.33 and lan0.77. These are two virtual network cards where packets are tagged with a specific VLAN ID. Kairos chip will be programmed to treat packets tagged with VLAN priority 7 as high-priority traffic and packets tagged with VLAN priority 3 as low priority traffic



3.3 Demo description

UDP packets are sent on logical connections lan0.33 and lan0.77. On the slave board, an application listens for incoming UDP packets and collects statistics for the received packets. Such statistics are shown by a node JS web application.

Under normal network traffic conditions, no differences can be seen in the packet delivery time. But if network is flooded with disturbing traffic, then the performances of the low-priority network drops whereas the performances of high-priority TSN traffic is not affected

This is just an example of how traffic can be shaped by means of the Kairos chip

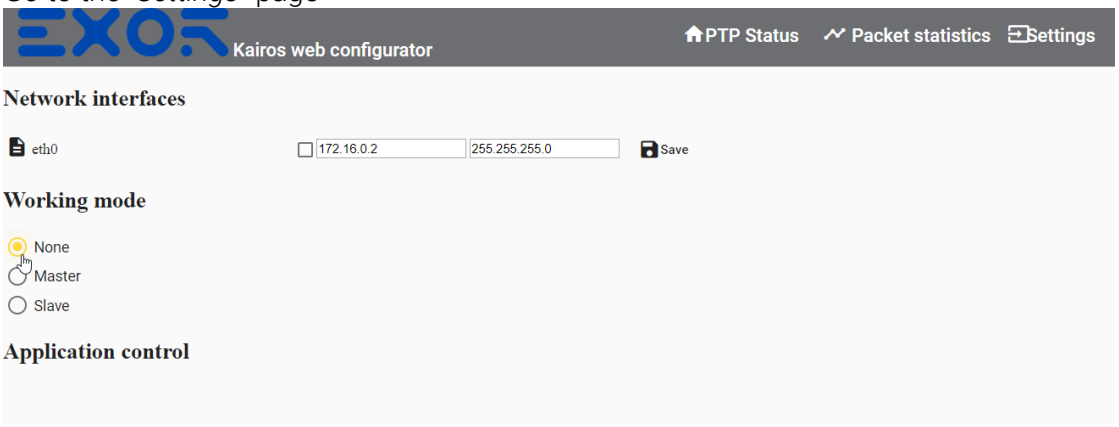
3.4 Walkthrough guide

3.4.1 Run demo application with no network traffic

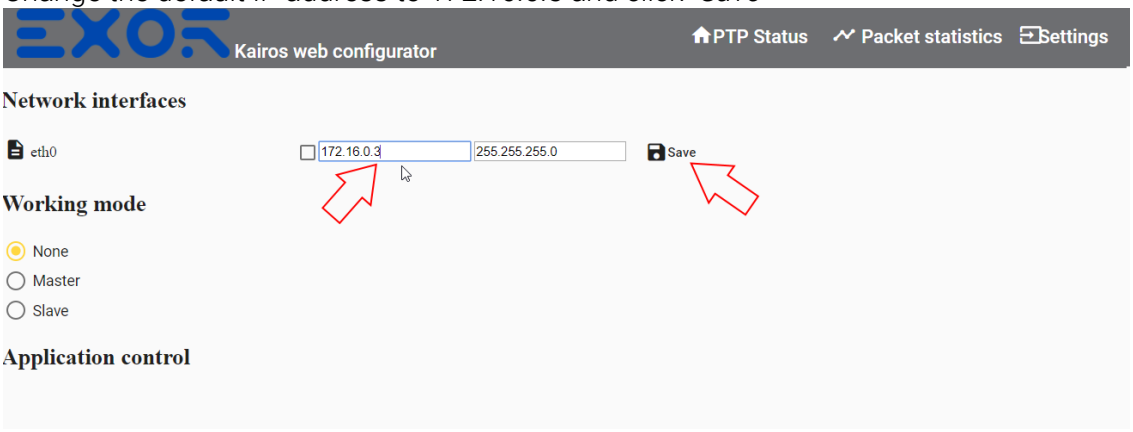
1. Switch on the master board (since boards are shipped with a fixed IP address, leave the slave board switched off to avoid conflicts)

2. Open a browser at the following address
   http://172.16.0.2:3000

3. Go to the *"Settings"* page

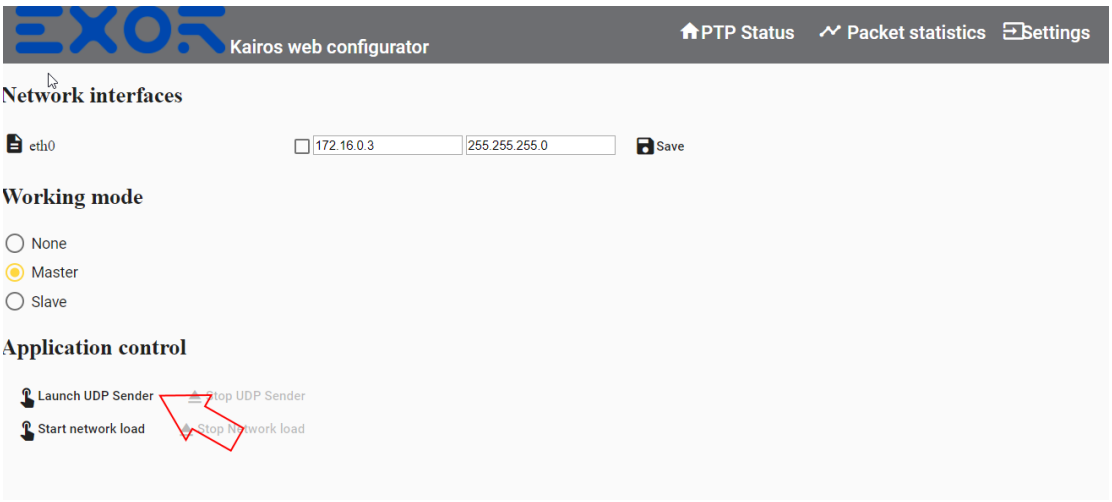


4. Change the default IP address to 172.16.0.3 and click *"Save"*



5. Select *"Master"* in the *"Working mode"* section

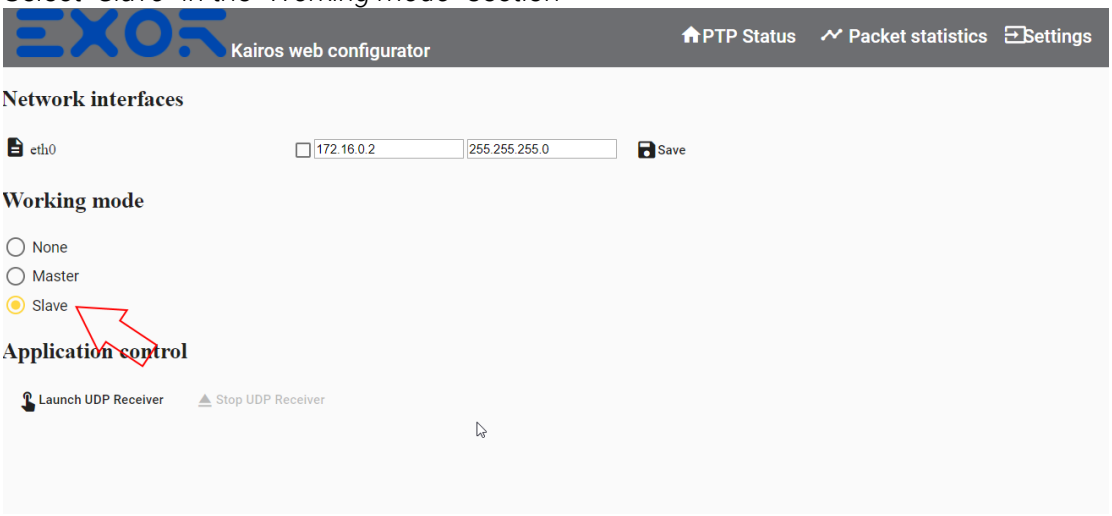
6. Click *"Start UDP sender"* button



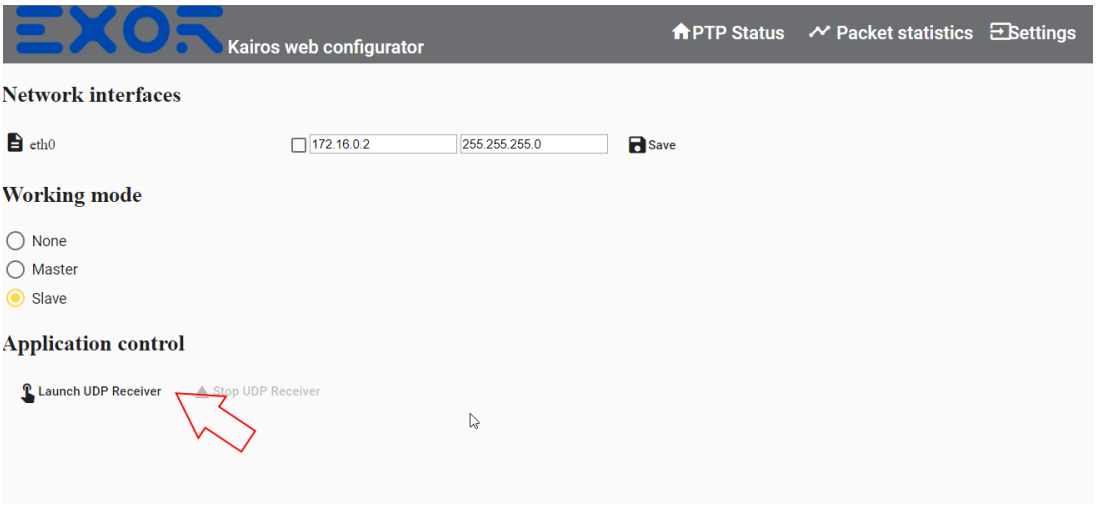
7. Open a browser at the following address
   http://172.16.0.2:3000

8. Go to the *"Settings"* page

9. Select *"Slave"* in the *"Working mode"* section



10. Click *"Start UDP receiver"* button



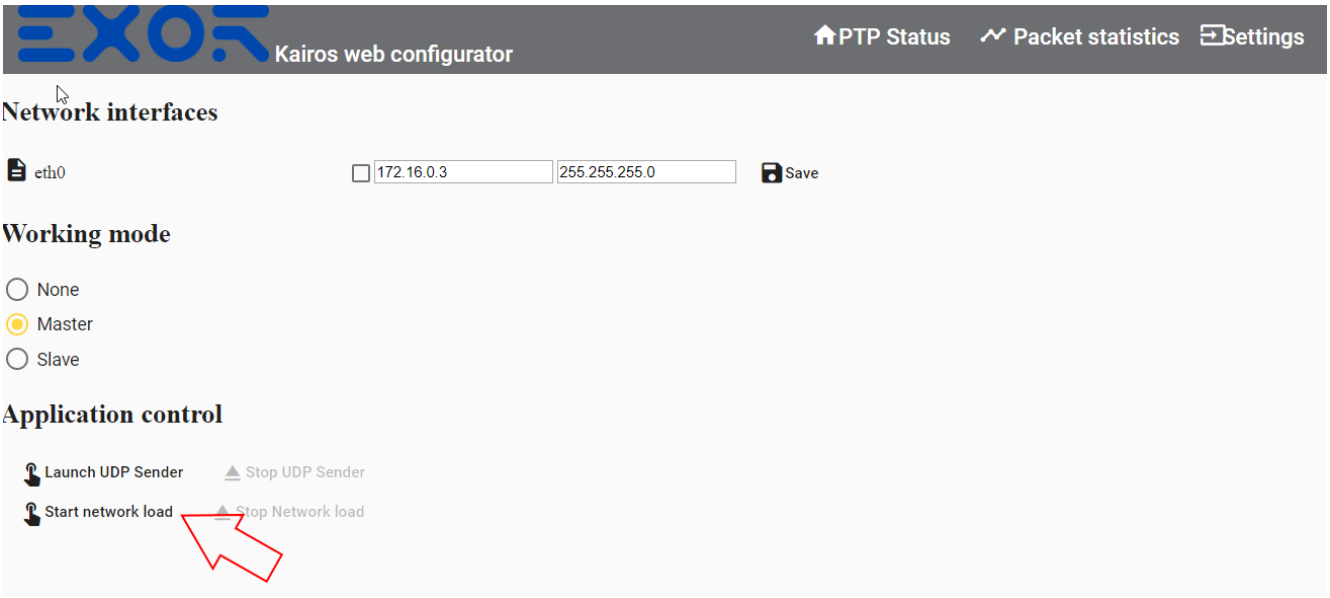
11. Go to *"Statistics"* page. You should see that there are no dropped packets and packets delivery time is the same for both low-priority and high-priority connections

3.4.2 Activate network traffic

1. Open a browser at the following address
   http://172.16.0.3:3000

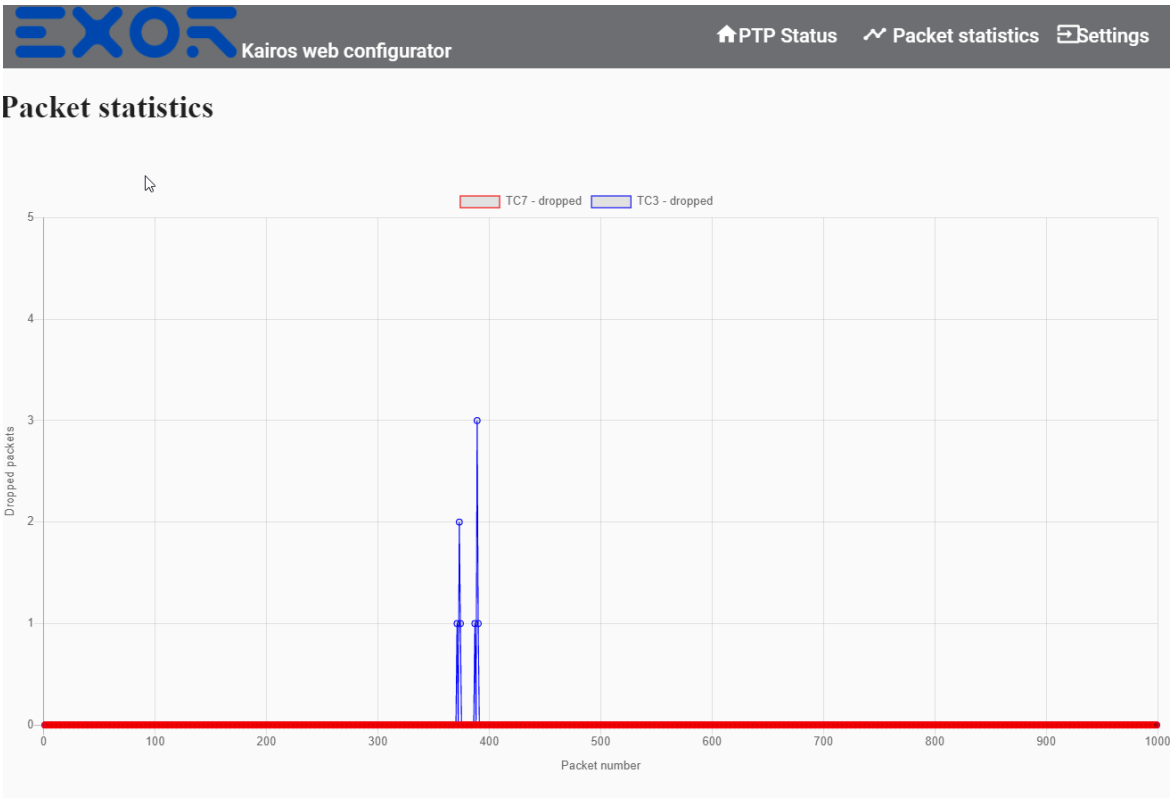
2. Go to the *"Settings"* page

3. Click *"Start network load"* button



4. Open a browser at the following address
   http://172.16.0.2:3000

5. Go to *"Statistics"* page. You should see that there are many dropped packets on the low-priority connection

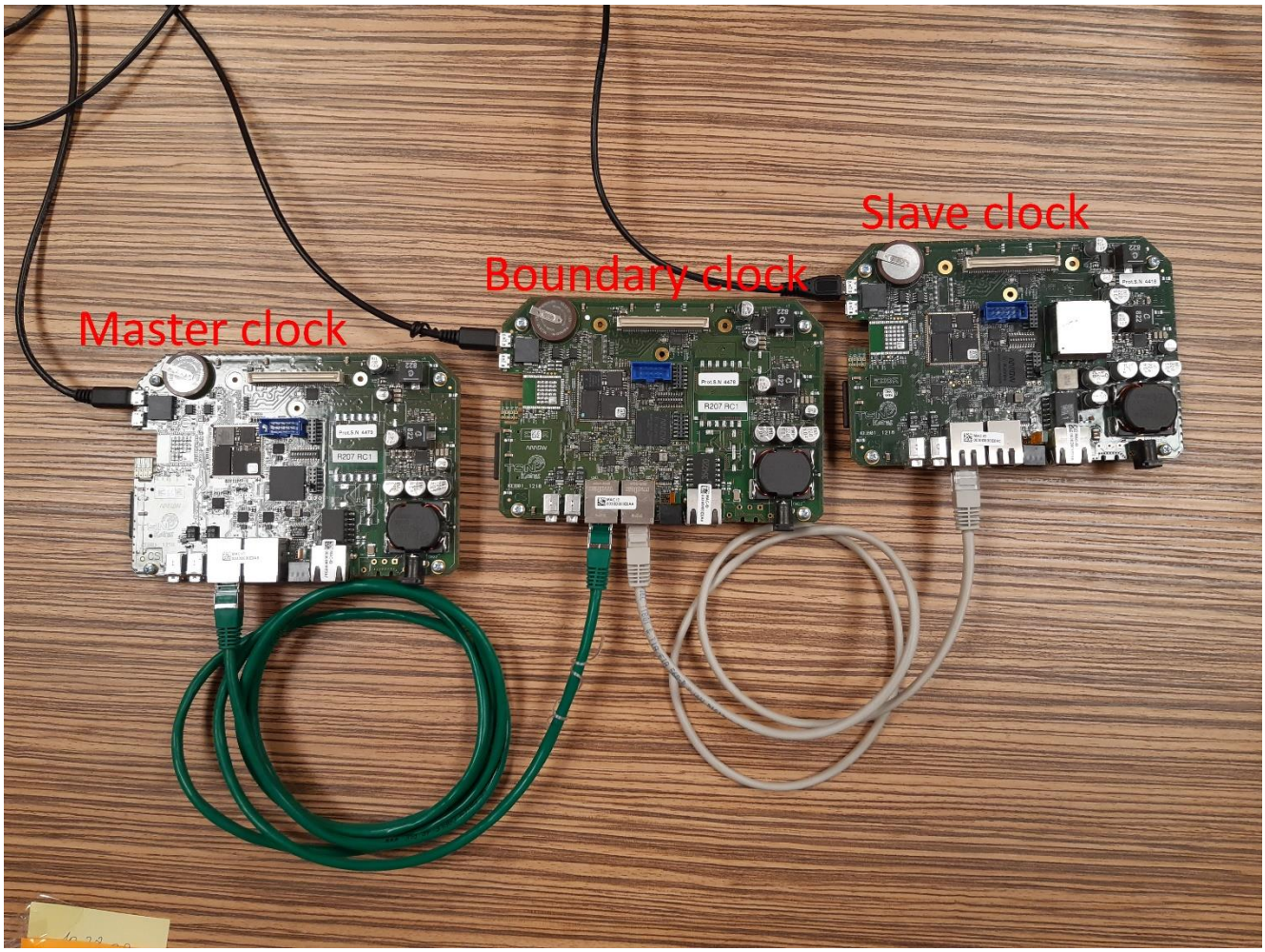


4 Boundary clock support

ptp4k implements boundary clock. To test this feature, you need three boards connected in this way

- Connect lan0 of board that should be used as a master clock to lan0 of the board running as boundary clock
- Connect lan1 of board running boundary clock to lan0 of the board running as slave clock

Hardware connections are shown in picture below



5 Setting up the build environment

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 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 Compiling Yocto BSP from scratch. if you are interested in building the BSP or chapter 10 if you are interested in building your own applications for the target.

5.1 Running the VirtualBox VM

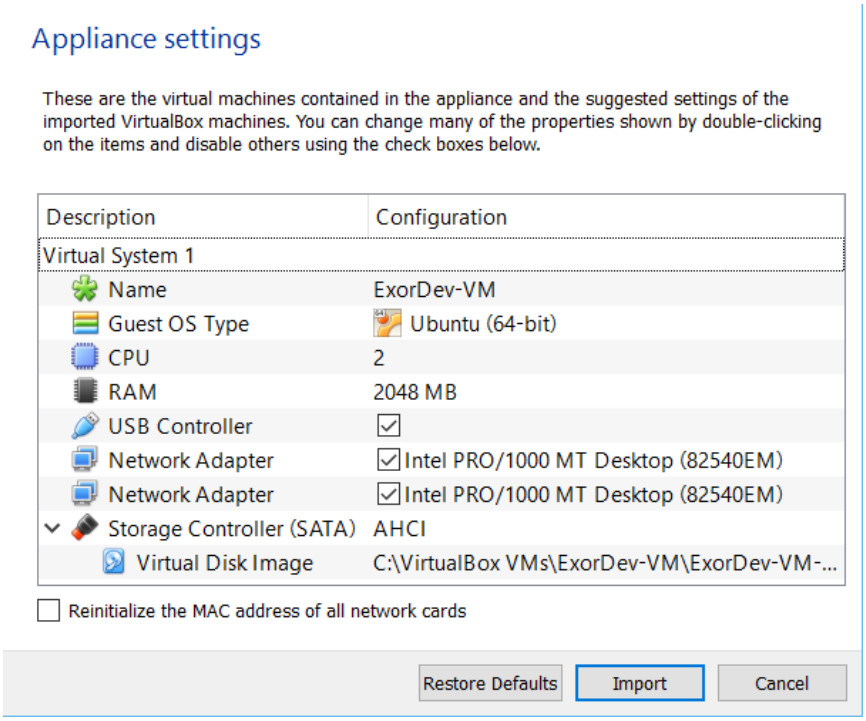
You can download the Exor's VirtualBox development VM from here:

```
http://download.exorembedded.net:8080/Public/VirtualBoxVMs
```

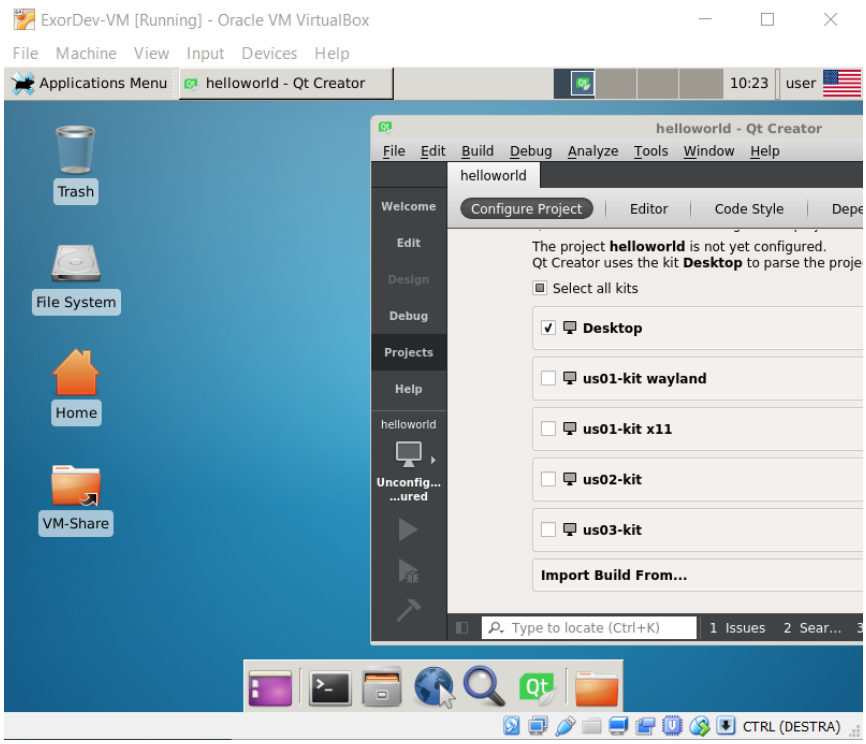
Instructions found on this document are compatible with versions 4.x of the VM. If you are about to use a greater version, please consider looking for an updated version of this manual.

The virtual machine comes in the OVA (Open Virtualization Archive) format. To import it on VirtualBox go 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 and IP address must be used in this case.



The default amount of RAM is set to 2GB but if you plan to work with Yocto, we recommend increasing 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"*. When import process terminates, you will be able to change VM settings again



The Linux operating system used is based on Ubuntu 16.04. The default user is:

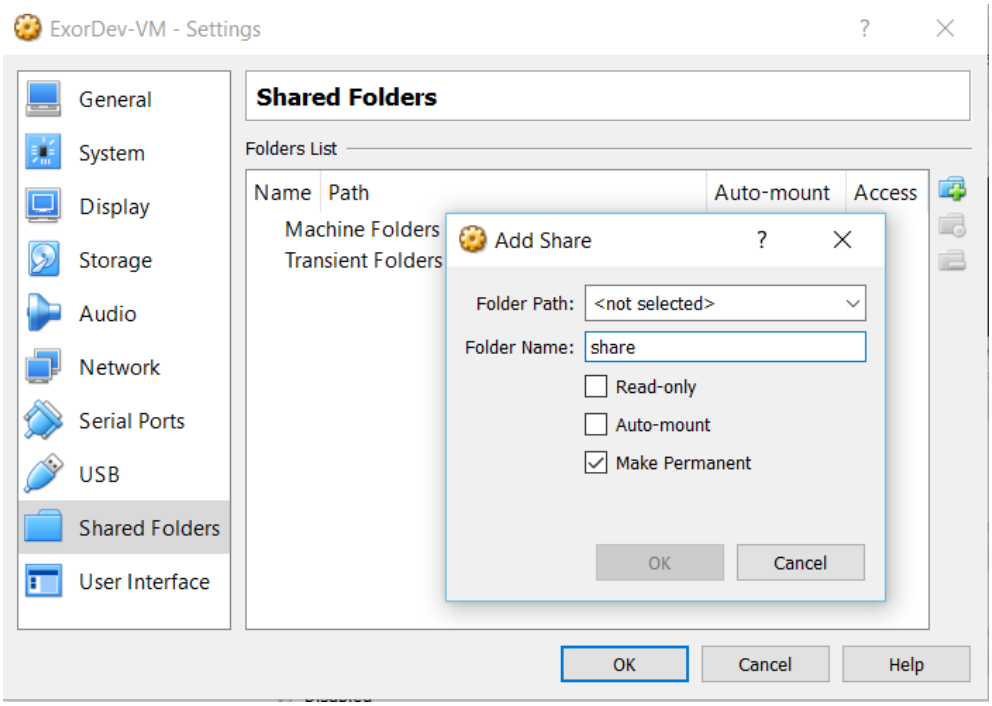
- username: *user*
- password: *password*

To run a command with root privileges, you can use `sudo` (password is not required)

5.1.1 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 the add button on the right. Configure options as follow:

- *Folder Path*: choose the host folder to share with the virtual machine
- *Folder Name*: must be `share`.
- *Read-only*: leave unflagged
- *Auto mount*: leave unflagged.
- *Make Permanent*: flag this option.



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. If the VM was already running, a restart is required

5.1.2 Configuring the SDK

To reduce the initial weight of the VM, the SDK is not shipped with it. A script named *"Install NSOM SDK.sh"* can be found on the desktop. By just double-clicking the link on the desktop, the SDK will be downloaded an installed.

During the installation of the SDK, QtCreator will be reconfigured and a new kit will be added.

5.1.3 Using QtCreator

The QtCreator IDE is already installed and configured to deploy and debug applications for each development kit. When creating a new project, please make sure to select the kit configuration for NSOM device. If not available, make sure that the SDK has been installed using the script that can be found on the desktop. There's also a *"Desktop"* kit configuration which can be used to build your application and run it on the virtual machine instead of deploying it, useful for fast testing and 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, how to change the hostname or IP address of the target device.

5.1.4 Compiling the BSP with Yocto

Inside `/home/user/exor-yocto-4.0` you will find the preconfigured Yocto workspace for building the BSP for your development kit. Before starting the build, please update the meta-exor layer to get the latest version of the recipes:

```
$ cd /home/user/exor-yocto-4.0/git/meta-exor
$ git checkout rocko
$ git pull
```

See section 6 below in this document to go ahead compiling the BSP.

6 Compiling Yocto BSP from scratch.

In this section, step-by-step guide to build the BSP is provided

6.1 Setup the build environment

If you are using Exor's VirtualBox VM you can skip the first two steps: you will find the `exor-yocto-4.0` folder already in the user's home (`/home/user/exor-yocto-4.0`).

1. Create a workspace directory structure:

```
$ mkdir -p exor-yocto-4.0
$ cd exor-yocto-4.0/
```

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 rocko
$ ./repo sync
```

3. Setup the Yocto environment. From the `exor-yocto-4.0` 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-4.0/build`. The source command above

4. Configure Yocto by copying the provided sample configuration files. From 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` file and set the `MACHINE` variable to `ns01-kit`

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

6. You are now ready to build the BSP.

6.2 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"
```

6.3 Compiling Yocto BSP

Make sure to run following commands from your `build` folder:

1. Compile the bootloader:

```
$ bitbake bootloader
```

2. The Linux kernel:

```
$ bitbake virtual/kernel
```

3. 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/nsom01`:

`ns01-kit-uboot.tar.gz`	U-Boot raw image
`ns01-kit-kernel.tar.gz`	Kernel (zImage) and dtb
`core-image-exor-[…]-ns01-kit.tar.gz`	Root File System

6.4 Creating the SDK (optional)

Start the SDK build for the x11 image:

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

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

7 Deploy BSP on SD card

This section describes how to prepare a bootable SD-card for the development 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 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.

7.1 Using an SD card 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 development kit:

```
http://download.exorembedded.net:8080/Public/nsom01/sdcard-images/
```

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.

7.2 Using BSP packages

To create a bootable SD card from the BSP packages build in chapter 6, please follow these steps

1. Create the SD-card partition layout :

```
# umount /dev/sdb*
# SIZE=`fdisk -l /dev/sdb | grep -m1 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
```

3. Now run these commands to perform the actual deploy

```
// Deploy files to SD-card for ns01-kit
# mkdir /media/BOOT/boot
# tar xzvf ns01-kit-kernel-[…].tar.gz --no-same-owner -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
```

8 Deploy BSP on eMMC

This section describes how to deploy the BSP on eMMC and boot from it. All the kits have the possibility to boot without an SD-Card except for the us03-kit.
On the iMX6Q the location where the bootloader needs to loaded is defined by OTP fuses that on the us03-kit are already set to use the SD-Card. Once the bootloader is loaded into ram the roots used will still be the one on the eMMC and the SD-card could be removed. For more information on OTP fuse programming please refer to NXP processor reference manual (chapter 5 *Fusemap* and chapter 46 *On-Chip OTP Controller*):

https://www.nxp.com/docs/en/reference-manual/IMX6DQRM.pdf

To deploy the BSP to the internal eMMC it is required to define the 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, `uboot.img`
- The rootfs, `core-image-exor.tar.gz`
- Kernel and dtb or a `kernel.tar.gz`

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 `/dev/mmcblk1` on all the development kits except for the us02-kit where it's `/dev/mmcblk0`, for this reason the operation is slightly different for the latter.

```
// Format eMMC and mount rootfs partition
# umount /dev/mmcblk1p*
# SIZE=`fdisk -l /dev/mmcblk1 | grep -m1 Disk | awk '{print
$5}'`
# CYLINDERS=$(( $(( $SIZE )) / 255 / 63 /512 ))
# echo -e "o\nn\np\n1\n2\n\nw" | fdisk -H 255 -S 63 -C $CYLINDERS
/dev/mmcblk1
# mkfs.ext4 /dev/mmcblk1p1
# mkdir emmc
# mount /dev/mmcblk1p1 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

```
// Deploy bootloader on eMMC
# dd if=u-boot.imx 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 connect to the kit's serial port using a client like putty and while keeping pressed 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
# setenv mmcboot 'run findfdt; mmc rescan; ext2load mmc 1:1 ${loadaddr}
/boot/zImage; ext2load mmc 1:1 ${fdtaddr} /boot/${fdtfile}; setenv mmcroot
/dev/mmcblk1p1; run mmcargs; bootz      ${loadaddr} - ${fdtaddr};'
# 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
```

9 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.

9.1 Cross development environment setup

Download the latest v4.x SDK from here:

```
http://download.exorembedded.net:8080/Public/nsom01/SDK
```

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/ns01-kit/`. 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 ns01-kit
$ source /opt/exorintos/ns01-kit/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
```

9.2 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:

```
exorNS01kit.local
```

9.3 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
```

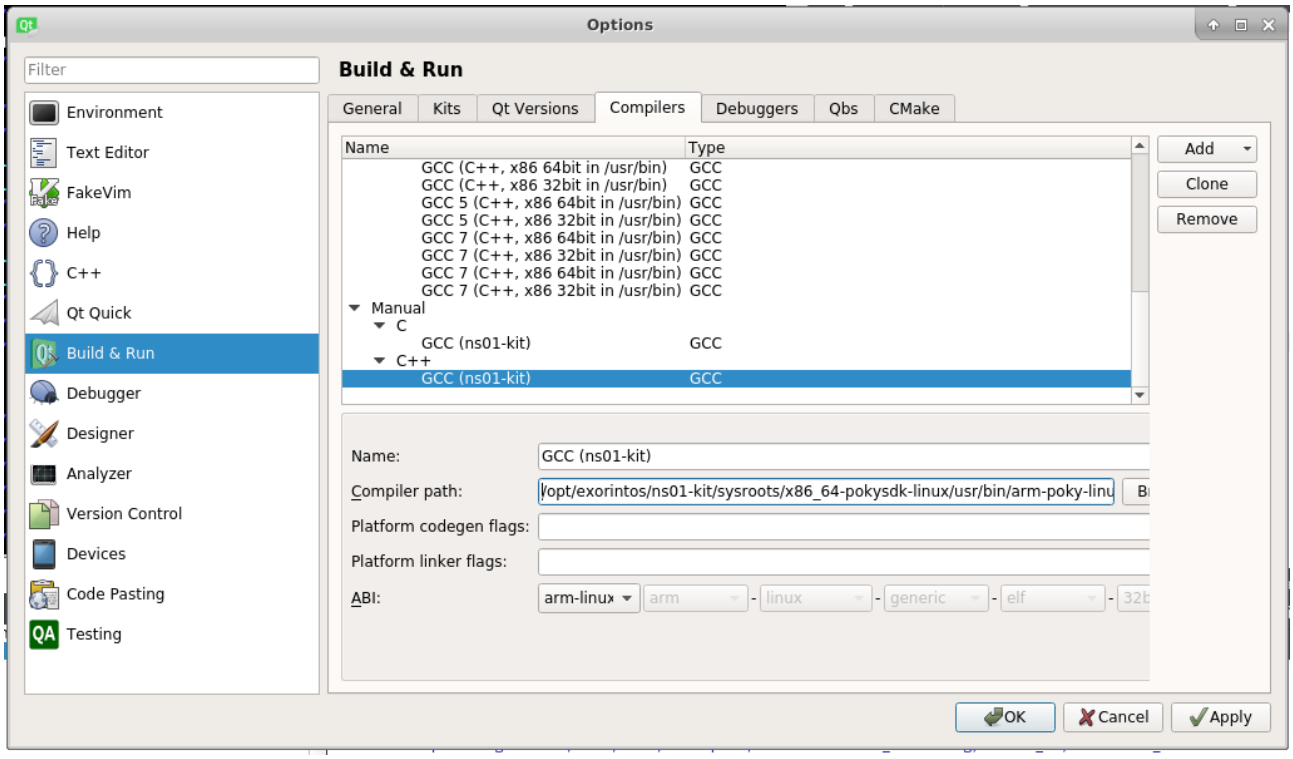
You will find QtCreator installed in `~/qtcreator-3.3.2`. Start it:

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

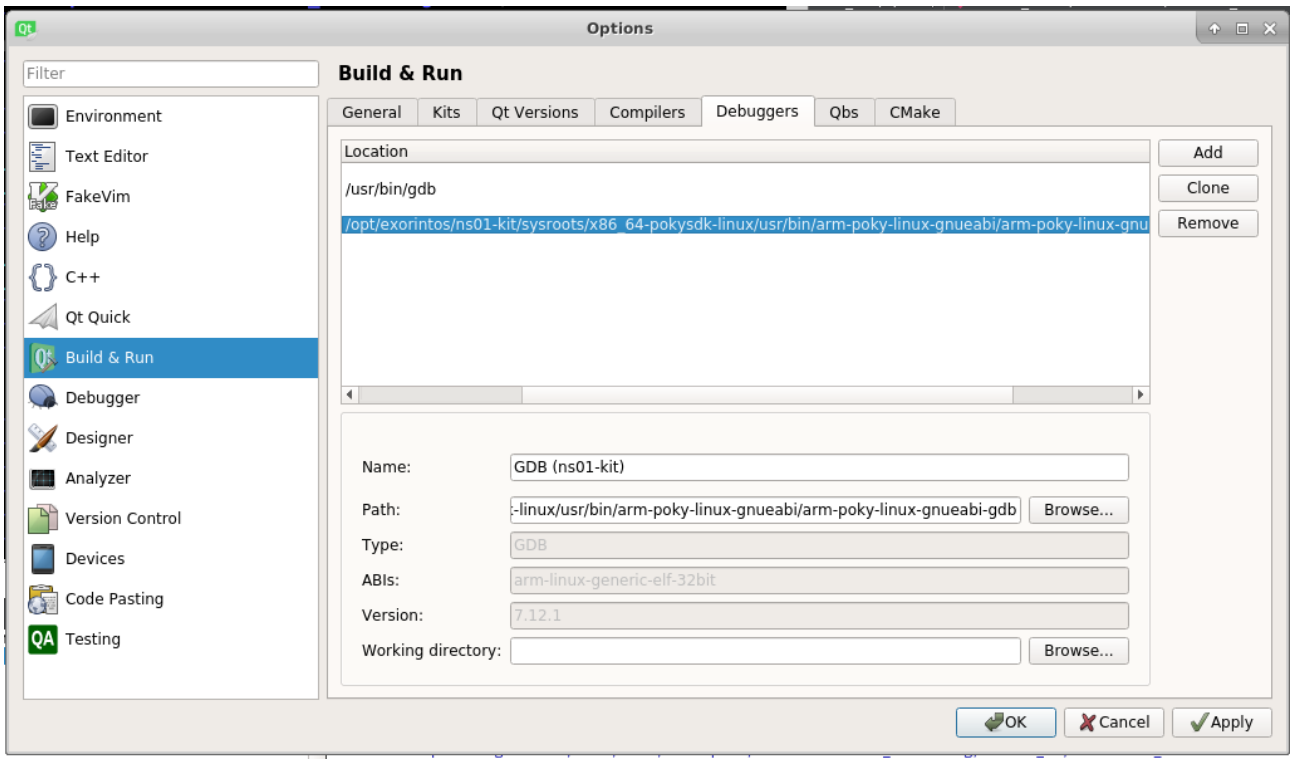
We are now going to setup the QtCreator build kit for the target.
From Tools menu select *"Options..."* → *"Build & Run"*, then follow these steps:

1. In the *"Compilers"* tab click on *"Add"* → *"GCC"* → *"C"* 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/ns01-kit/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, select *"arm-linux-generic-elf-32bit"* as ABI and finally click *"Apply"*

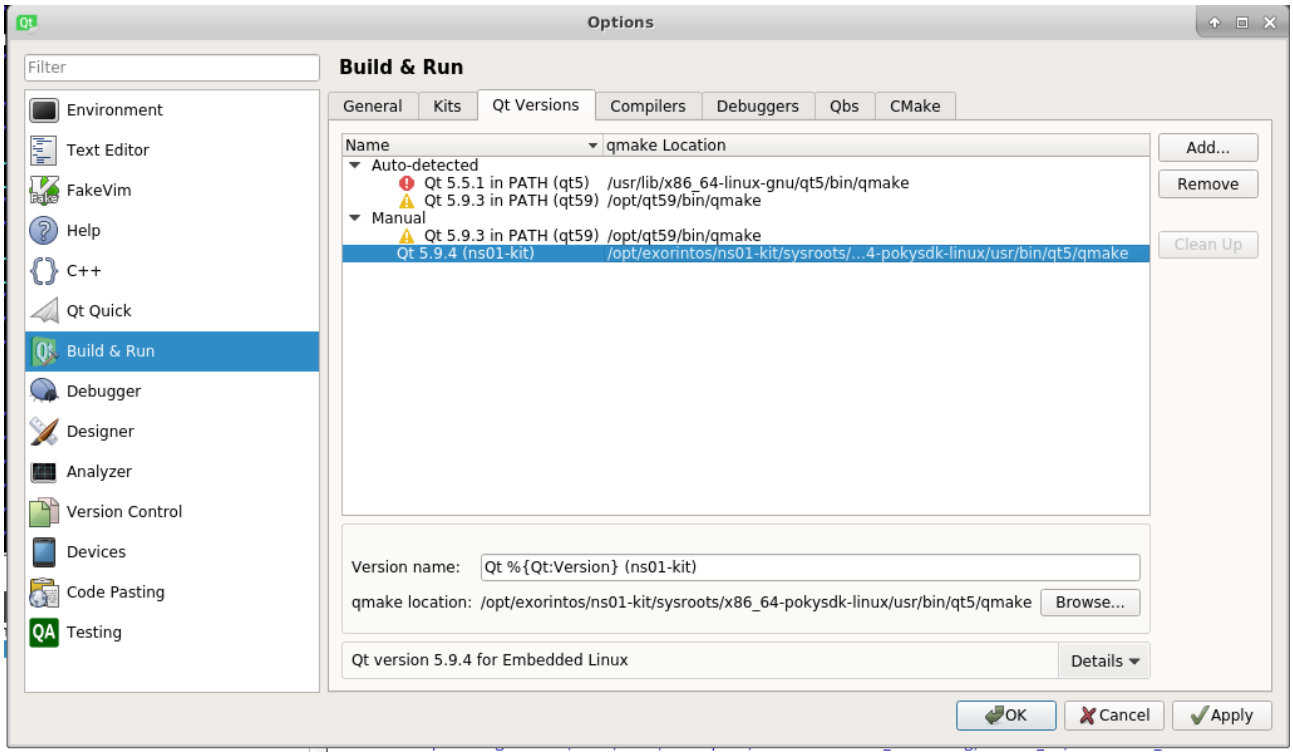
2. From the same tab, click *"Add"* → *"GCC"* → *"C++"* and select `/opt/exorintos/ns01-kit/sysroots/x86_64-pokysdk-linux/usr/bin/arm-poky-linux- gnueabi/arm-poky-linux-gnueabi-g++` instead. Again, select *"arm-linux-generic-elf-32bit"* as ABI and click *"Apply"*



3. From *"Debuggers"* tab, press *"Add"* and select gdb from the same directory. The default location is `/opt/exorintos/ns01-kit/sysroots/x86_64-pokysdk-linux/usr/bin/arm-poky-linux-gnueabi/arm-poky-linux-gnueabi-gdb` Optionally edit *"Name"*, then click *"Apply"*



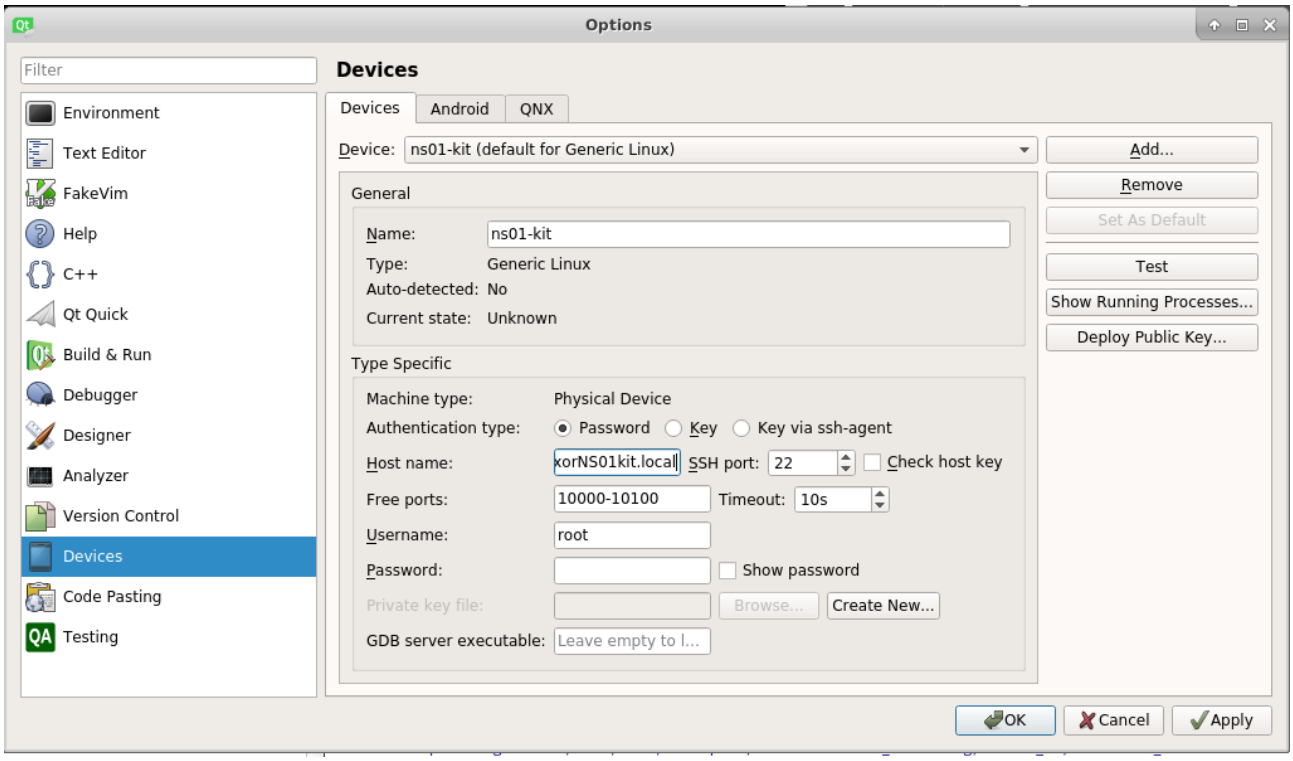
4. From *"Qt Versions"* tab, press *"Add"*. The default path to select is: `/opt/exorintos/ns01-kit/sysroots/x86_64-pokysdk-linux/usr/bin/qmake`. QtCreator should automatically recognize the qt version selected. Press *"Apply"*



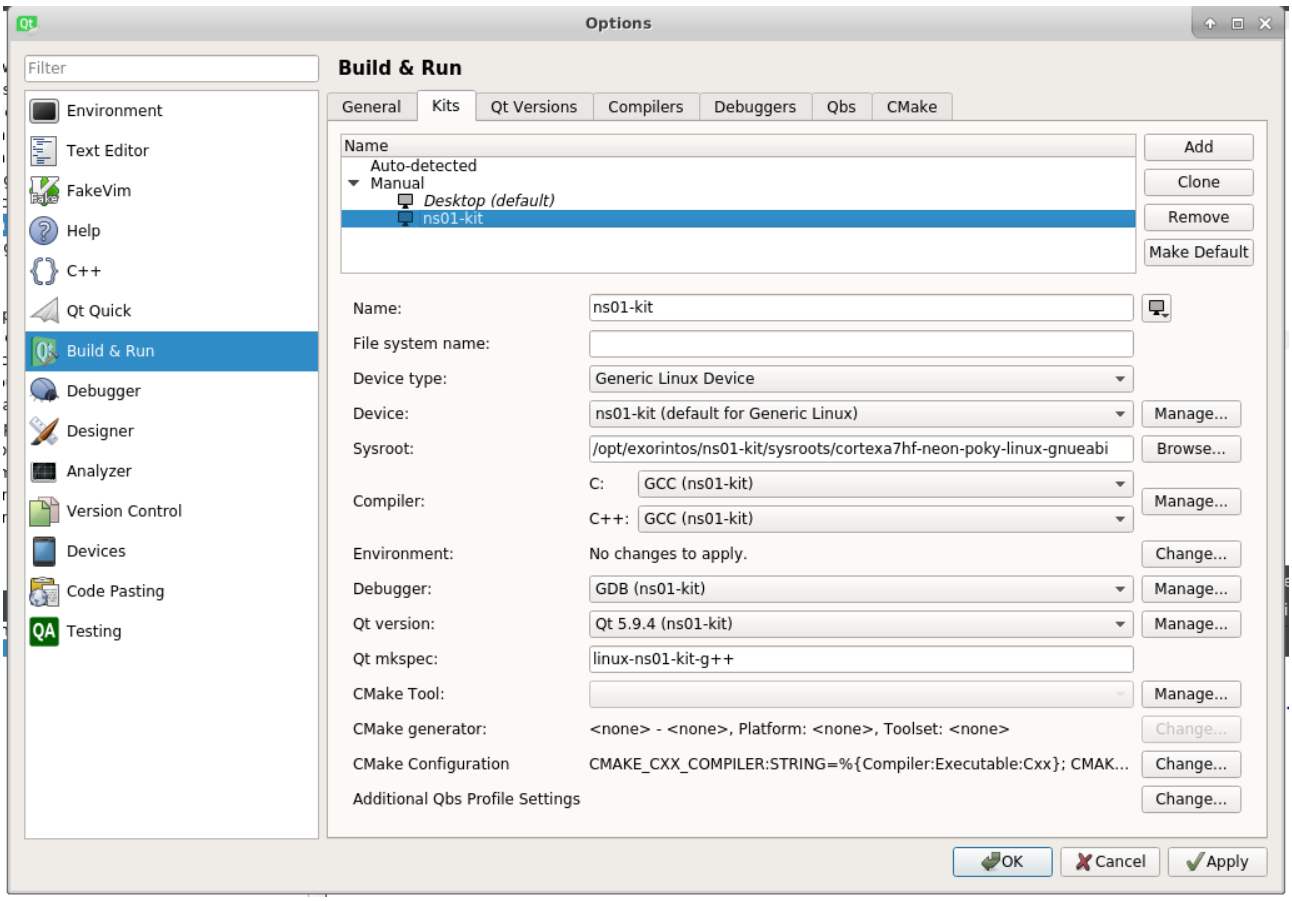
5. This step is required for configuring automatic application deploy to the target.
   Move from *"Build & Run"* section to *"Devices"*. Click *"Add"*, select *"Generic Linux Device"* and press *"Start Wizard"*. Fill in this information:
   - *Name*: the device name, `ns01-kit`

- *Host name*: `exorNS01kit.local`
- *Username*: `root`
- *Authentication type*: set to *"Password"*
- *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.



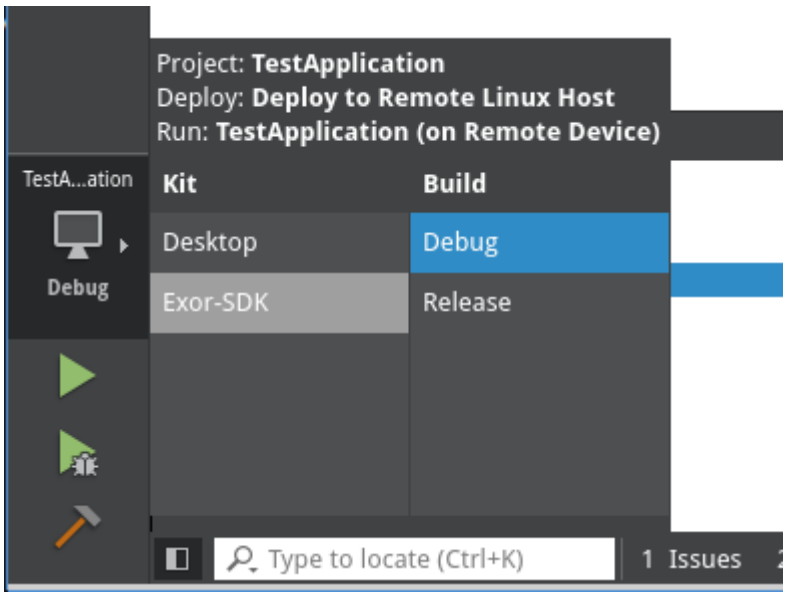
6. Finally move again to *"Build & Run"* section, *"Kits"* tab. Combine all pieces together in a new kit. Click *"Add"* and fill in as follows:
   - *Name*: choose a name for the kit
   - *Device Type*: select *"Generic Linux Device"*
   - *Device*: select the device configured in step 5
   - *Sysroot*: if the SDK is installed in the default location, these are the paths to select: `/opt/exorintos/ns01-kit/sysroots/cortexa7hf-neon-poky-linux-gnueabi`
   - *Compiler*: select C and C++ compilers by name as configured in step 1 and 2
   - *Debugger*: select debugger by name as configured in step 3
   - *Qt version*: select Qt version added in step 4



9.4 Application deployment

Before starting here, make sure QtCreator has been correctly configured for application deployment and that the development kit is reachable.

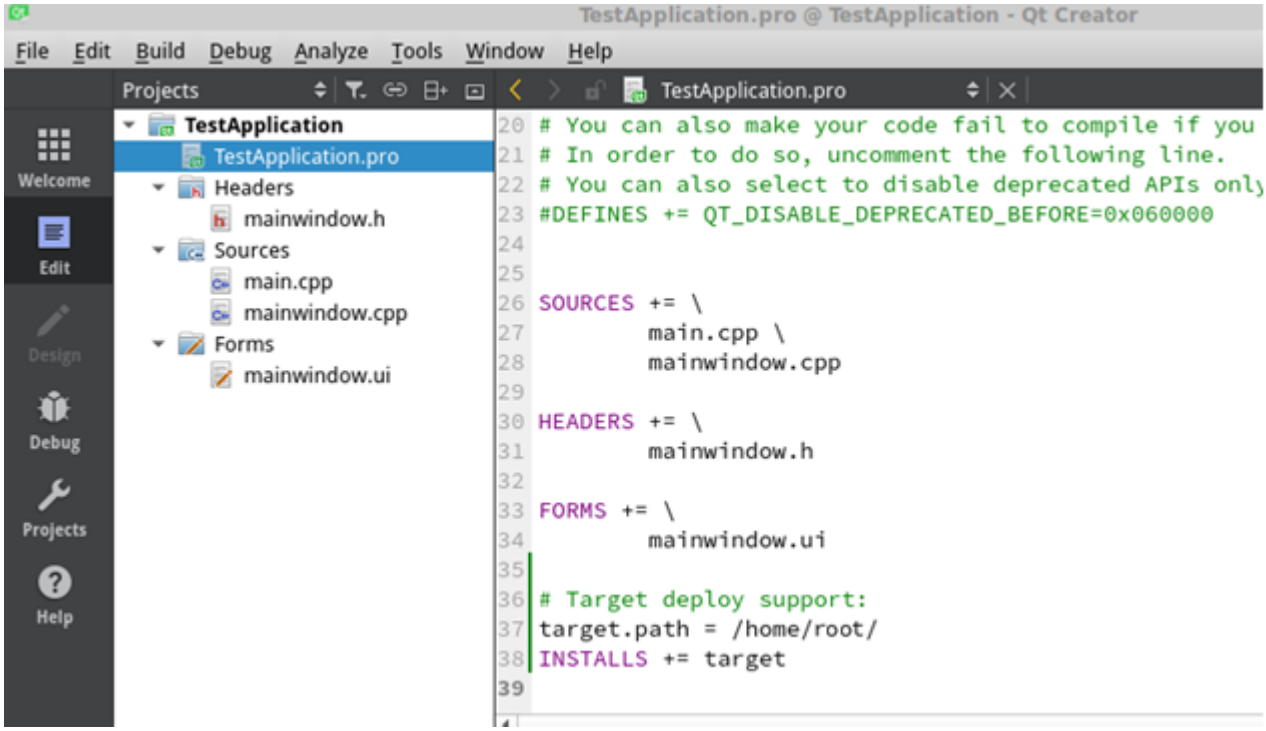
1. First, let's create a dummy Qt project. Select *"File"* → *"New File or Project..."* → *"Qt Widgets Application"* and click *"Choose"*. Enter a project name, press *"Next"*. Make sure that in the *"Kit Selection"* wizard dialog the SDK kit for the target is selected.
2. Make sure the target kit currently in use by checking in the menu on the left shown below:



3. Now edit the .pro project file to add these two lines:

```
target.path = /home/root/
INSTALLS += target
```

This will define where the application will be installed on the device (`/home/root`)



4. Finally press the green *"play"* button in the menu on the left or use the "Ctrl+R" shortcut. QtCreator should compile the application and an empty Qt window should appear on the device

10 Building the development kit applications

After setting up the build environment, we can build the applications included in the Kairos development kit from the source code

10.1 Folders structure

The provided source code is organized (starting from the user's home directory) are follow

`workspace`	
`ptp4k`	PTP stack. This a customized version of the ptp4l Linux application
`sdk`	Qt project that contains all the applications to access Kairos chip
`kairos_rw`	Kairos registers access library
`kspi`	Command line utility to read and write Kairos registers
`kfdb`	Command line utility to configure Kairos FDB entries
`ktsn`	Command line utility to configure Kairos TSN scheduler
`pubsub`	
`udpsend`	Demo application that runs on the master board (OPC UA publisher)
`udprecv`	Demo application that runs on the slave board (OPC UA subscriber)
`rawsend`	Application to generate traffic on the network
`Web`	
`kairos-node`	Angular project of the frontend of the web interface
`kairos-web`	Node JS backend application of the web interface

10.2 Building ptp4k

Ptp4k is provided as QtCreator projects. To build the application,

1. Launch QtCreator
2. Open QtCreator project ptp4k.pro located in the folder `workspace/ptp4k/ptp4k`
3. Build and run the application as explained in section *"Application deploy"*

10.3 Building kspi, ktsn and kfdb

kspi and ktsn are provided as QtCreator projects. To build the applications,

4. Launch QtCreator
5. Open QtCreator project sdk.pro located in the folder `workspace/sdk`
6. Build and run the application as explained in section *"Application deploy"*

10.4 Building web interface

10.4.1 Building the backend

Node JS applications are not built. To run the backend, simply invoke node followed by the name of the javascript file to execute

First, install node version 8 or greater

```
$ sudo apt install curl
$ curl -sL https://deb.nodesource.com/setup_10.x | sudo -E bash -
$ sudo apt install nodejs
```

Check if node is correctly installed

```
$ node --version
```

You can now run the backend application

```
$ cd workspace/web/kairos-node
$ node ./kairos.js
```

10.4.2 Building the frontend

To build the frontend application, Angular is required. To install angular, first check if npm is properly installed

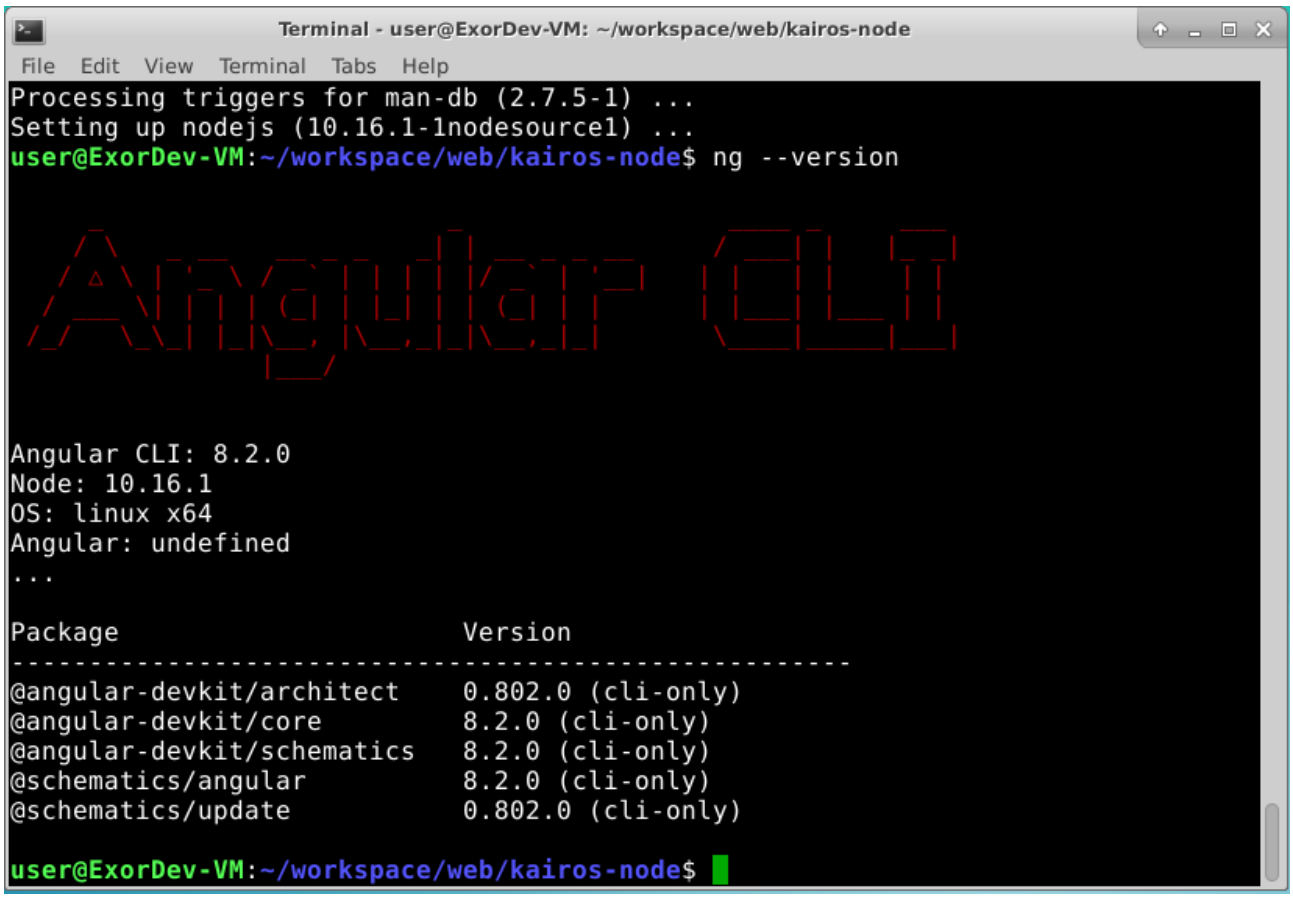
```
$ npm --version
```

If the above command does not return any error, you can install Angular

```
$ sudo npom install -g @angular/cli
```

Finally, check if Angular has been installed correctly

```
$ ng --version
```



Now the frontend application can be built

```
$ cd workspace/web/kairos-web
$ ng build --prod
```

Packaged files are now available in `workspace/web/kairos-web/dist/kairos-web`. To deploy the frontend application, copy all the files in the `files` subfolder of the backend

11 OpenHMI Portable runtime

OpenHMI 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. OpenHMI 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 OpenHMI is a standard Linux OpenHMI runtime provided as a chroot-based container designed to run under Linux 32bit ARM platforms.
The portable OpenHMI runtime is provided for rapid prototyping and evaluation purposes and contains a subset (Codesys V3/, Modbus and the internal variables protocol) of the available protocols. For example, serial protocols are not supported, since the serial ports on the evaluation kits are 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.

11.1 OpenHMI portable runtime installation

By default, OpenHMI in preinstalled on both the standard SD image and the rootfs generated by our standard Yocto recipes.
The portable can however also be downloaded separately from here:

```
http://download.exorembedded.net/Public/OpenHMI/
```

Then to install and run it from ssh follow these steps:

1. Copy it into the kit.
   ```
   scp jmobile-[…]-portable-devkit.tar.gz root@[hostname]:~
   ```

2. Connect to the kit:
   ```
   ssh root@[hostname]
   ```

3. Now, from the remote shell, untar the package in a folder with write permissions (e.g. /opt)
   ```
   tar xzpf jmobile-[…]-portable-devkit.tar.gz
   rm -rf jmobile-[…]-portable-devkit.tar.gz
   ```

4. Start OpenHMI
   ```
   jmobile_portable/run.sh
   ```

11.2 Run OpenHMI portable runtime at boot

In both cases it's possible to configure the BSP to automatically start OpenHMI Runtime at boot:

1. Remove the script xserver-nodm:
   ```
   update-rc.d -f xserver-nodm remove
   ```

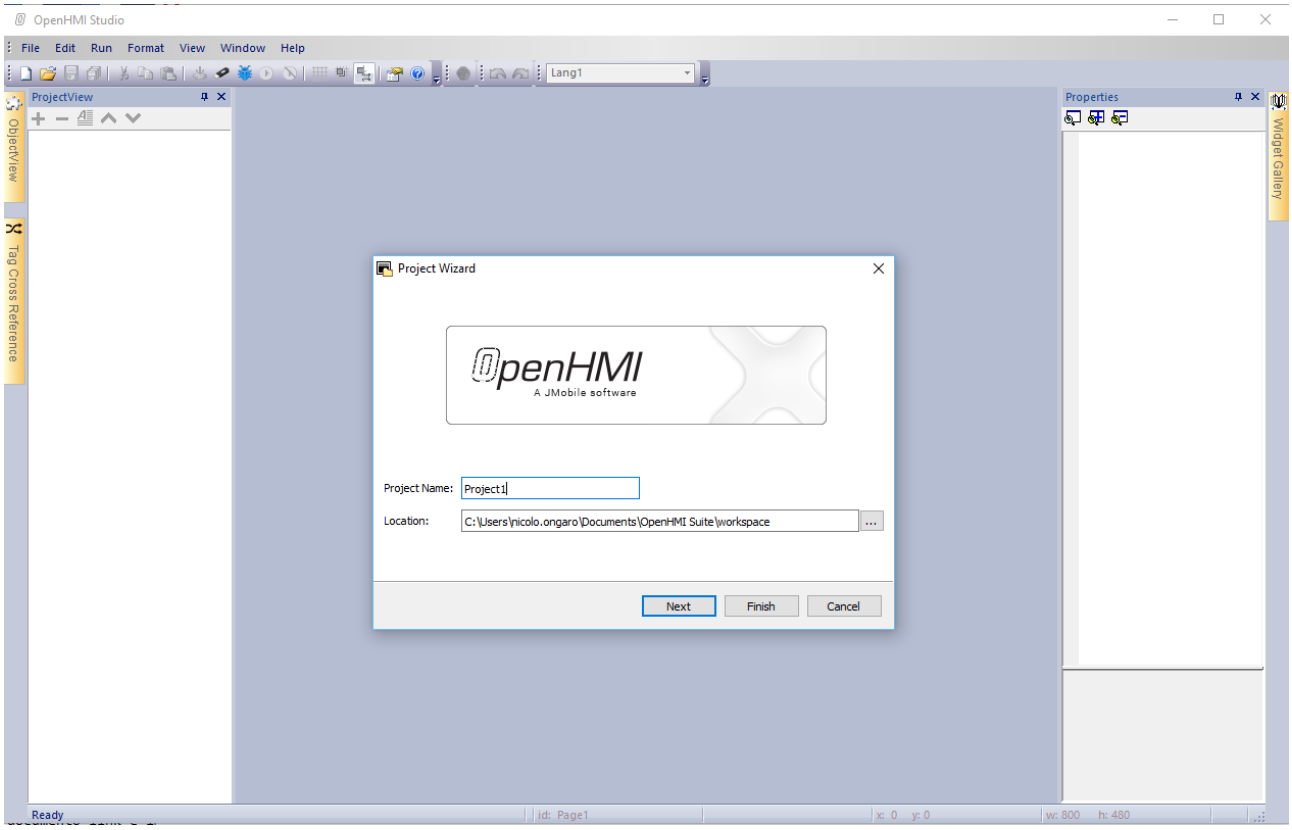
2. Add a new script to the init sequence:
   ```
   echo "/home/root/jmobile_portable/run.sh &" > /etc/init.d/jmobile
   chmod a+x /etc/init.d/jmobile
   update-rc.d jmobile defaults 99
   ```

11.3 OpenHMI Studio quick start guide

To download a free trial of OpenHMI Suite go to our web page dedicated to development kits on exorint.com:

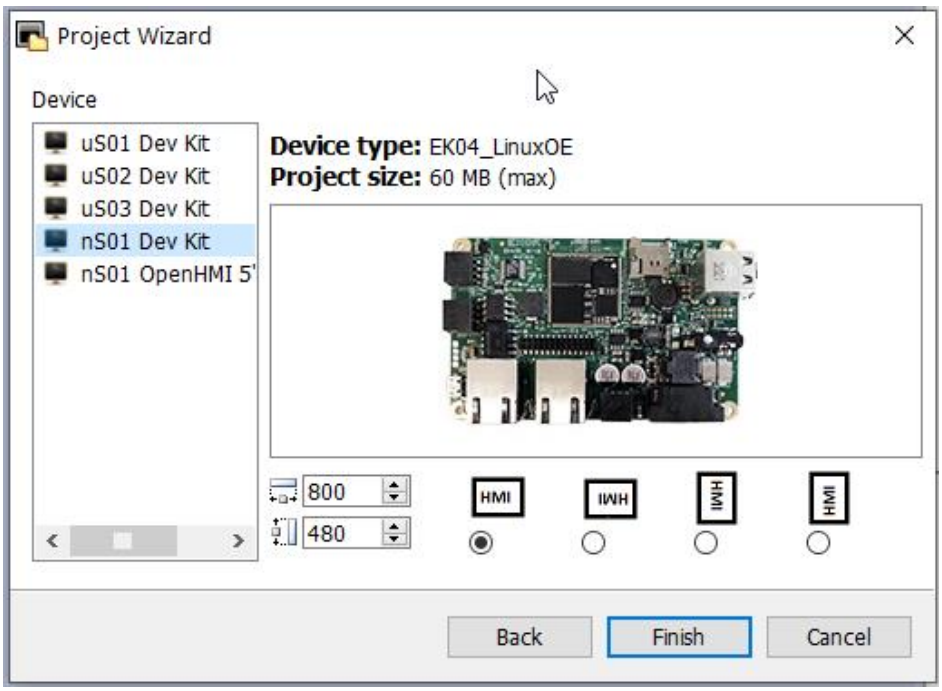
```
https://exorint.com/product-category/embedded/dev-kits/
```

Select the device you are working with then, from the *"Download"* section, download the latest version of OpenHMI Suite. After installation, start OpenHMI Studio and create a new project from *"File"* → *"New.."*

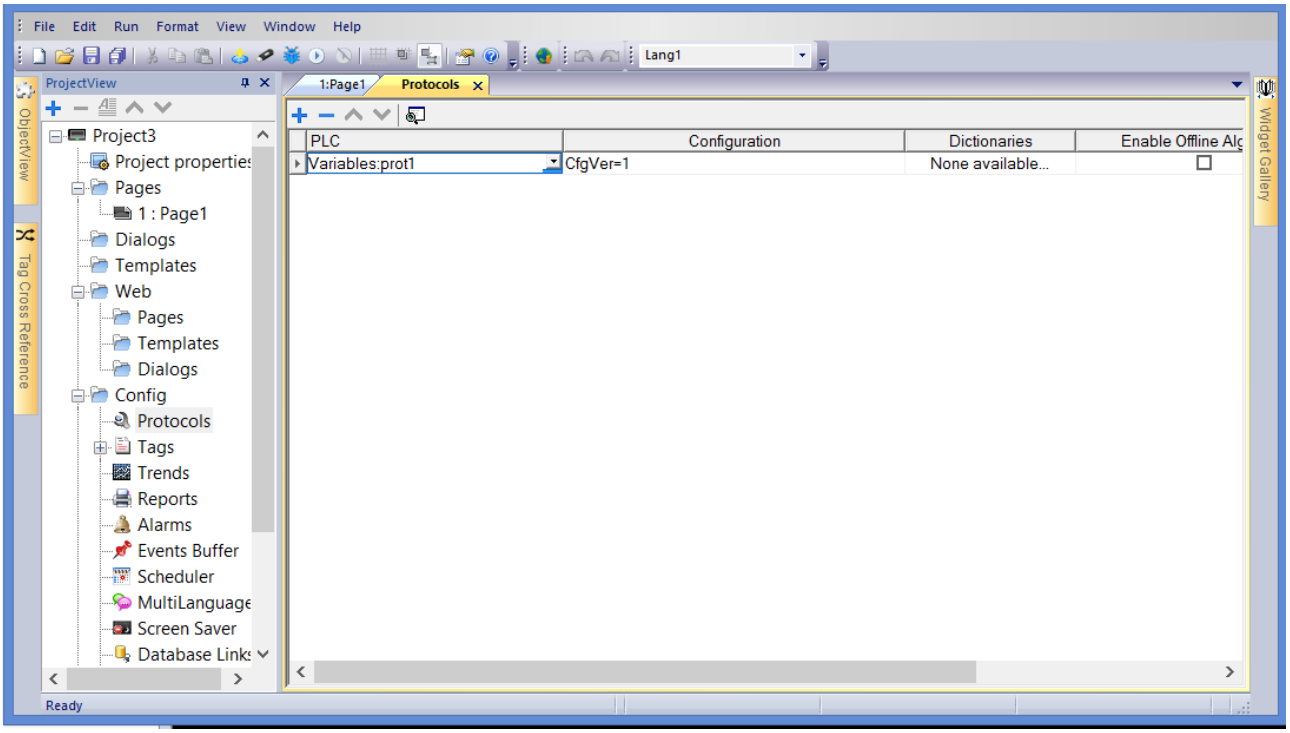


Enter a project name, select a location folder and click on *"Next"*. Select now the correct target corresponding to the board (ns01devkit):

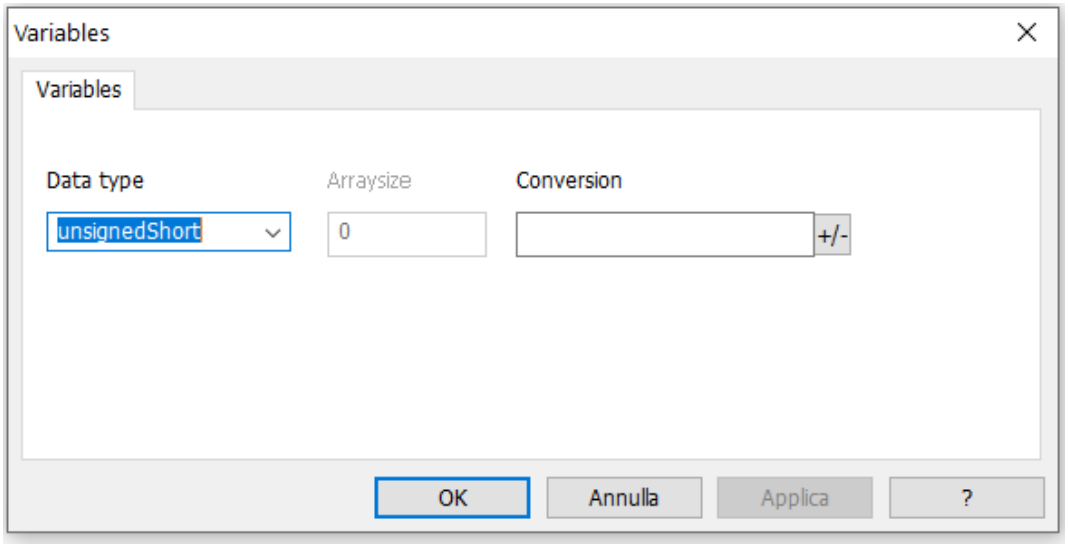
NOTE: please use only lower-case letters and numbers for the project name



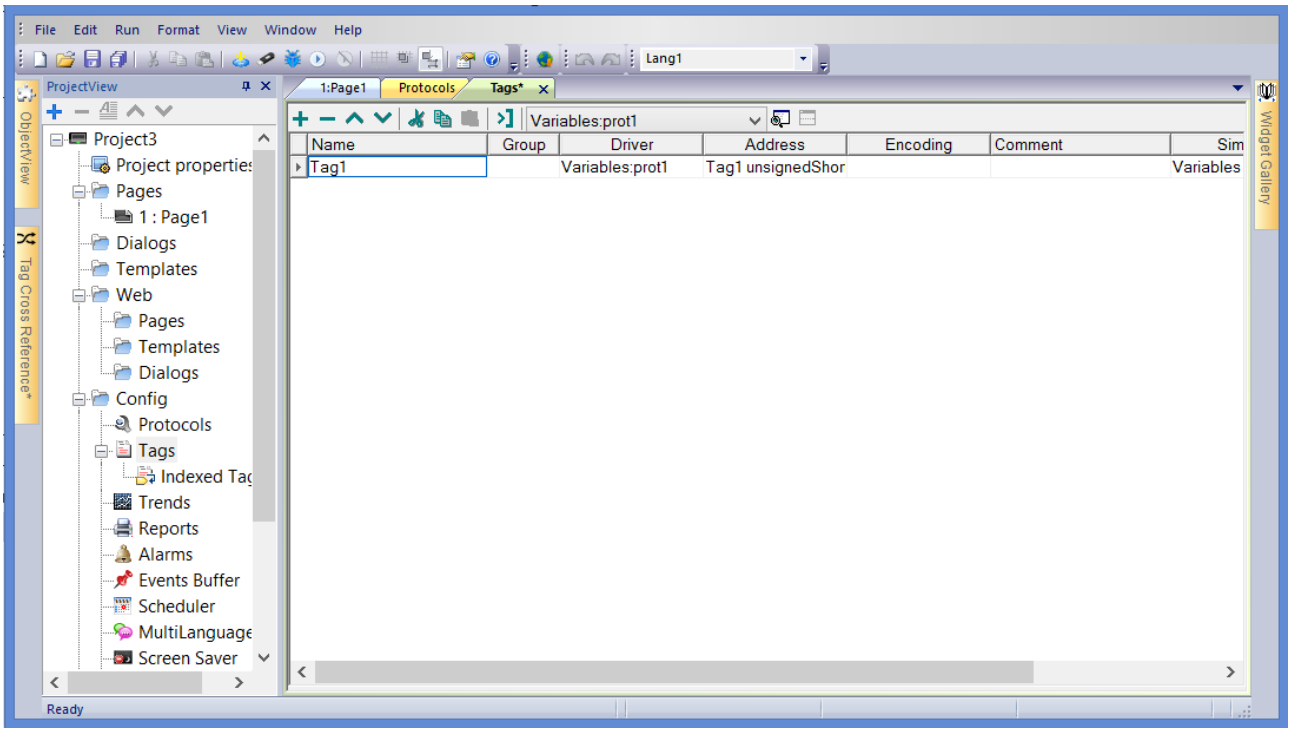
The goal is to create a project simply consisting of an increasing numerical counter.
From the *"ProjectView"* pane located on the left, click *"Protocols"*. Then click the *"+"* sign to add a new protocol and select *"Variables"* as shown in the figure below:



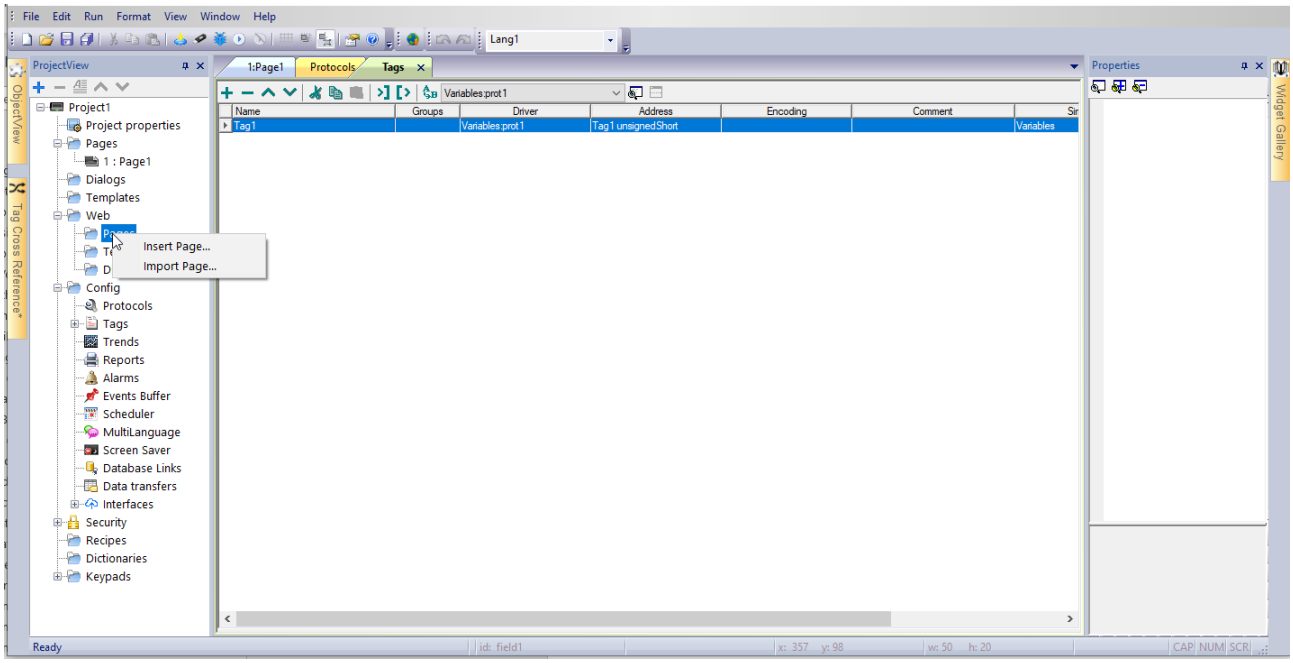
In the left panel, select *"Tags"*. Press the *"+"* sign and add an unsignedShort tag named "Tag1".



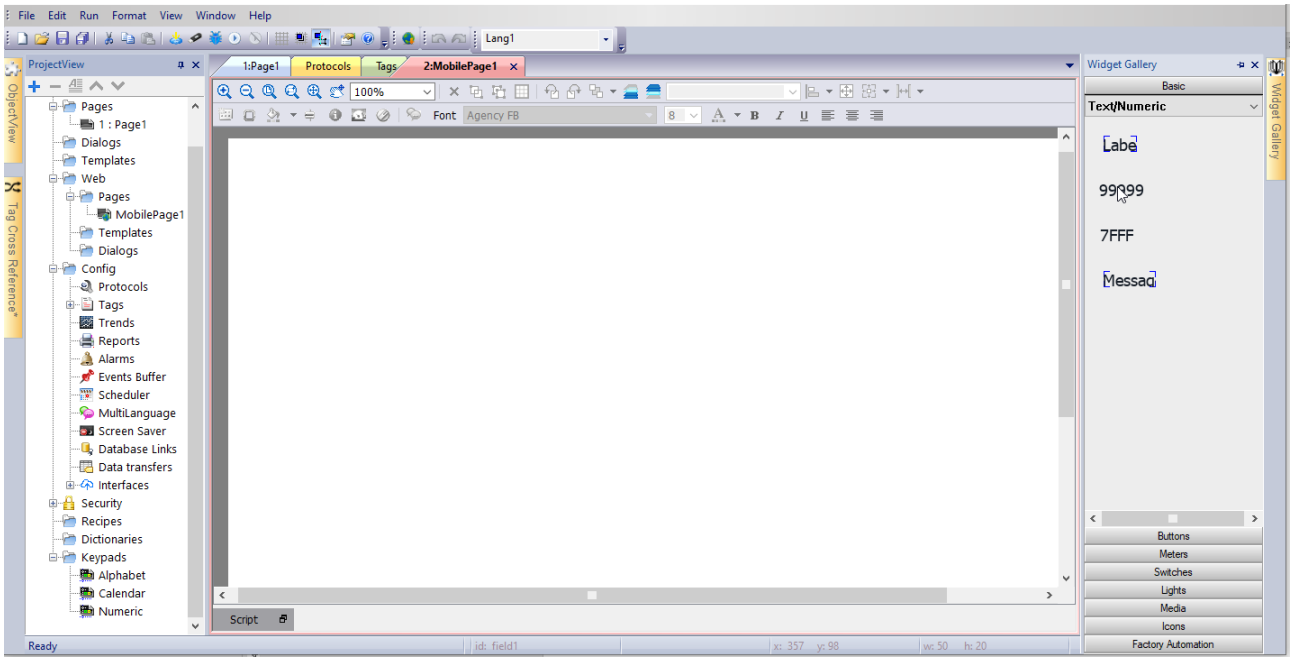
This variable will represent our counter

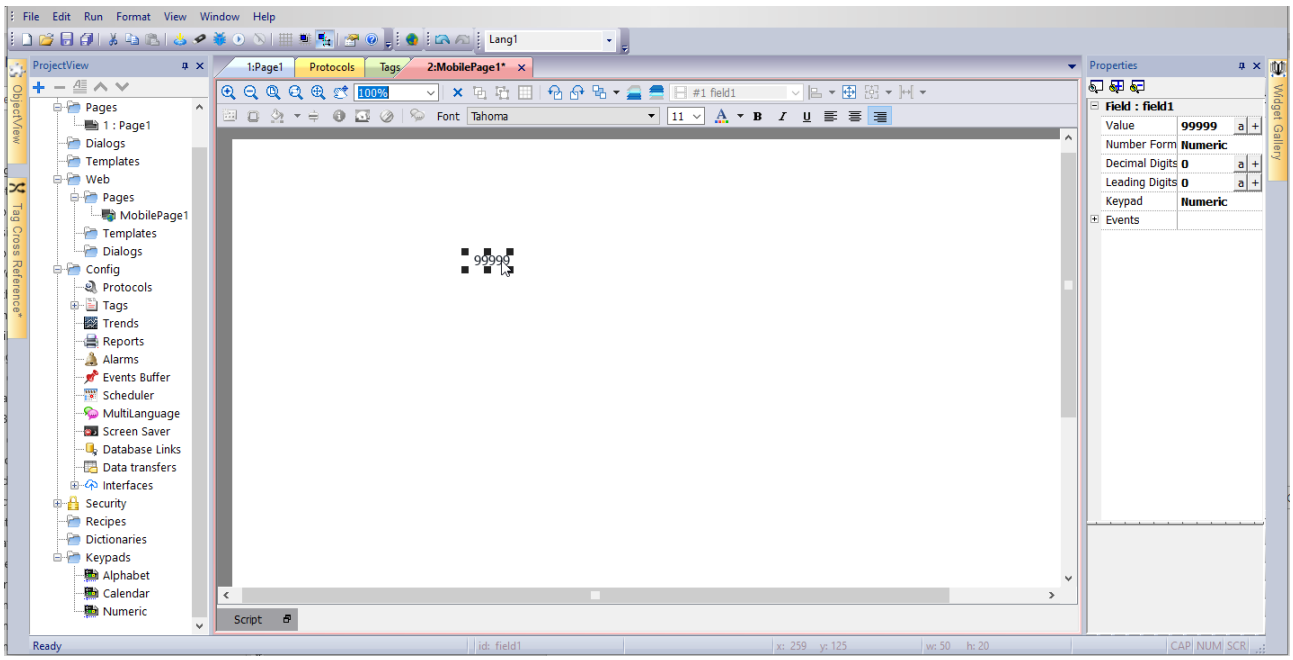


Since the Kairos development kit does not have a local display, we need to use web pages. Web pages show their content in any browser
In the left pane, right-click on *"Web"* → *"Pages"*, then click *"Insert page..."*

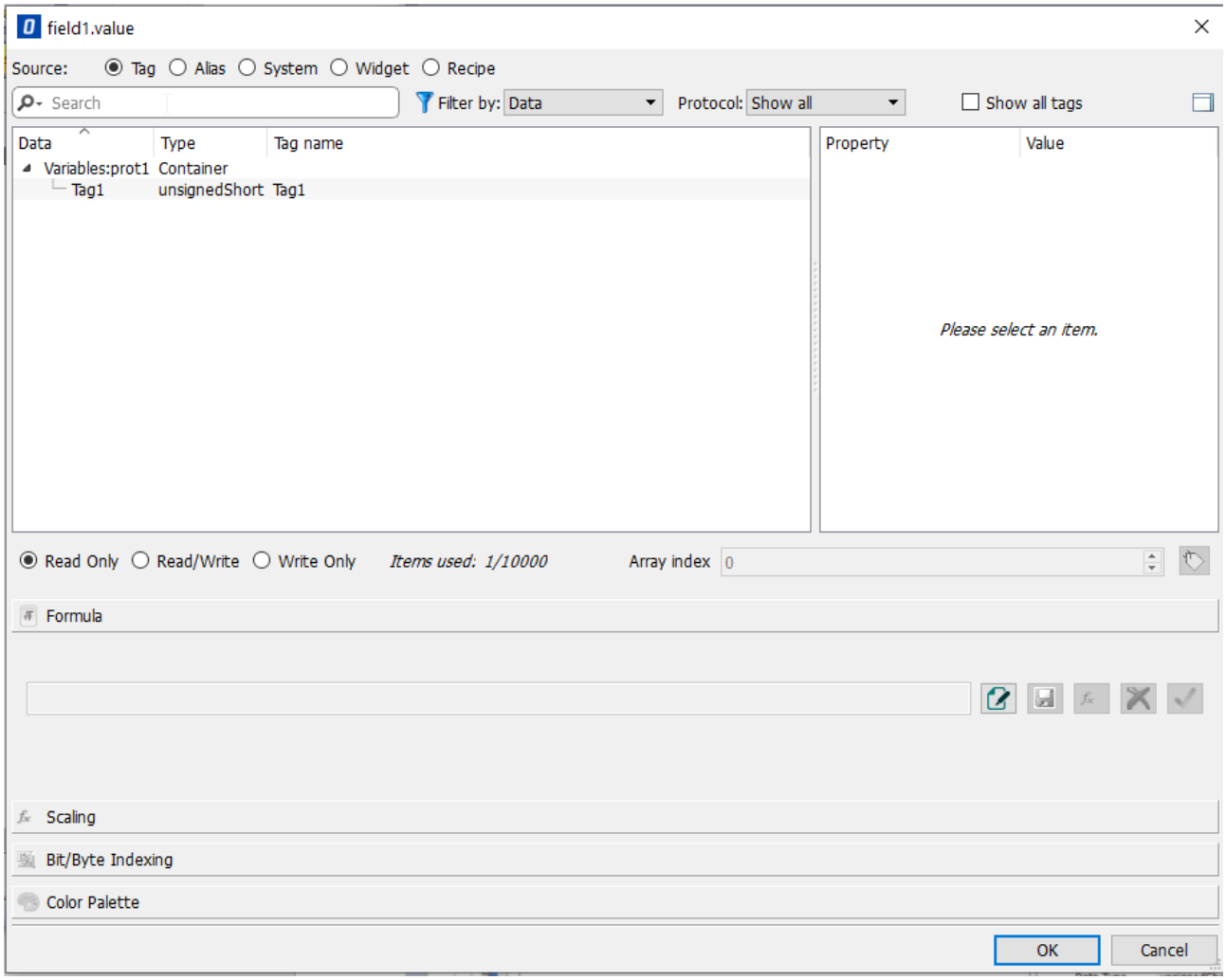


Add a numeric field widget to the project's page by dragging it from the Widget Gallery (Widget Gallery is located on the right):



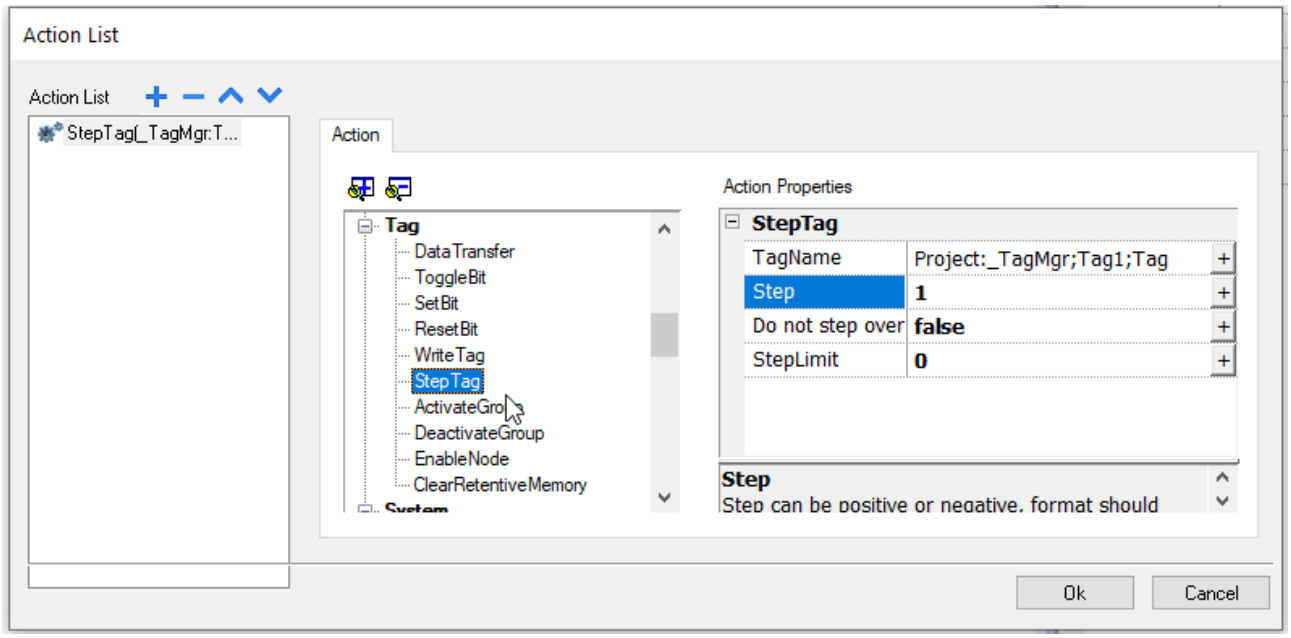


Double click on the numeric filed and select *"Tag1"* to bind the widget to the tag's value:

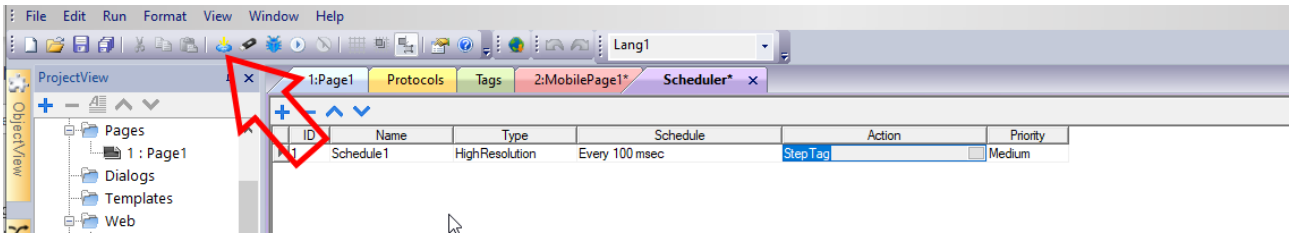


In the left pane, double click *"Scheduler"*. Add a new scheduler by clicking the *"+"* sign in the toolbar. In the *"Type"* column, select *"HighResolution"*. In the "Action" column, click the button with the ellipsis. This will open the *"Action List"* form. Here,

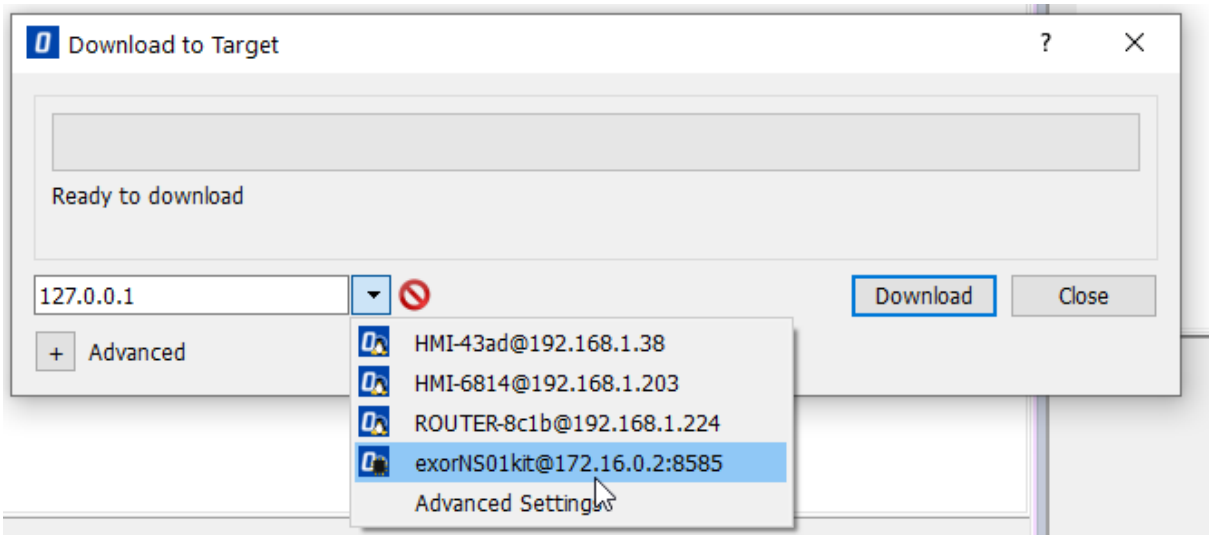
1. Browse the actions tree and select *"Step Tag"*
2. Click the "+" sign close to *"TagName"* and select the tag *"Tag1"*
3. Enter "1" in the "*Step*" field
4. Click *"OK"* to confirm changes



The project is now ready and can be downloaded to the Kairos development board.
Make sure the Kairos development kit is powered on, properly connected to your network and that OpenHMI runtime is running.
Click the *"Download to Target"* icon in the toolbar (or press the shortcut Ctrl+D).



Select the target from the drop-down list and click *"Download"* to deploy the project:



Open your favorite web browser and enter the IP address and port you selected when you downloaded the project (in the above screenshot 172.16.0.2:8585).
You will be asked to provide login credentials: default values are

Username: admin

Password: admin

You should now see a page with the incrementing counter