Setting up your PC as a CCGX development environment

This page explains setting up a cross compiling environment for the CCGX. Cross compiling means that the software is built/compiled on a different system (your computer) than the one on which it is executed (your CCGX).

Notes:

- rebuilding the whole rootfs and image from scratch is something entirely different, explained here: https://github.com/victronenergy/venus
- this page assumes your host pc / virtual runs linux
- it is often faster to do most development and debugging of software on your pc first. So no cross-compiling required at first. See here.
- when looking for ordinary packages, such as git, gdb, or something else, as opposed to your own development, make sure to have a look at all pre-compiled and available packages too. Login to the ccgx and run this command to see available packages:

  ```
  opkg list
  ```

- another alternative: compile on the ccgx itself. Use opkg to install git, make and gcc, then checkout whatever source you want to compile. Easier than cross compiling, but can be a bit slower :)

To cross compile, you need to setup an SDK, which contains the gcc compiler, as well as all header files and other setup of the CCGX.

BASICS - command line cross compile

First the basics: simply cross compiling a project, using the command line. This example has been made on Ubuntu.

Then replace dash with bash:

```
sudo dpkg-reconfigure dash   (and choose NO)
```

And go to Yocto documentation and execute the long apt-get install found under Ubuntu

Install the SDK

The SDK includes the gcc compiler for the arm and correct libraries.

Download the latest sdk (replace the 1.28 with latest version number):

```
wget
http://updates.victronenergy.com/feeds/ccgx/images/venus-eglibc-i686-arm-too
```

```
lchain-qte-v1.28.sh
```

Install it (it will ask where you want to have it installed):

```
chmod +x ./venus-eglibc-i686-arm-toolchain-qte-v1.28.sh
sudo ./venus-eglibc-i686-arm-toolchain-qte-v1.28.sh
```

Make a symlink /opt/venus/current

```
sudo ln -s /opt/venus/v1.28 /opt/venus/current
```

Now to use this SDK, the following command is to be executed in the terminal where you also call make or start qtcreator. This has to be done every time, but you are free to automate it of course:

```
source /opt/venus/current/environment-setup-armv7a-vfp-neon-ve-linux-gnueabi
```

Cross compile your first project

Create a file helloworld.c with the following content:

```
#include <stdio.h>

int main()
{
    puts("hello world");
    return 0;
}
```

The following is needed to compile above for a ccgx

```
# first setup the environment:
. /opt/venus/current/environment-setup-armv7a-vfp-neon-ve-linux-gnueabi

$CC helloworld.c -o helloworld
```

The resulting binary is now called helloworld. This is an ARM executable, this can be checked with

```
file helloworld
```

This should report something like: helloworld: ELF 32-bit LSB executable, ARM, EABI5 version 1 (SYSV), dynamically linked (uses shared libs), for GNU/Linux 2.6.16, not stripped

Cross compiling QT projects

QT projects rely on the qmake engine. Compiling a projects works like this:

```
# first setup the environment:
. /opt/venus/current/environment-setup-armv7a-vfp-neon-ve-linux-gnueabi
# change directory to the location of the qmake file (.pro extension)
cd <path to project file>
# Run qmake to create a makefile. Use the qmake supplied with the SDK.
/opt/venus/current/sysroots/i686-ve-linux/usr/bin/qmake <project file>.pro
# Build the project
make
```

LUXURY - now that you have done the basics, go for the IDE

Now that you have successfully compiled a project, it is time for the full thing!

Install xubuntu (13.10 or newer, you need a recent one in order to use the new QT Creator)

Install QT Creator:

```
sudo apt-get install qtcreator (or take latest from qt website)
```

Install git

```
sudo apt-get install git (same, you can take latest one also)
```

Install svn

```
sudo apt-get install subversion
```

Setup tools and compile the GUI project with QT Creator

For example the CCGX GUI project (note that this is not an open source project (yet)!):

```
git clone --recursive https://git.victronenergy.com/ccgx/gui.git
~/dev/ccgx/gui
```

To see on which branch you are:

```
git status
```

Open a new terminal and set the environment:

```
??
```

To start QTCreator, start a new terminal, and execute the command

```
source /opt/venus/current/environment-setup-armv7a-vfp-neon-ve-linux-gnueabi
```

And then start qtcreater:

```
qtcreator
```

Configuring QT Creator manually

Goto options→Build & Run, and add a compiler, name it CCGX, and point it to

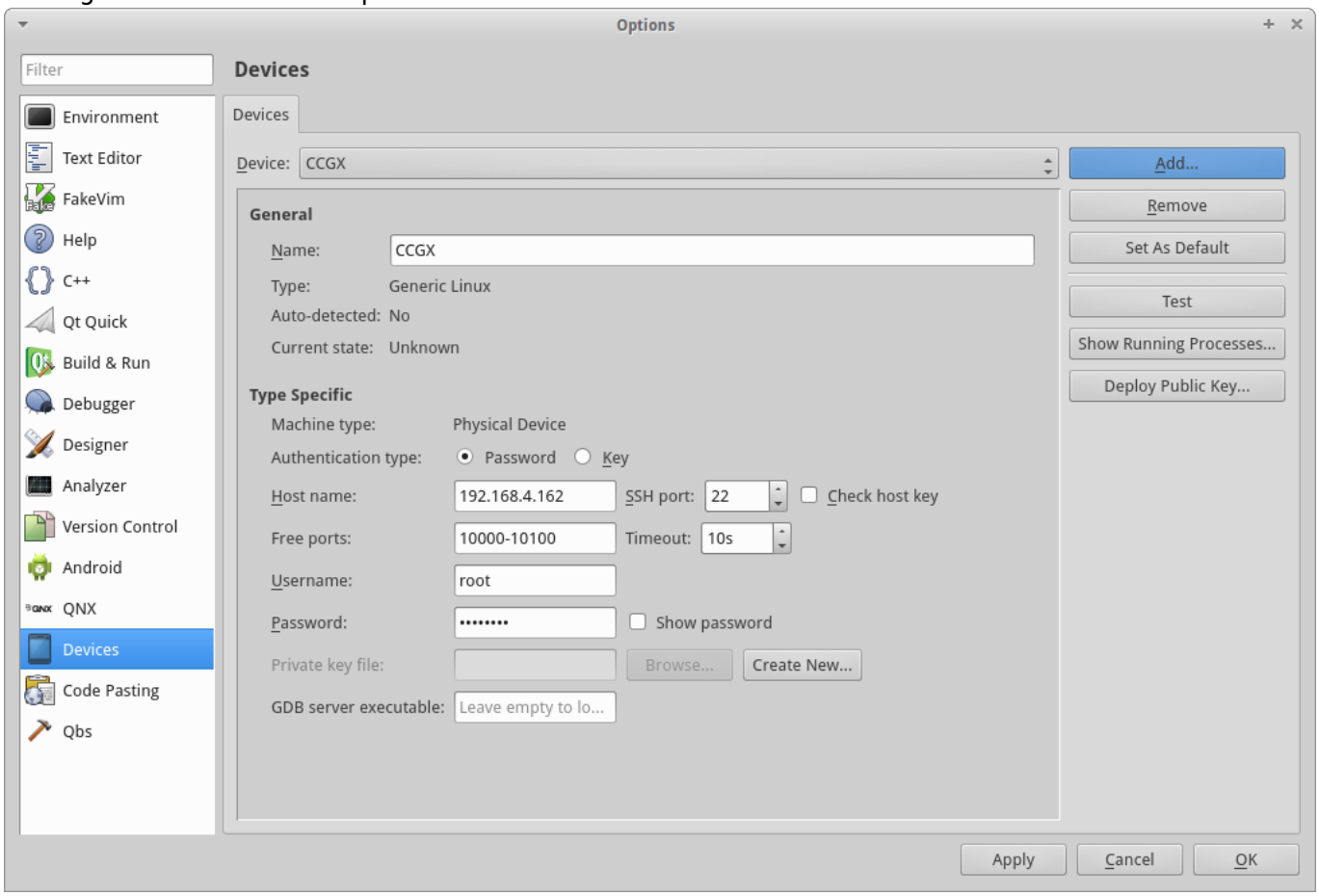
```
/opt/venus/current/sysroots/i686-ve-linux/usr/bin/armv7a-vfp-neon-ve-linux-gnueabi/arm-ve-linux-gnueabi-gcc
```

Then add qt version: QT Versions, add, point it to

```
/opt/venus/current/sysroots/i686-ve-linux/usr/bin/qmake
```

Next add a new kit which uses the compiler and the QT version you just created.

Add device, see screenshot. You can set your own root password on the CCGX in Settings→General→Set root password. More info here.



Now you are ready to start compiling. Open a QT project file (.pro extension) and chose the CCGX kit, and chose Build→Build Project. If you get an error message 'c: Command not found', you probably forgot to run the environment script before starting QT Creator.

After a successful build deploy the executable to the CCGX (Build→Deploy project). Note that it is not possible to overwrite an executable that is currently running.

Crosscompile run:

Executable on device: /opt/color-control/gui

Crosscompile debug:
install gdbserver on the ccgx:

```
opkg install gdbserver
```

More info

Mount CCGX file system locally

To save yourself some time copying files back and forth between your PC and your CCGX, for example while editing Python code, use sshfs to mount the CCGX drive to your local machine:

```
  mkdir ~/rem
  sshfs root@ccgx:/opt/color-control ~/rem
```

Use fusermount -u PATH to unmount it again. or just reboot your machine.

Mounting a ubifs image

To get the right tools in ubuntu, install mtd-tuils:

```
sudo apt-get install mtd-utils
```

Info can be found on this page: http://pjankows.blogspot.nl/2012/01/how-to-mount-ubi-image.html

However, don't use mtdram, but use nandsim.

```
sudo modprobe ubi
sudo modprobe nandsim first_id_byte=0x20 second_id_byte=0xac
third_id_byte=0x00 fourth_id_byte=0x15
sudo flash_erase /dev/mtd0 0 0
sudo ubiformat /dev/mtd0 -f ubi.img
sudo ubiattach -p /dev/mtd0
sudo mount -t ubifs /dev/ubi0_0 /mnt/ubifs/
```

Developing and running on your PC instead of immediately on the target

Developing, running, debugging a module on your (Linux) PC is often much faster than first having it uploaded to the CCGX everytime you want to run it. And the good news is that it is perfectly possible.

Most of our modules will run perfectly on a pc: localsettings, dbus_gps, dbus_modbustcp, dbus_fronius, etcetera. Even the gui runs on a pc, but that is a bit more difficult since it needs some changes which we made to the qt libraries.

Most -perhaps even all- of our D-Bus implementations (C, Cpp, Python, etc) automatically choose the Session D-Bus instead of the System D-Bus which is used on the ccgx. This is done at either compile- or run-time. To see what is going on on the D-Bus, use the DBusCli command-line tool. Make sure to omit the -y commandline parameter. Tips and tricks for command line D-Bus access here.

For some modules you'll need to run localsettings on your pc. Github repo including explanation of what it is is here. Look for localsettings on this page for instructions on setting it up. Note that it shouldn't be necessary to change the dbus config files! Since it will be using the (open) session dbus, and not the system dbus which is usually locked down.

DISQUS

~~DISQUS~~