WCH on Linux

Introduction

Developing for any micro-controller requires at least the following tools:

- A toolchain, that is a compiler, an assembler, a link editor, and miscellaneous related utilities. MounRiver Studio, WCH's IDE, integrates a complete toolchain, which can also be downloaded separately from the same web site if you want to automate your builds and tests (continuous integration).
- A tool to program (or "flash") the micro-controller, that is a utility that can transfer the binary code of your application (your firmware) in the flash memory of the micro-controller. WCH provides WCHISPTool - a Windows application - for this purpose, but it doesn't work well under Wine. WCH also provides WCHISPTool_CMD, a multi-platform command-line tool, which is the one you'll have to use under Linux.
- A tool to interface the micro-controller with your IDE's debugger. WCH provides a modified version of OpenOCD designed to support the WCH-Link / WCH-LinkE / etc. family of adapters. This version of OpenOCD comes with MounRiver Studio's toolchain.

This document covers the installation and use of those tools.

Preliminary task

Correct operation of the flashing and debugging tools requires that your user can control the WCH-LinkE adapter, as well as your development board when it is connected to your computer through its USB device port. For this purpose, we assume that group `plugdev` exists on your machine and that your user is a member of this group.

To determine whether group `plugdev` exists, type:

```
grep plugdev /etc/group
```

If nothing is displayed, the group doesn't exist and you need to create it with the following command:

```
sudo groupadd plugdev
```

If the group exists, you'll see a line like the following:

```
plugdev:x:46:vincent
```

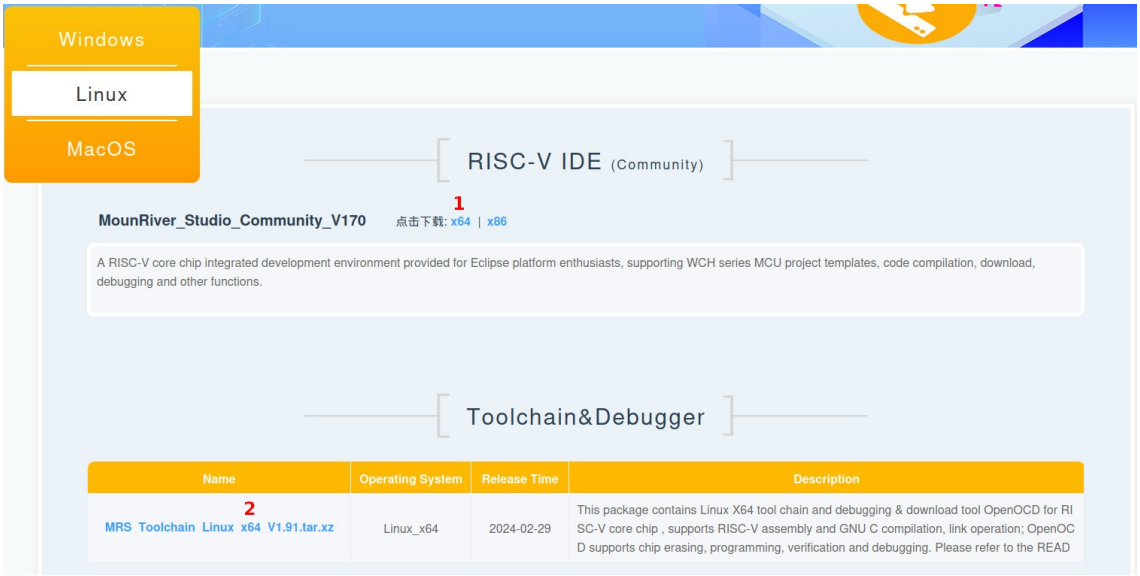
If your user name is not listed after the colon following the group's numeric identifier, you'll need to add yourself to the plugdev group with the following command:

```
sudo usermod -a -G plugdev your-user-name
```

You'll need to log off and back in for the change to take effect.

IDE and toolchain

To download the MounRiver Studio IDE (MRS for short) or the independent toolchain, visit http://www.mounriver.com/download and click link 1 in the screenshot below for the full IDE, or 2 for the independent toolchain. We will only cover the case of the IDE in this document.



Unpack the archive in the directory of your choice, for instance `/opt`. To facilitate future IDE updates, create a symbolic link `/opt/mrs` pointing at the directory extracted from the archive. For example:

```
cd /opt
sudo tar xJf MounRiver_Studio_Community_Linux_x64_V170.tar.xz
sudo rm MounRiver_Studio_Community_Linux_x64_V170.tar.xz
sudo ln -s MounRiver_Studio_Community_Linux_x64_V170 mrs
```

For the WCH-LinkE adapter and development boards to be correctly detected, udev rule files must be copied in `/etc/udev/rules.d`. Only `50-wch.rules` matters for this, but if you also want to use OpenOCD with non-WCH micro-controllers, you will want to copy `60-openocd.rules` too. Here's the procedure:

```
cd /opt/mrs/beforeinstall
# Create destination directory if it is missing
sudo mkdir -p /etc/udev/rules.d
# Fix file permissions
chmod a-x *.rules
# Copy rules
sudo cp *.rules /etc/udev/rules.d/
# Reload udev rules
sudo udevadm control --reload
```

You also need to create an application launcher to be able to start the IDE:

```
# Create destination directory if it is missing
mkdir -p ~/.local/share/applications
# Create the launcher
cat <<EOF > ~/.local/share/applications/mounriver-studio.desktop
[Desktop Entry]
Name=MounRiver Studio Community IDE
Comment=MounRiver Studio Community IDE
Exec=sh -c "LD_LIBRARY_PATH=/opt/mrs/beforeinstall
/opt/mrs/MRS_Community/MounRiver\ Studio_Community"
Icon=mounriver-studio
Terminal=false
Type=Application
StartupNotify=false
Categories=Development;IDE;
EOF
```

Please note the the `LD_LIBRARY_PATH=/opt/mrs/beforeinstall` in the launcher makes it unnecessary to copy the `.so` files from `/opt/mrs/beforeinstall` into `/usr/lib`. This will avoid the risk of breaking your system in case of conflicts, and will also ensure MRS uses the exact libraries it expects rather than the system's.

Finally, download the following icon and copy it in `/usr/share/pixmaps` (with `sudo`): https://riscv-mcu.defert.com/mcu-wch-linux/mounriver-studio.png

Your IDE is now ready for use. If you want to use its toolchain directly to build your projects from the command line (e.g. for continuous integration), you can create a series of symbolic links to expose those tools more conveniently:

```
sudo mkdir -p /opt/toolchain
cd /opt/toolchain
# GCC 9.3 modified by WCH (used by default by MRS)
sudo ln -s ../mrs/MRS_Community/toolchain/RISC-V\ Embedded\ GCC riscv-none-embed-wch
# GCC 12 modified by WCH (if you prefer to use it)
sudo ln -s ../mrs/MRS_Community/toolchain/RISC-V\ Embedded\ GCC\ 12 riscv-none-elf-wch
# OpenOCD 0.11 modified by WCH
sudo ln -s ../mrs/MRS_Community/toolchain/OpenOCD openocd-wch
```

Notes on USB vendor and product ID

When used in RISC-V mode (aka. WCH-LinkRV), your WCH-LinkE adapter shows up as 1a86:8010.

When used in ARM-WINUSB mode (aka. WCH-DAPLink), it shows up as 1a86:8012.

Finally, when used in ISP mode, during its firmware update, it shows up as 4348:55e0.

This is also the case with any WCH MCU having a USB device port when used in BOOT mode (e.g. CH32V203, CH32V307, CH582, etc), which is the mode used by WCHISPTool_CMD and the ch37x driver.

As a consequence, for proper operation in all modes, the following udev rules must be present:

```
ATTR{idVendor}=="1a86", ATTR{idProduct}=="8010", MODE="0660", GROUP="plugdev"
ATTR{idVendor}=="1a86", ATTR{idProduct}=="8012", MODE="0660", GROUP="plugdev"
ATTR{idVendor}=="4348", ATTR{idProduct}=="55e0", MODE="0660", GROUP="plugdev"
```

This is normally the case if you use the 50-wch.rules file which comes with MRS.

Note that if you don't have the last line (4348:55e0), your WCH-LinkE will be bricked when its firmware will be updated. This is recoverable, MRS includes WCH-Link firmware images in the MRS_Community/update/Firmware_Link subdirectory and instructions are provided in WCH-LinkUserManual.pdf, but making sure the udev rule is present will avoid you the trouble.

WCHISPTool_CMD

Installation of the CH37x driver

WCH micro-controllers with a USB interface also feature the Code-Protect function, which prevents the flashing and debugging of the micro-controller, and which is factory-activated. This is of course a problem for development, which is why WCH deactivates it on the development boards they sell. Other development board manufacturers don't deactivate it, though, so you will very likely run into this issue. The same applies to your custom boards, of course. The CH32V00x MCU series having no USB interface, they aren't affected by Code-Protect.

Code-Protect being active means that you won't be able to use the WCH-LinkE adapter to communicate with the micro-controller. To deactivate Code-Protect, you will need to connect your development board to your computer using the board's USB device port while holding the BOOT button pressed, or with the BOOT jumper in place if there's no BOOT button. You will then have a few seconds to deactivate Code-Protect using WCHISPTool_CMD, as we'll see later.

However, the CH37x driver must be installed on your system for this procedure to succeed. The driver's sources can be downloaded from WCH's web site. I'll suppose you have saved this file under the name CH37X_LINUX.ZIP and that you've opened a terminal window in the directory where this file is located. Here is what you need to do to build and install the CH37x driver:

```
# Install the packages needed to compile the driver.
sudo apt -y install dkms build-essential linux-headers-generic
# Unpack the driver's sources downloaded from here:
# https://www.wch-ic.com/downloads/CH37X_LINUX_ZIP.html
# and stored in the current directory.
unzip CH37X_LINUX.ZIP
# Create the DKMS configuration file.
# The chosen options will cause the driver to be rebuilt and reinstalled
# whenever the Linux kernel is updated.
cat <<EOF > CH37X_LINUX/driver/dkms.conf
PACKAGE_NAME=ch37x
PACKAGE_VERSION=1.0
AUTOINSTALL=y
BUILT_MODULE_NAME[0]=ch37x
DEST_MODULE_LOCATION[0]=/kernel/usb/misc
EOF
# Create the target directory for the driver's sources.
sudo mkdir -p /usr/src/ch37x-1.0
# Copy the driver's sources.
sudo cp CH37X_LINUX/driver/* /usr/src/ch37x-1.0/
# Delete temporary files.
rm -rf CH37X_LINUX CH37X_LINUX.ZIP
# Build and install the driver.
sudo dkms install ch37x/1.0
# Create destination directory if it is missing.
sudo mkdir -p /etc/modules-load.d
# Configure automatic loading of the driver upon boot.
sudo sh -c 'echo ch37x > /etc/modules-load.d/ch37x.conf'
# Create destination directory if it is missing.
sudo mkdir -p /etc/udev/rules.d
# Associate the driver to group plugdev so that it can be used
# by non-privileged users.
sudo sh -c 'echo "KERNEL==\"ch37x0\", OWNER=\"root\", GROUP=\"plugdev\",
MODE=\"660\"" \
    > /etc/udev/rules.d/70-ch37x.rules'
# Reload udev rules.
sudo sudo udevadm control --reload
```

Installation of WCHISPTool_CMD

You will need WCHISPTool_CMD to disable CODE-Protect - thus allowing the use of you non-WCH development board with MRS - or to flash your micro-controllers using the command line (e.g. to automate "hardware in the loop" tests).

To install WCHISPTool_CMD, download it from WCH's web site and save the archive to the directory of your choice, for instance `/opt/toolchain`, open a terminal window in the directory where this file is located and execute the following commands:

```
# Unpack the archive downloaded from here:
```

```
# https://www.wch-ic.com/downloads/WCHISPTool_CMD_ZIP.html
# and saved in the current directory.
sudo unzip WCHISPTool_CMD.ZIP
sudo rm WCHISPTool_CMD.ZIP
cd WCHISPTool_CMD
# Fix file permissions
sudo chmod -R a+rX Linux
sudo find Linux -type f -name WCHISPTool_CMD -exec chmod a+x {} ';'
# Move files of interest to us
sudo mv Linux/* .
# Delete unused files
sudo rm -rf Linux macOS Windows
```

You can now run WCHISPTool_CMD with the following command:

```
/opt/toolchain/WCHISPTool_CMD/bin/x64/WCHISPTool_CMD
```

Using WCHISPTool_CMD

WCHISPTool_CMD needs a `.ini` configuration file to define the micro-controller model used and its configuration options. These pieces of information correspond to those displayed in the user interface of the WCHISPTool Windows application.

In order to save you from struggling with Wine and WCHISPTool, I have generated an `.ini` file with the default configuration options for each supported micro-controller and made them available here: https://codeberg.org/20-100/Awesome_RISC-V/src/branch/master/WCH/config.ini.

WCHISPTool_CMD can use 2 types of communication interfaces, USB device port of the development board, or a serial port. Using the USB port is mandatory to disable Code-Protect, and optional to flash the micro-controller - which can also be flashed through the serial port (or using OpenOCD, or with MRS) if Code-Protect is disabled.

In order to use the USB port, the CH37x driver must be installed and loaded, and you must enable the USB bootloader of the micro-controller. For this purpose, development boards offer either a push button marked "BOOT", or a pin header marked "BOOT" immediately next to another pin header marked GND. While connecting your development board to your computer, you must either hold the BOOT button pressed, or the BOOT and GND pin headers shorted. You then have around 20 seconds to execute the WCHISPTool_CMD command. Then, the micro-controller will exit BOOT mode and start to execute its firmware.

In order to use the serial port, you must create a /dev/ttyISP0 symbolic link pointing at the desired serial port (e.g. /dev/ttyUSB0 or /dev/ttyACM0) - for some reason, WCHISPTool_CMD checks the name of the symbolic link and refuses to proceed if directly given the name of the correct device.

Here are the commands you will need - I suppose the /opt/toolchain/WCHISPTool_CMD/bin/x64 directory is in your PATH, or that you created an un alias for WCHISPTool_CMD:

- To flash the micro-controller through its USB device port:

  `WCHISPTool_CMD -p /dev/ch37x0 -c` *`CH32V208WBU6.ini`* `-o program -f` *`your-firmware.hex`*

- To verify the flashing through the USB device port:

  `WCHISPTool_CMD -p /dev/ch37x0 -c` *`CH32V208WBU6.ini`* `-o verify -f` *`your-firmware.hex`*

- To flash the micro-controller through its serial port:

```
WCHISPTool_CMD -p /dev/ttyISP0 -b 115200 -c CH32V208WBU6.ini -o program -f your-firmware.hex
```

- To verify the flashing through the serial port:

```
WCHISPTool_CMD -p /dev/ttyISP0 -b 115200 -c CH32V208WBU6.ini -o verify -f your-firmware.hex
```

Of course, parameters in italic must be replaced with your own values.

Using WCHISPTool to generate .ini files

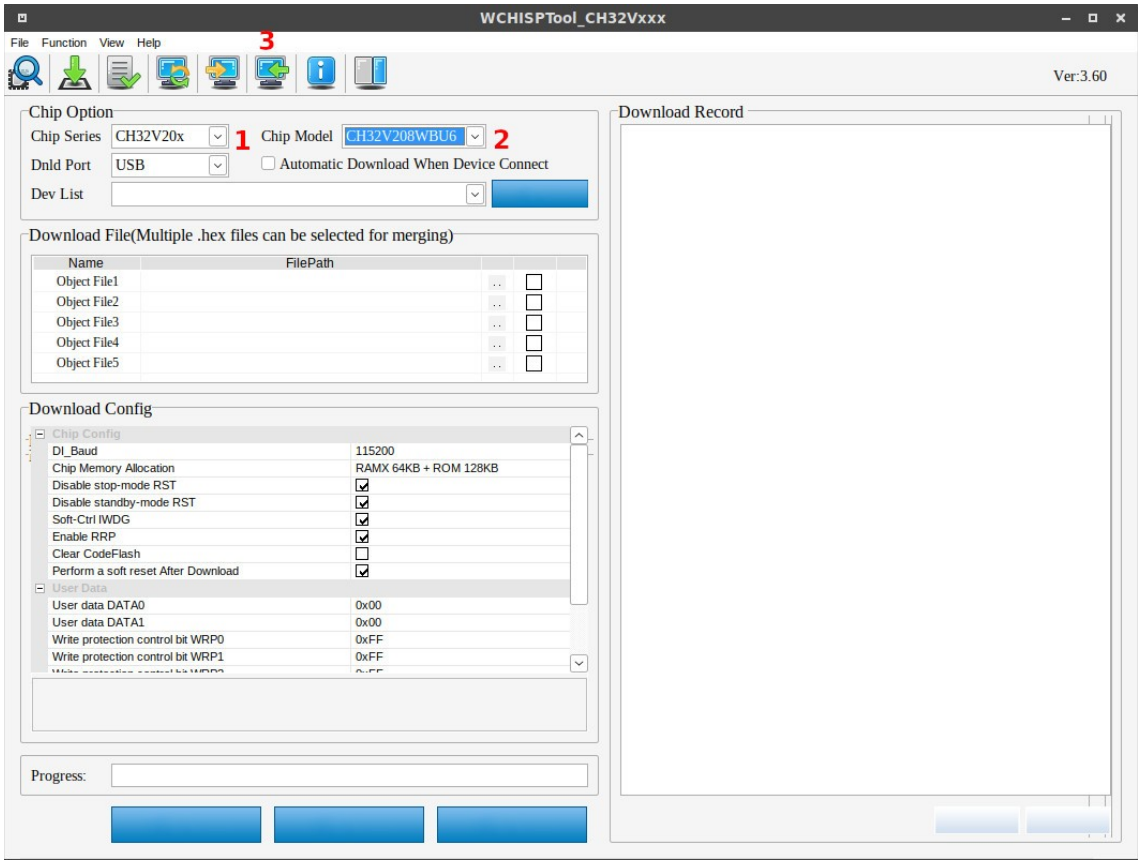
The WCHISPTool Windows application works poorly under Wine, but enough for you to generate `.ini` files if you want to. I suppose you have installed Wine and then WCHISPTool using the installer downloaded from WCH's web site.

WCHISPTool's main program, WchISpStudio, doesn't work at all under Wine, but its role is only to start sub-programs dedicated to each micro-controller family, which work a bit better. In order to start them manually, you must change to WCHISPTool's installation directory and start the dedicated program from here. For instance:

```
cd ~/.wine/drive_c/WCHISPTool
wine WCHISPTool_CH32Vxxx/WCHISPTool_CH32Vxxx.exe
```

Another possibility is to add WCHISPTool's installation directory to Wine's PATH and to directly start the dedicated program with Wine.

Once the dedicated program is started, select the MCU series, then the MCU model (e.g. CH32V20x then CH32V208WBU6), then click the configuration export icon and choose a name for the .ini file. This is summed up on the screenshot below:



Using OpenOCD

I suppose you have created the symbolic links in /opt/toolchain as described in the "IDE and toolchain" section. I also suppose you have disable Code-Protect on the target MCU, otherwise nothing will work. And of course, you need a WCH-LinkE adapter to connect to your board.

You can use OpenOCD to flash your micro-controller as follows:

```
/opt/toolchain/openocd-wch/bin/openocd \
    -f /opt/toolchain/openocd-wch/bin/wch-riscv.cfg \
    -c "program your-firmware.elf verify reset exit"
```

To use OpenOCD as a server to debug your application with GDB, the following command is enough:

```
/opt/toolchain/openocd-wch/bin/openocd \
    -f /opt/toolchain/openocd-wch/bin/wch-riscv.cfg
```

See OpenOCD's documentation to start your debugging session, and GDB's for more information. Using MRS for debugging will relieve you from struggling with this added complexity.