ARDUINO | prpl | OpenWrt Wireless Freedom

Quick Bio

Who am I ?

Senior Software Engineer, working for Arduino Srl since 2014

My main tasks :

- Linino distro (embedded linux distro) maintaining
- Plain Toolchains and cross-platform ones (mips, mingw-w64, arm etc.)
- Kernel Drivers backporting
- Shell scripting
- QA
- Support

LininoOS - What is LininoOS ?

Linino is the combination of two well known words around the world :

Linux + Arduino

LininoOS (rings a bell to you or not ?!?) is in an embedded linux distro relying its core on **OpenWRT** which is a well known embedded linux distro, unleashing to their limits the capabilities of the most popular SOHO market routers.

Its repositories provide the most common unix productivity packages and programming languages such as :

- Python
- Ruby
- Erlang
- PHP
- many more counting…

LininoOS - An ecosystem, why not ?

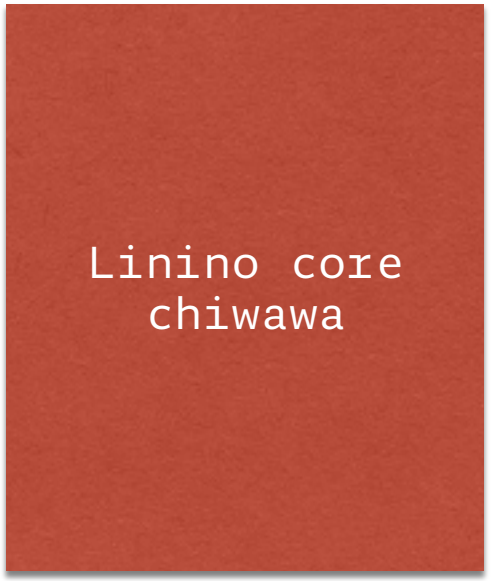
But….we'd like to think about it as more an **ecosystem** composed of:

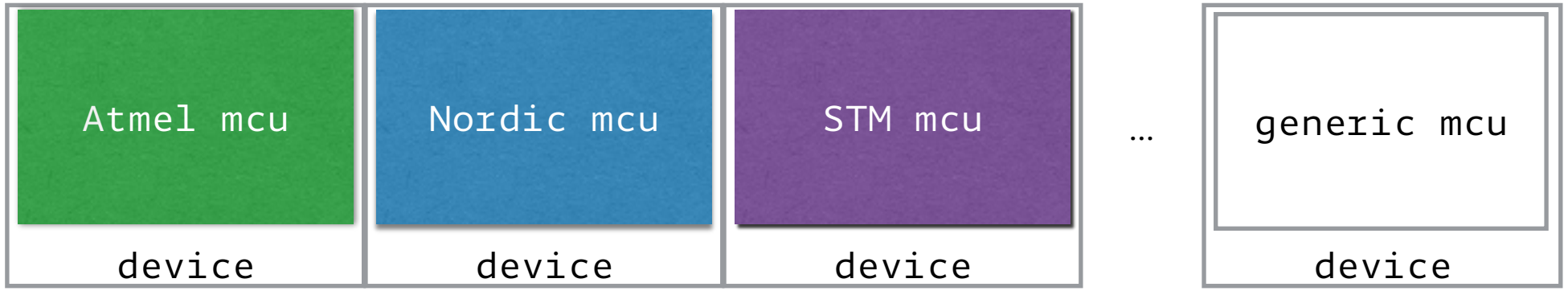
- a set of wi-fi enabled **open hardware** modules to easily make things;
- an **open software** layer to easily put intelligence on the things;
- an **embedded** development environment;
- an **open cloud** based set of services that make things easily interconnected;
- **people** of course! evangelizers, enthusiasts, makers, hackers, … the ecosystem is for them.

LininoOS - MCU agnostic approach

At the very same time, the Linux CPU and the on-board MCU are two separate entities but able to interact with each other as well in a seamless way.

And it does not depend on the given MCU !

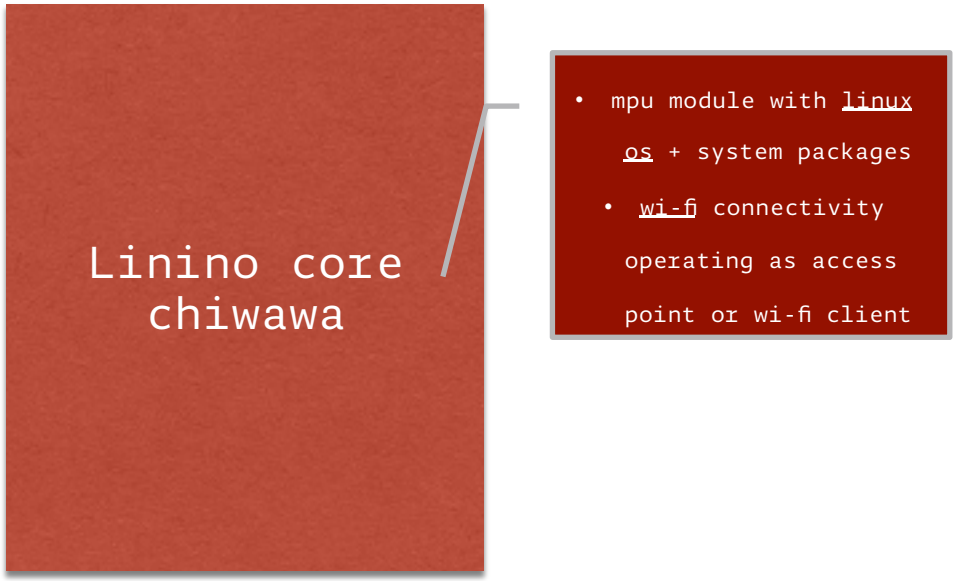


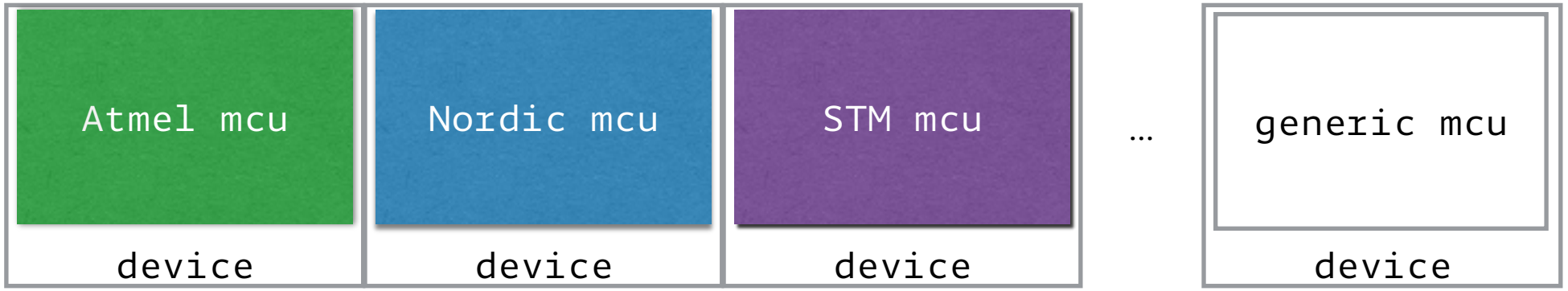


LininoOS - MCU agnostic approach

At the very same time, the Linux CPU and the on-board MCU are two separate entities but able to interact with each other as well in a seamless way.

And it does not depend on the given MCU !

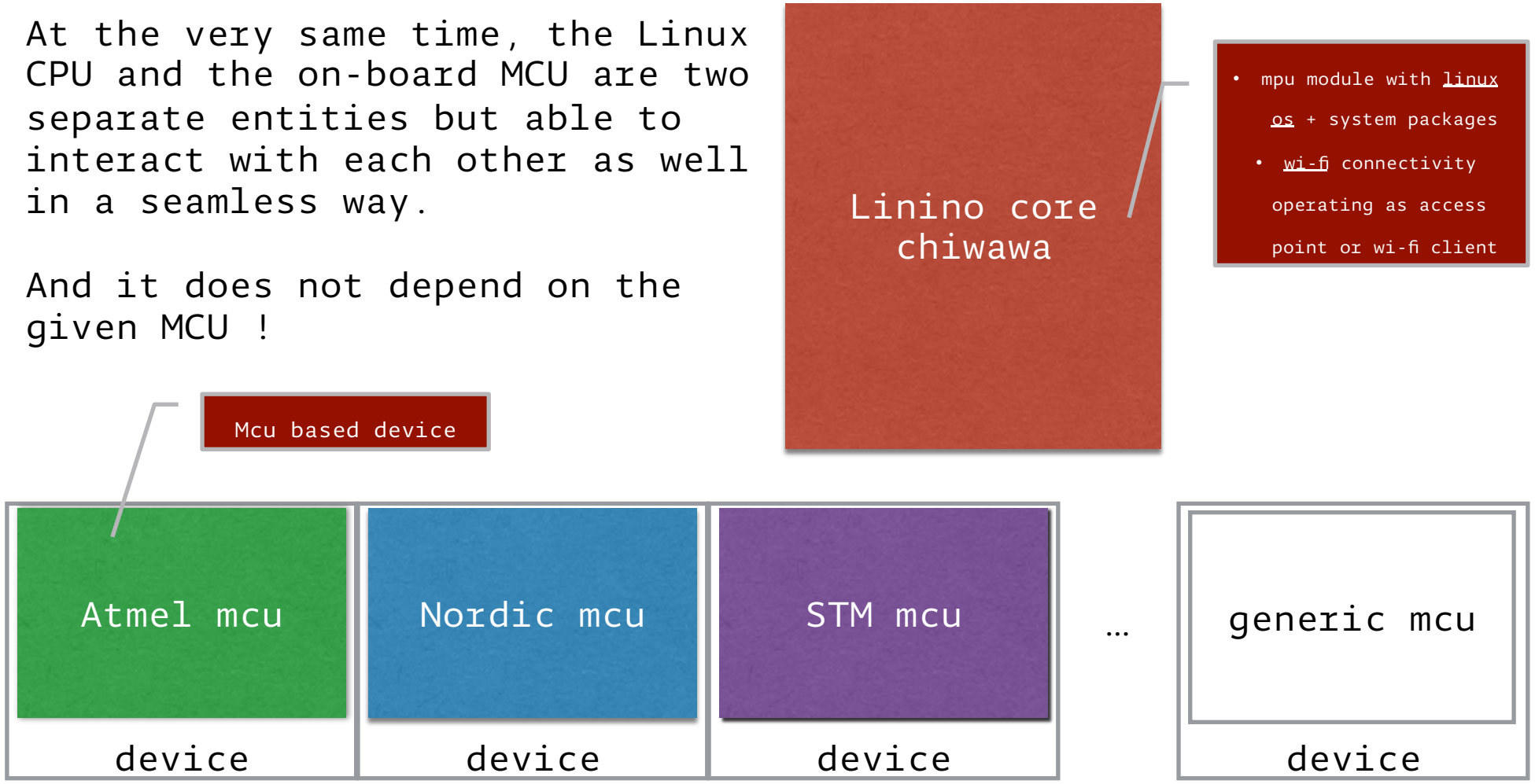




LininoOS - MCU agnostic approach

At the very same time, the Linux CPU and the on-board MCU are two separate entities but able to interact with each other as well in a seamless way.

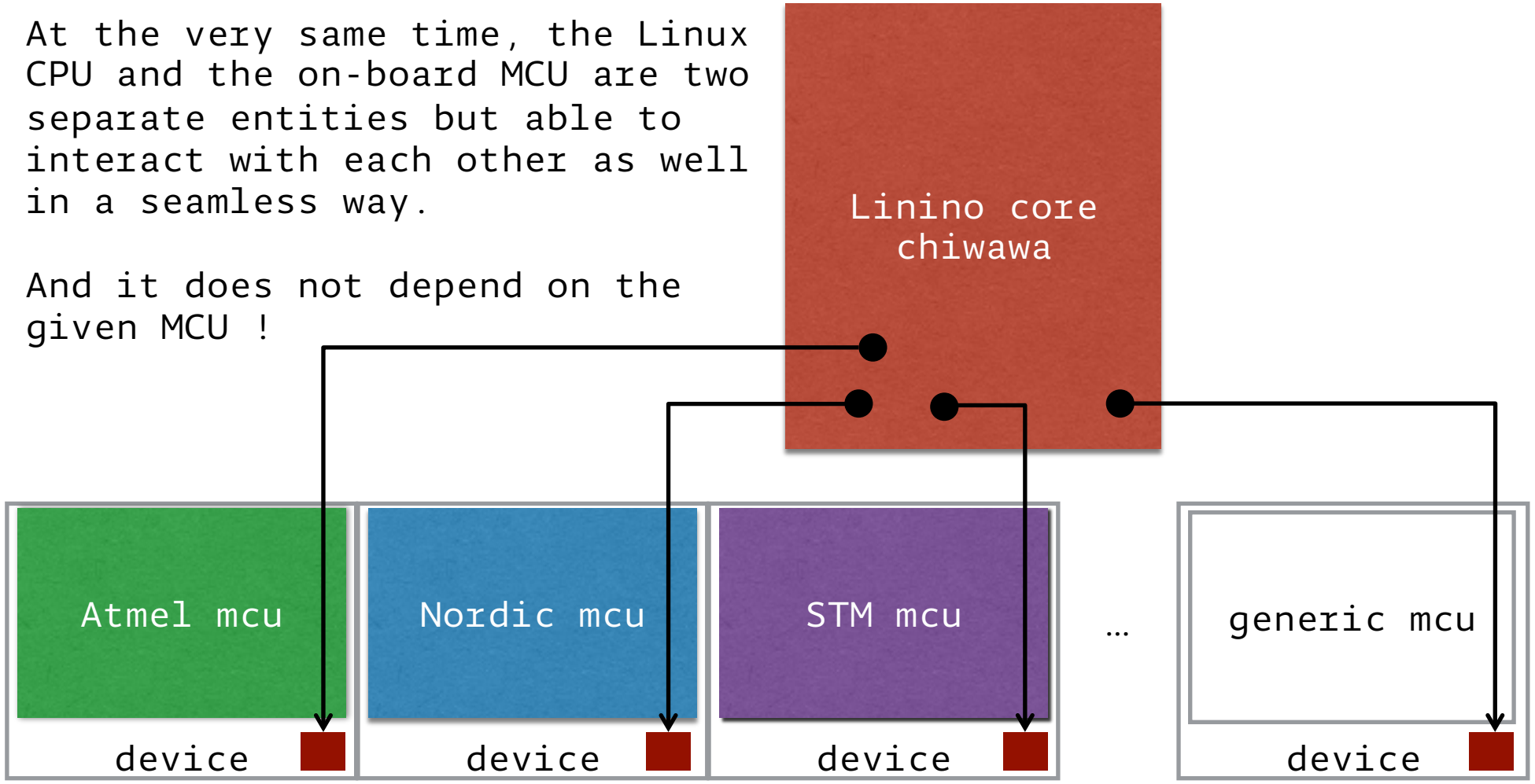
And it does not depend on the given MCU !



LininoOS - MCU agnostic approach

At the very same time, the Linux CPU and the on-board MCU are two separate entities but able to interact with each other as well in a seamless way.

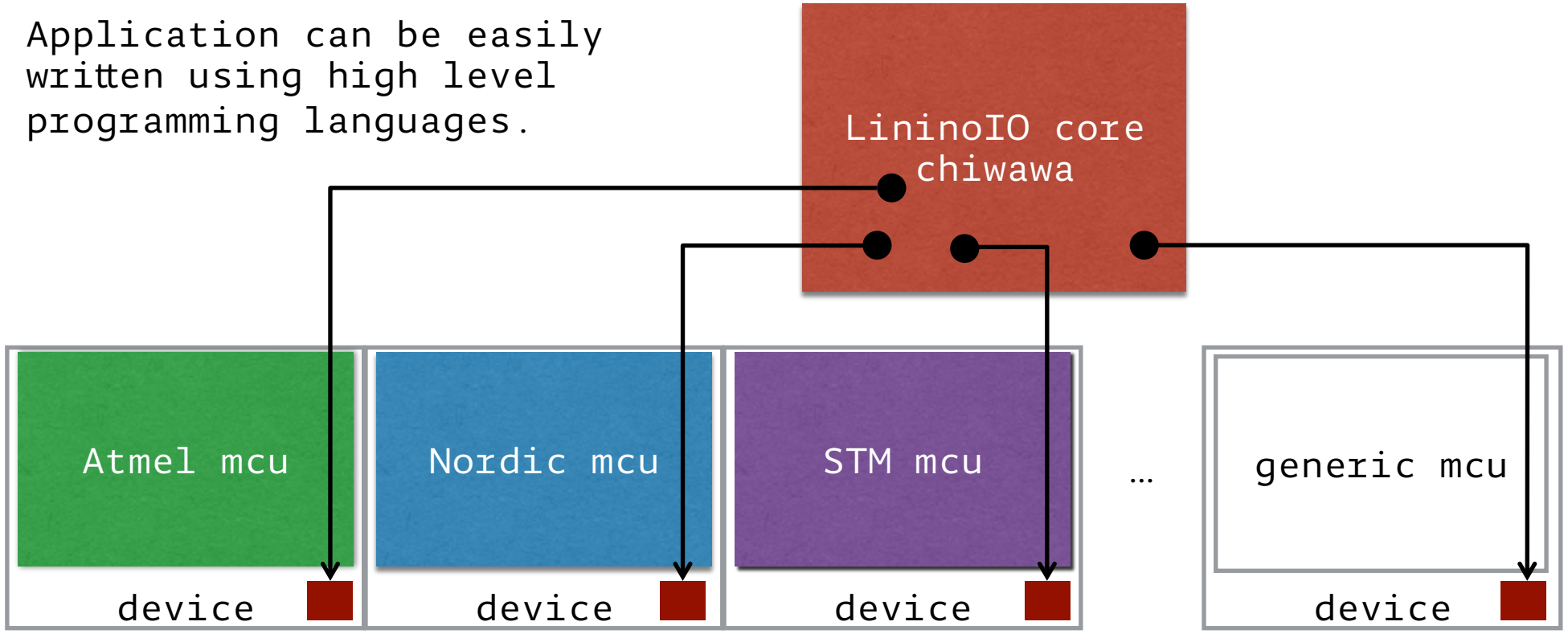
And it does not depend on the given MCU !



LininoOS – MCUIO approach

LininoIO is a software framework able to expose microcontroller features (such as GPIO, Analog Converters, PWM, I2C, SPI) inside the microprocessor environment in a typical UNIX fashion.

Application can be easily written using high level programming languages.



Nordic IoT 6lowpan daemon + Tian

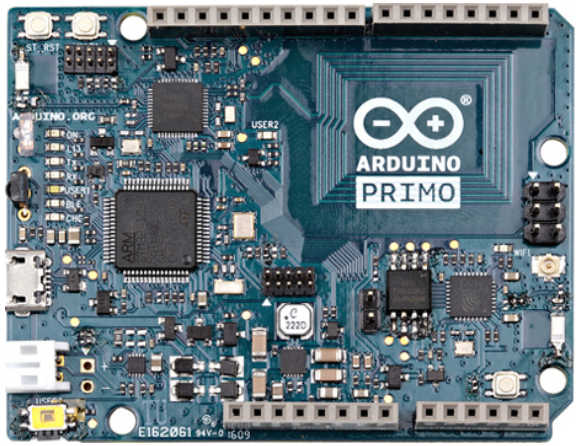
ARDUINO

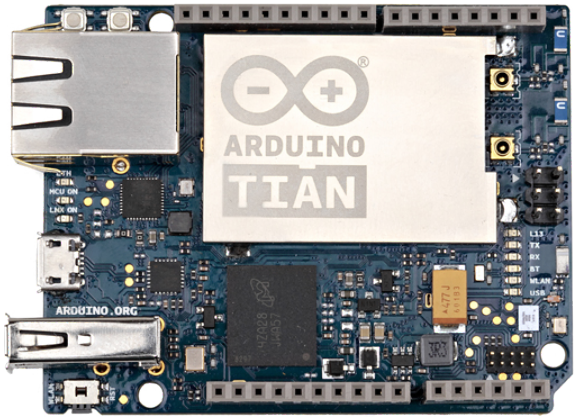
Nordic Semiconductors brings their know-how to Arduino developing a custom bluetooth 6lowpan daemon running on **Arduino Tian** and custom firmware for **Arduino Primo** Boards.

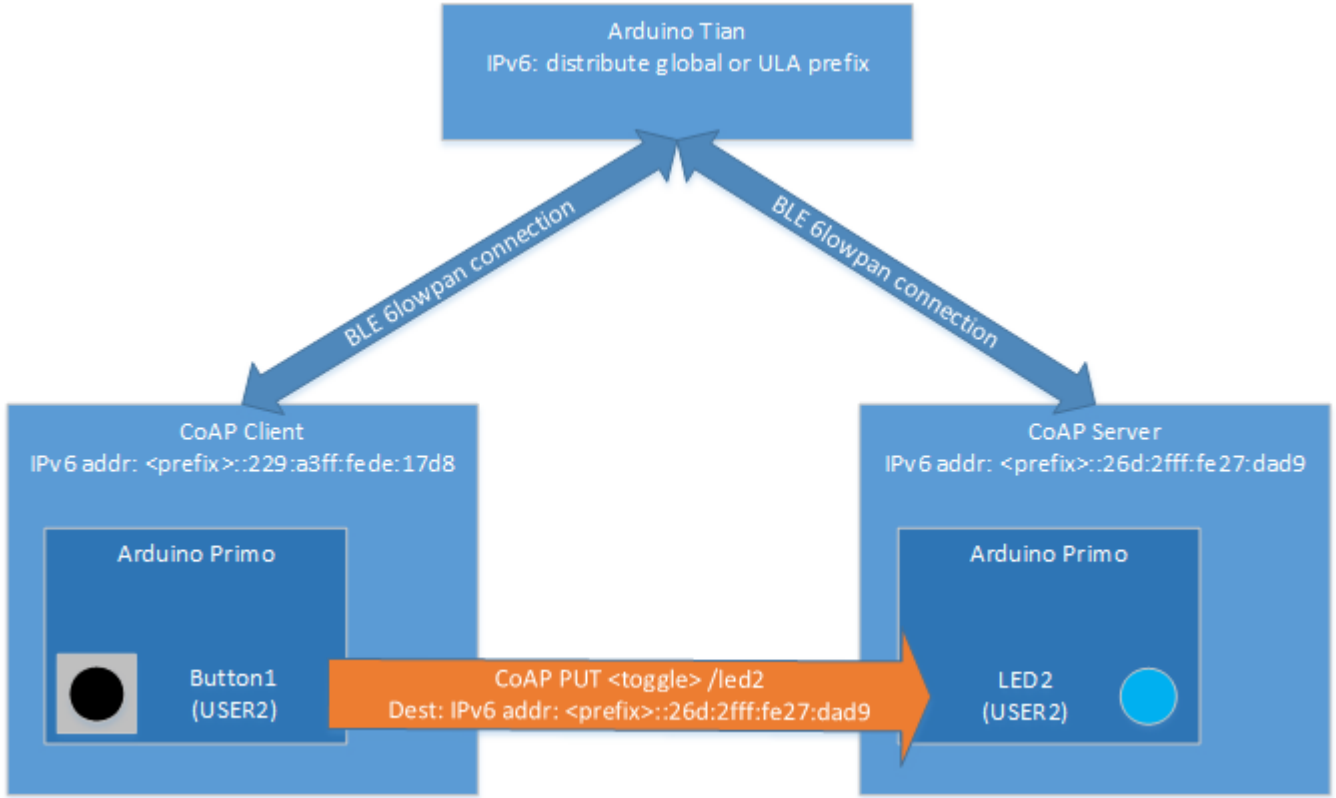
- 1x Arduino Tian
- 2x Arduino Primo
- Bluez v5.35 or higher
- Chaos Calmer 15.05 with custom kernel v3.18.11

Let's see the setup :

Nordic IoT 6lowpan daemon + Tian







Nordic IoT 6lowpan daemon + Tian

The bluetooth 6lowpan daemon is generated by applying a simple patch when building the bluetooth stack itself and then packaged as a standalone IPK.

For further reference, please point your browsers at:

Nordic Github :

https://github.com/NordicSemiconductor/arduino-primo-iot-examples

https://github.com/NordicSemiconductor/Linux-ble-6lowpan-joiner

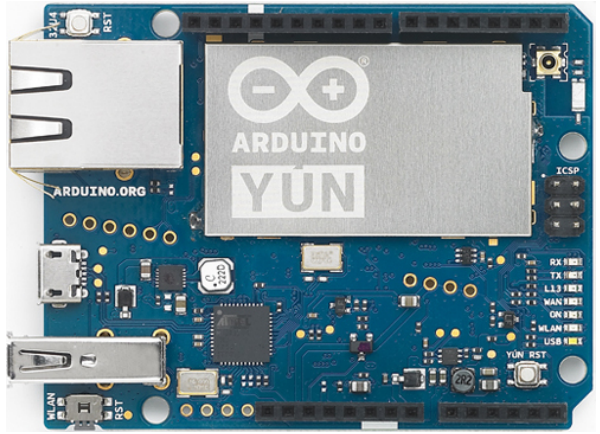
Linino Github :

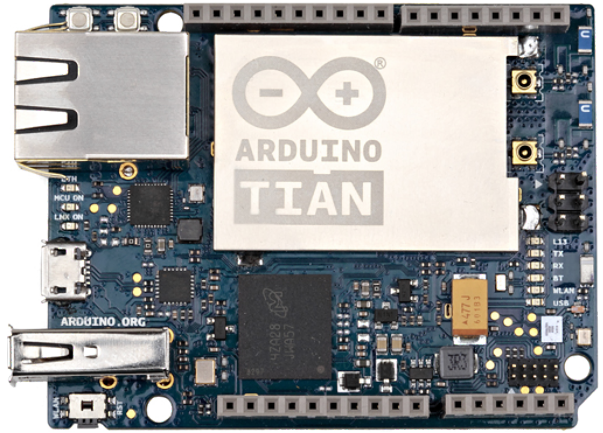
https://github.com/linino/linino_CC/tree/samd

Bluetooth audio with external speaker

Bluetooth speaker connection (scenario)

- Arduino Tian + BLE speaker
- Arduino Yun + BLE dongle + BLE speaker
- Bluez v5.35
- Pulseaudio 6.0
- OpenWRT Chaos Calmer 15.05 running custom kernel v3.18.11
- Expanded Filesystem (Pivot Overlay)

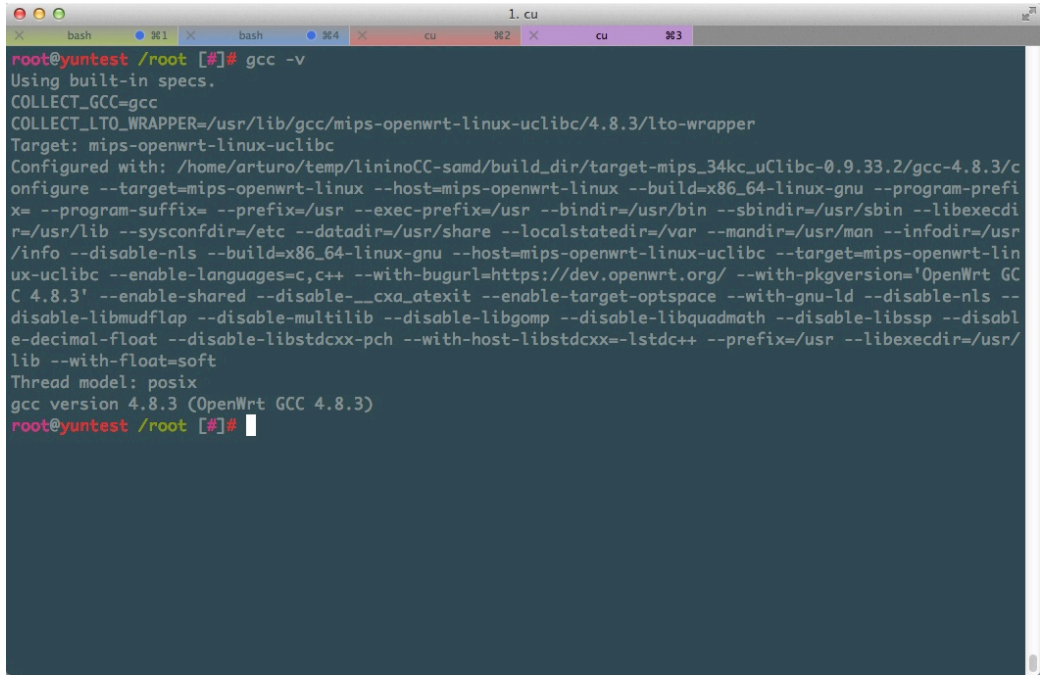




Bluetooth audio with external speaker

```
// bluetooth speaker connection

// initialize bluetooth interface
$ hciconfig -a hci0
$ hciconfig -a hci0 up
$ hciconfig -a hci0 piscan
$ hciconfig -a hci0 sspmode enabled

// entering bluetooth console and attach device
$ bluetoothctl
$ power on
$ agent on
$ default-agent

// scanning for available BT devices
$ scan on
$ scan off
$ trust <MY-MAC-ADDRESS>
$ pair <MY-MAC-ADDRESS>
$ connect <MY-MAC-ADDRESS>

// exiting BT console
$ exit

// set default pulseaudio sink
$ pactl set-default-sink bluez_sink.<MY-MAC-ADDRESS>

// finally play the file
$ paplay <MY-FILE>.wav
```

The “GCC family” – the “plain” GCC



To install it just type from shell:

```
$ opkg update
$ opkg install gcc
```

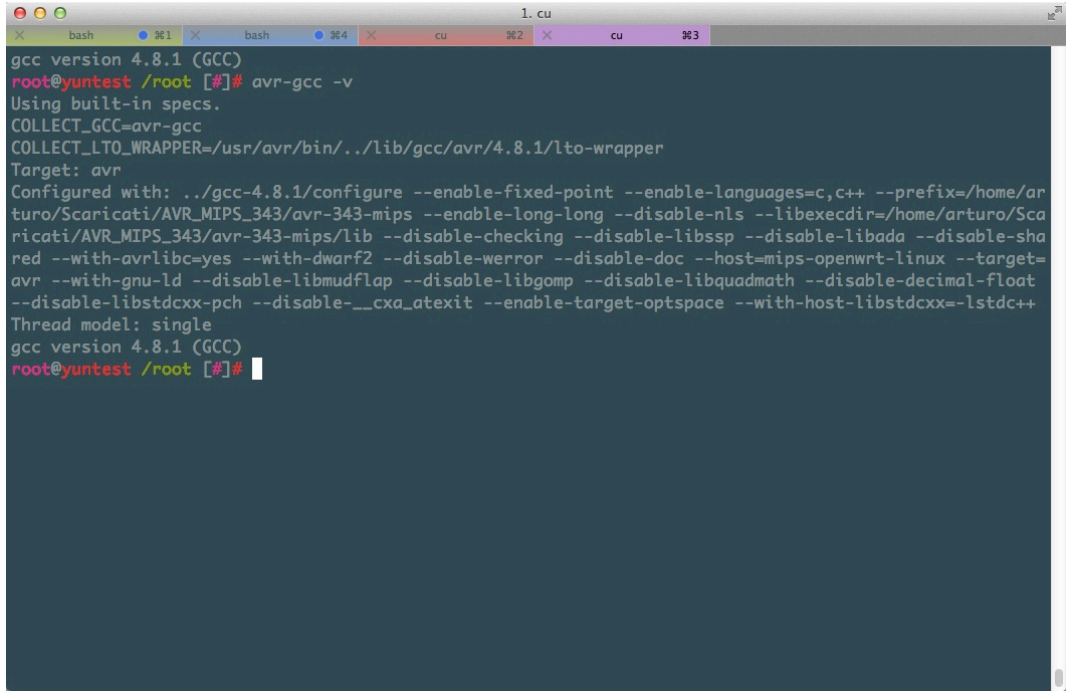
The GCC is officially residing in the OpenWRT repositories since the latest stable release of the distro : Chaos Calmer 15.05

But….we were also able to provide GCC v4.6.3 for our old distro based on Attitude Adjustment 12.09 leveraging the work made for Chaos Calmer 15.05

Please keep in mind the limitations of the current C library (uClibc) when building an executable, or provide patches to enhance its overall functionalities.

That means rebuilding the whole linux image of course…

The “GCC family” – AVR toolchain



Follow our guide here :

http://goo.gl/PiLCFV

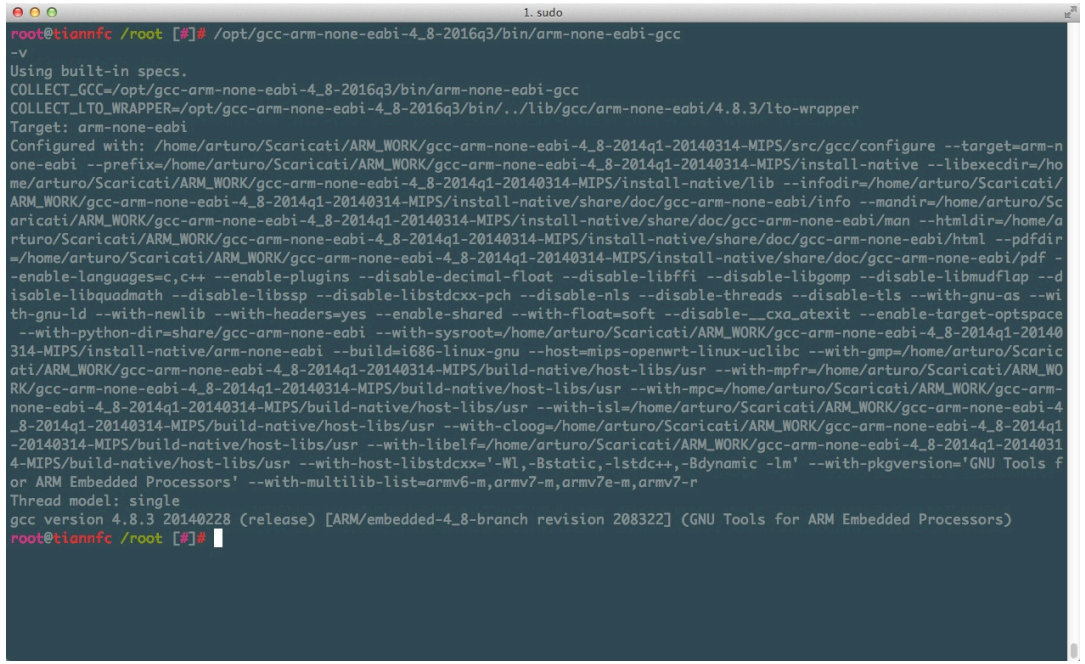
The AVR toolchain has been built with the **Canadian-Cross** compilation :

- is a well known technique for building cross-compilers for other machines especially if the given target is a particular architecture such as **AVR**, **ARM** or **xtensa**.
- It implies the existence of a native ‘target’ compiler in your search **PATH**.

To build the firmware on our board we need :

- the arduino libraries (ipk provided)
- An Arduino *.ino sketch
- The **make** command
- A prebuilt makefile to set the necessary paths

The “GCC family” – ARM toolchain



Then it’s just a matter to replace the original entries with the cross ones in the prebuilt bash scripts shipped with the original sources.

The ARM toolchain has been built with the **Canadian-Cross** technique.

We need to leverage the well-known **Crosstools-NG suite** to generate a sysroot-ed cross-toolchain for building the actual one.

To build the firmware we need, as the **AVR** platform :

- the arduino libraries
- An Arduino *.ino sketch
- The **make** command
- A prebuilt makefile to set the necessary paths for the **ARM** architecture

The “GCC family” – Demonstration

The toolchains have also been built for a testbed **x86** machine for running as VM on VirtualBox.

It has been proven a reliable system to test the software before deploying it on our boards.

To build the firmware for **AVR** or **ARM** platform we again need :

- the arduino libraries
- An Arduino.ino sketch
- The **make** command
- A prebuilt makefile to set the necessary paths for the architecture

So…..let’s run the demo !

THANK YOU

For further information please get in touch with me at :

arturo@arduino.org

prpl | OpenWrt Wireless Freedom