

How To Install A New Platform On TinyOs 2.0

Samuel Boivineau, Raja Jurdak, and Antonio Ruzzelli

Table of Contents

I. Main Steps	3
1. Starting from scratch versus Adapting from a current platform.....	3
2. The process.....	3
II. The example of AquisGrain	4
1. Tree of Tinyos 2.0.....	4
2. The Serial Port.....	8
3. Leds	9
4. Clock	10
5. Radio Frequency Module	11
6. How to download an application on the board.....	12

I. Main Steps

1. Starting from scratch versus Adapting from a current platform

Because many platforms are available today, it is possible to do a comparison between the platform to port and existing platforms to see if some components are common. If so, then the code used can be adapted to the new platform, usually by just looking the wiring of the boards, to check the pins used.

The two main components of a mote are the microcontroller and the radio frequency module.

The microcontroller is usually an Atmega128 or a MSP430. Both are fully supported by many platforms so the code related to each one is available in *tinys-2.x/tos/platforms/<platform>*. It is important to check that every pin is well configured, from the pins connected to the leds to the pins connected to the radio frequency module.

This radio frequency module can be a CC2420 or a CC1000, for example. This component is also fully supported by many platforms.

2. The process

In order to port a new platform, each device needs to be tested alone. Next the whole platform can be tested before releasing the “driver”.

Many applications are available in *tinys-2.x/apps/* and *tinys-2.x/apps/tests/* :

Blink : uses the leds and the timers

TestSerial : uses the serial connection and the timers

TestAM : uses the radio in broadcast mode, and leds and timers

Many others applications can be modified to just check the device tested is working, by blinking the leds for example.

II. The example of AquisGrain

1. Tree of Tinyos 2.0

In Tinyos 2.0, each set of files such as chip-related files, or hardware-related files or component-related files is located in a specific directory.

For example a file dealing with the ADC of Atmega128 is located in "*tinyos-2.x/tos/chips/atm128/adc/*", or a file creating a generic component like a counter is located in "*tinyos-2.x/tos/lib/timer/*". We now list the main directories and their purpose :

tinyos-2.x/	
apps/	Every application should be stored in this directory.
doc/	The doc built by 'make aquisgrain doc'
support/	The system files for make, java, python or c.
tos/	
chips/	Each specific chip has its own directory
lib/	For generic components
platforms/	Each platform has its own directory
sensorboards/	Each platform has its own directory
system/	System components (scheduler etc)
types/	Header files

Add a new platform

There are basically two ways to add a new platform in tinyos-2.0. Either it can be created, or an existing platform can be adapted.

After examining the existing platforms in Tinyos, we identified that MicaZ uses an Atmega128 and a CC2420 tranceiver. Thus, we opted for an adaptation of the Micaz platform to Aquisgrain.

The requirements for a successfull MicaZ-to-Aquisgrain adaptation are :

Identification of application that can validate this specific device (eg: TestSerial for the Uart)

When an application is built, what are the errors, files missing, etc ?

For each file needed, is there any ajustement (pins number, interrupts number, etc) or the file can be copied from the MicaZ platform ?

Finally, is the device working properly ?

The tutorial¹ of tinyos.net can be used as an help tool. However, it is based on an msp340 chip which presents different set of files than our chip.

¹ <http://www.tinyos.net/tinyos-2.x/doc/html/tutorial/lesson10.html>

File .platform

This file defines every file needed by make during the compilation.

```
#
# FILE: aquisgrain/.platform
#
# Includes that should take precedence come first. Platforms come before
# chips because they may override files. These must be specified as
# @includes instead of -I's to @opts, otherwise the %T won't be processed
# by ncc.
#
# $Id: .platform,v 1.4 2006/12/12 18:23:43 vlahan Exp $
#
push( @includes, qw(

    %T/platforms/aquisgrain
    %T/platforms/aquisgrain/chips/cc2420
    %T/chips/cc2420
    %T/chips/atml28
    %T/chips/atml28/adc
    %T/chips/atml28/pins
    %T/chips/atml28/spi
    %T/chips/atml28/i2c
    %T/chips/atml28/timer
    %T/lib/timer
    %T/lib/serial
    %T/lib/power

) );

# Options passed to the compiler
# arch : avr, atmega128, no debug
# scheduler : macros to post tasks to the scheduler

@opts = qw(
    -gcc=avr-gcc
    -mmcu=atmega128
    -fnesc-target=avr
    -fnesc-no-debug
    -fnesc-
scheduler=TinySchedulerC,TinySchedulerC.TaskBasic,TaskBasic,TaskBasic,runTask,postT
ask
);

push @opts, "-mingw-gcc" if $cygwin;
```

File hardware.h

It is included when each .nc file is compiled. *Hardware.h* is used to define the main constants and variables. The timer is accounted separately by a file named *AGTimer.h*.

```
#ifndef _H_hardware_h
#define _H_hardware_h
```

```
#include <atm128hardware.h>
#include <Atm128Adc.h>

#ifndef MHZ
/* Clock rate is ~8MHz except if specified by user
   (this value must be a power of 2, see AGTimer.h) */
#define MHZ 8
#endif

#include "AGtimer.h"

enum {
    PLATFORM_BAUDRATE = 57600L
};

#endif // _H_hardware_h
```

Final tree:

The illustration below shows the final tree of files added to the tinyos-2.x tree, in order to support the Aquisgrain platform.

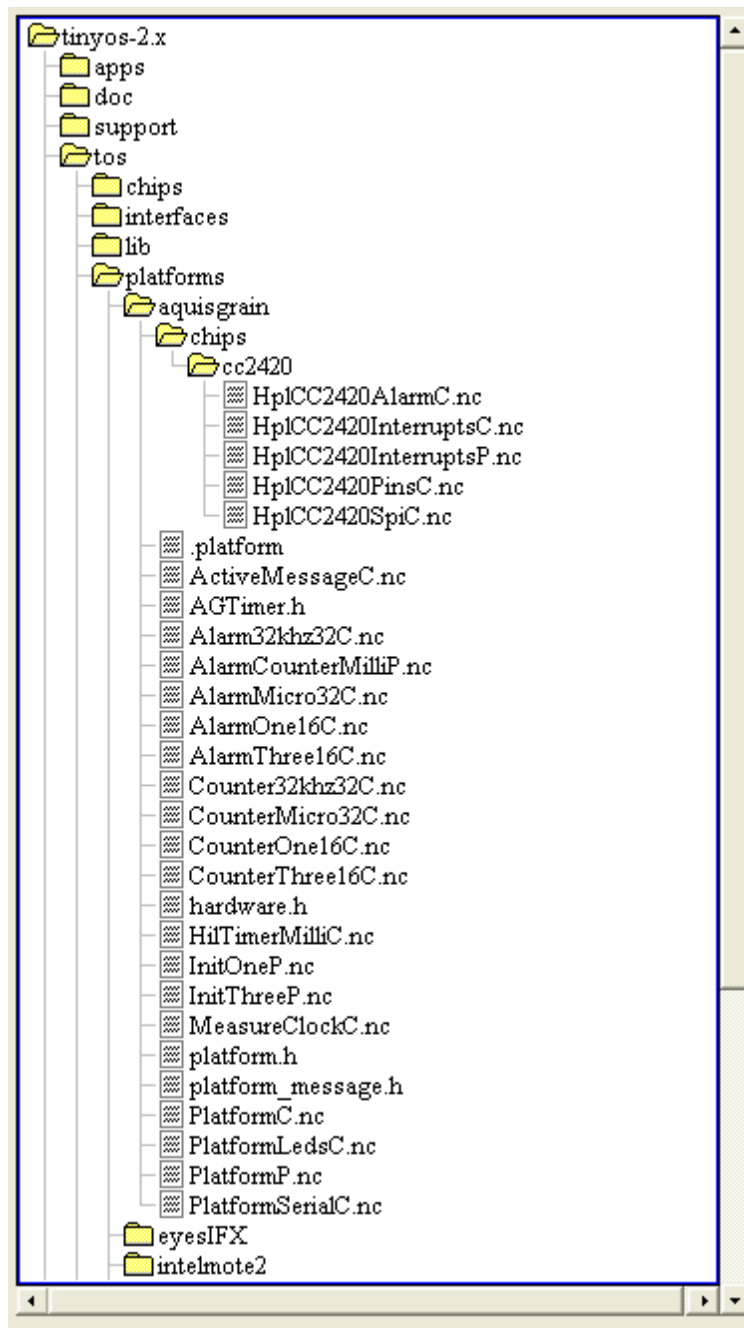


Illustration 1: File tree of Aquisgrain platform

2. The Serial Port

The microcontroller Atmega128 is provided with an UART which is used to connect a sensing device to the MIB510 programming board. This last one is in turn connected to a PC through a serial cable. This system allows two main functionalities :

It is used to program the sensing unit before deployment.

It acts as a gateway between the network of sensor nodes and the user at the PC.

The TestSerial application that is located in "*tinys-2.x/apps/tests*" can be used to test the serial port. By using a timer, this application sends the incremental value of a counter to the serial port. Consequently a led display the value received through the serial port. Thanks to this application, the serial port has been successfully tested for both emission and reception.

The only file to add is *PlatformSerialC.nc* which details UART specifics. The *PlatformSerialC.nc* has not been modified from the version of Mica Family.

3. Leds

The Atmega128 chip is also used on Mica boards, so their files are used like a base.

The configuration of pins is made in *PlatformLedsC.nc* :

```
Led0 = IO.PortA2; // Pin A2 = Red LED
Led1 = IO.PortA1; // Pin A1 = Green LED
Led2 = IO.PortA0; // Pin A0 = Yellow LED
```

In contrast with the tinys version 1, the version 2.0 names leds by number and not by colour. The rest of *PlatformLedsC.nc* file is equal to the one for Mica boards.

Although leds might be tested separately, they have been tested in conjunction with the clock.

4. Clock

Because the clock is dependant of the microcontroller (counters and registers are built-in the microcontroller), we just need to adapt the code of a platform using the Atmega128 chip. So the Micaz board is choosen. All the files related to the counters/timers are copied into the aquisgrain directory :

<i>AGTimer.h</i>	<i>Alarm32khz32C.nc</i>	<i>AlarmCounterMilliP.nc</i>
<i>AlarmMicro32C.nc</i>	<i>AlarmOne16C.nc</i>	<i>AlarmThree16C.nc</i>
<i>BusyWaitMicroC.nc</i>	<i>Counter32khz32C.nc</i>	<i>CounterMicro32C.nc</i>
<i>CounterOne16C.nc</i>	<i>CounterThree16C.nc</i>	<i>HilTimerMilliC.nc</i>
<i>MeasureClockC.nc</i>		

The only one modified to Aquisgrain is **AGTimer.h** (previously named *MicaTimer.h*). See TEP102² for more information about timers and their implementation on tinyos-2.0 They have been tested successfully with the application Blink that uses the timer1.

5. Radio Frequency Module

This module is based on CC2420. Due to many files needed, a directory is created within platforms/aquisgrain/chips/CC2420. The files from MicaZ platform are copied to this directory and fitted to the AquisGrain platform.

HplCC2420InterruptsC.nc : It defines the interrupts used between Atmega128 and CC2420. On AquisGrain, the interrupt INT4 is used for the pin FIFOP while the interrupts for CCA and FIFO pins are emulated through timer polling.

```
InterruptCCA = HplCC2420InterruptsP.CCA;
```

HplCC2420InterruptsP.nc : It defines the implementation of CC2420 interrupts.

HplCC2420AlarmC.nc : This file implements an alarm specific to the chip CC2420.

HplCC2420PinsC.nc : It maps the CC2420 pins to Atmega128 pins for the aquisgrain platform.

```
CCA = IO.PortD5;
CSN = IO.PortB0;
FIFO = IO.PortE5;
FIFOP = IO.PortE4;
RSTN = IO.PortD7;
SFD = IO.PortD4;
VREN = IO.PortB5;
```

HplCC2420SpiC.nc : It specifies the configuration of the SPI bus.

HplCC2420AlarmC.nc : A platform independent abstraction of an asynchronous 32KHz, 16-bit timer for the CC2420.

MotePlatformP.nc : It contains a function *power_init()* that starts the voltage regulator of the CC2420.

Only *HplCC2420PinsC.nc* and *HplCC2420InterruptsC.nc* are modified to fit with AquisGrain.

² <http://www.tinyos.net/tinyos-2.x/doc/html/tep102.html>

6. How to download an application on the board

In order to download the application on the board, first check the number of the serial port provided by windows in the Device Manager.

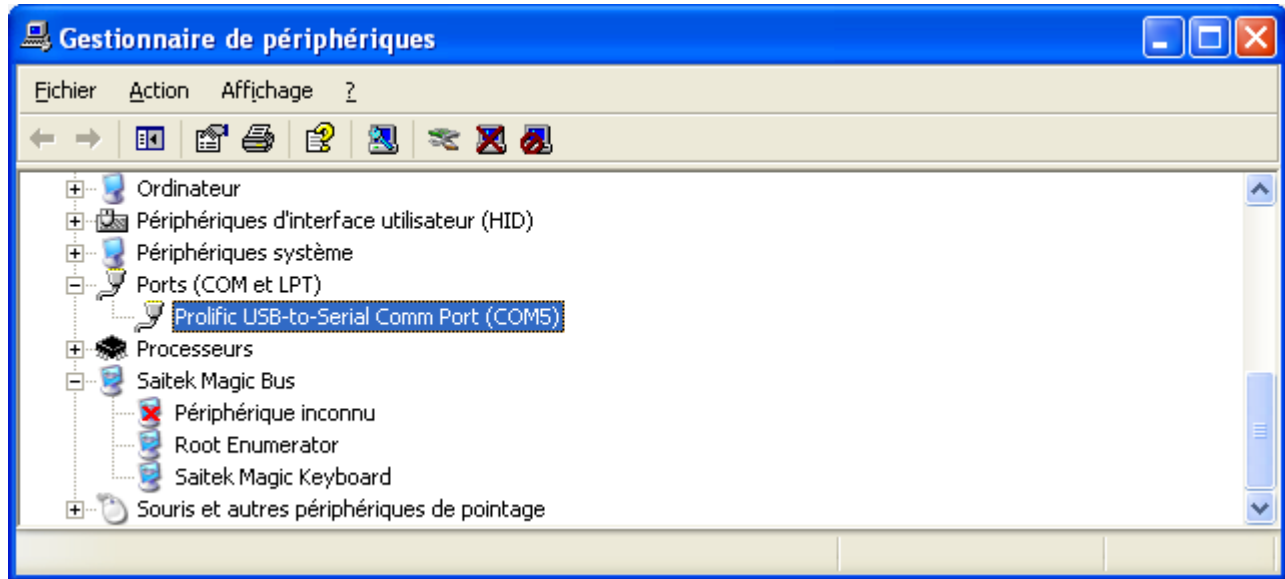


Illustration 2: Check the correct number of serial port to use

Then use this number minus one with `/dev/ttySX` (`COM5` becomes `/dev/ttyS4` and so on). The command to build the binary is :

```
$ make aquisgrain install.0 mib510,/dev/ttySX
```

The pattern of the command is :

```
$ make <platform> install.<MOTE_ID> <programming_board>,<usb_or_serial_port>
```

The options `<programming_board>` and `<usb_or_serial_port>` are optionnal, depending of your platform. For example with the Telosb platform, these options are useless.

The option `install` can be replaced with `reinstall` that just sets the mote Id and download the binary without recompiling it.

The process of compilation is explained below :

```
/opt/tinyos-2.x/apps/MViz
Kaiser Sausser@corleone /opt/tinyos-2.x/apps/MViz
$ make aquisgrain install.0 mib510,/dev/ttyS4
mkdir -p build/aquisgrain
mig -target=null -java-classname=MVizMsg java MViz.h mviz_msg -o MVizMsg.java
javac MVizMsg.java
1   compiling MVizAppC to a aquisgrain binary
ncc -o build/aquisgrain/main.exe -Os -finline-limit=100000 -Wall -Wshadow -Wnesc
-all -target=aquisgrain -fnesc-cfile=build/aquisgrain/app.c -board=micasb -I/opt
/tinyos-2.x/tos/lib/net/ -I/opt/tinyos-2.x/tos/lib/net/ctp -I/opt/tinyos-2.x/tos
/lib/net/le -I. -fnesc-dump=wiring -fnesc-dump='interfaces(!abstract())' -fnesc
dump='referenced(interfacedefs, components)' -fnesc-dumpfile=build/aquisgrain/wi
2   ring-check.xml MVizAppC.nc -lm
   compiled MVizAppC to build/aquisgrain/main.exe
   24124 bytes in ROM
   1672 bytes in RAM
avr-objcopy --output-target=srec build/aquisgrain/main.exe build/aquisgrain/main
.srec
3   avr-objcopy --output-target=ihex build/aquisgrain/main.exe build/aquisgrain/main
.ihex
   writing TOS image
4   tos-set-symbols build/aquisgrain/main.srec build/aquisgrain/main.srec.out-0 TOS_
CODE_ID=0 ActiveMessageAddressC$addr=0
   installing aquisgrain binary using mib510
uisp -dprog=mib510 -dserial=/dev/ttyS4 --wr_fuse_h=0xd9 -dpart=ATmega128 --wr_fu
se_e=ff --erase --upload if=build/aquisgrain/main.srec.out-0
Firmware Version: 2.1
Atmel AVR ATmega128 is found.
5   Uploading: flash
Fuse High Byte set to 0xd9
Fuse Extended Byte set to 0xff
rm -f build/aquisgrain/main.exe.out-0 build/aquisgrain/main.srec.out-0
Kaiser Sausser@corleone /opt/tinyos-2.x/apps/MViz
$
```

Illustration 3: Build and download of an application to aquisgrain platform

- 1 Firstly, the message is built by using mig, the message interface generator for nesC. Mig needs a header file, describing the message structure, and mig will generate nesC files, so that we can use the message structure over the radio and the serial connection.
- 2 Next, the NesC files (*.nc) are converted in one C file (app.c), and this file is compiled to exec (main.exe).
- 3 Next, the binary file is translated in many formats, by using avr-objcopy with platform-specific options.
- 4 Next, the symbols are set (the mote id in our case) on the binary.
- 5 Finally the mote is programmed by using the programming board options specified in the command.