An operating system for the MSP430 microcontroler: development of a FAT driver for TinyOS-2.x

G. Goavec-Merou, J.-M Friedt
Association Projet Aurore, Besançon

August 1, 2009

Why an OS on a microcontroler ?

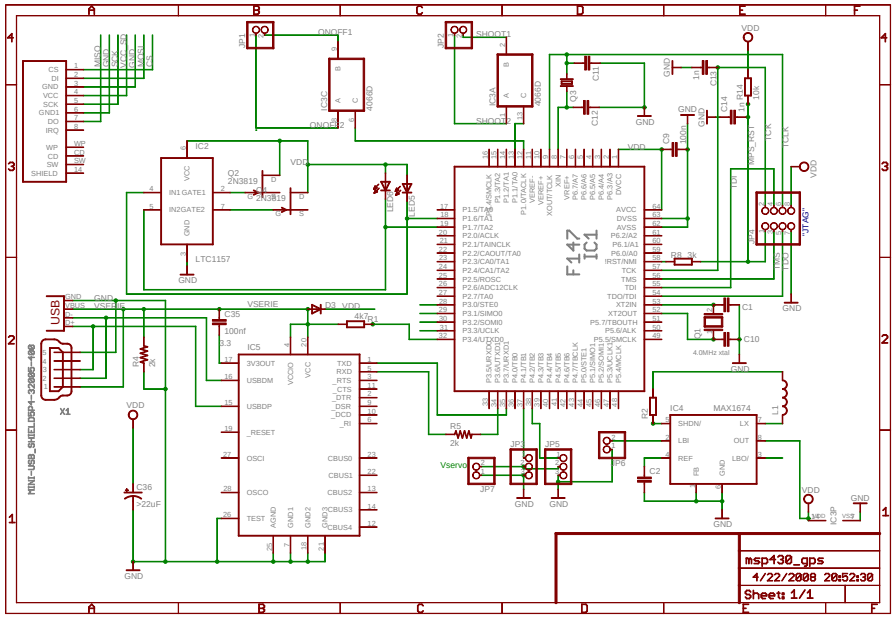
- *Sensor networks* are under development for a wide range of applications: a set of *nodes* spatially distributed monitor environmental properties
- complex and reusable software ⇒ dedicated operating system for such applications (low memory footprint & power consumption)
- **TinyOS-2.x** is a free (BSD license) operating system (a.k.a executive environment) for embedded platforms
- Data acquisition during periodic wakeup sequences
- **Data storage** over a filesystem: radiofrequency links are not (yet) reliabl enough to answer our needs (+ consumption & communication range)

Presentation

- Two main systeme classes:
  - Powerful but enrgy consuming platforms.
  - Low power platforms will litte energy consumption.
- A sensor node must be
  - autonomous
  - use little electrical power to run on batteries

Objective:
comfort of developing an application in the context of the former (OS) within the constraints of the latter: *executive environment* generates a monolithic application within the context close to that found in an OS.

Node architecture



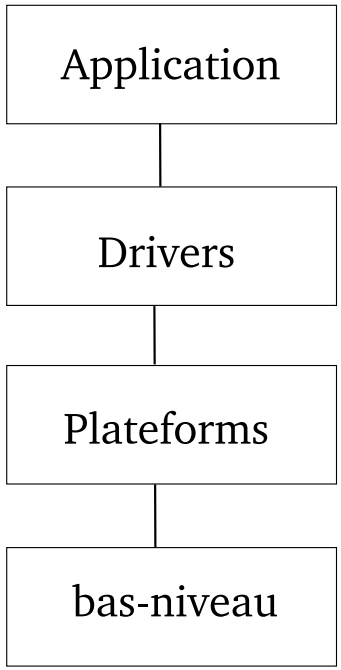
A non commercial platform to make sure we understand the whole configuration procedure of TinyOS-2.x:

- MSP430F149 microcontroler (60 KB flash, 2 KB de RAM)
- L1 C/A ET312 GPS receiver (RS232)
- removable SD card (SPI bus)

TinyOS-2.x

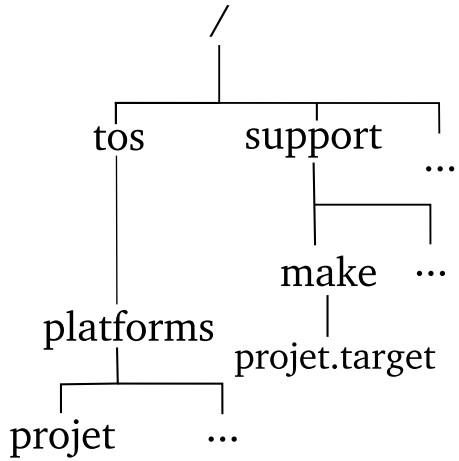
Compromise between low level programming and an operting system:

- set of routines
- structure including abstraction layers during the *development phase*
- uses the nesC programming language
- generates a single monolithic application



Node integration

- Defined by two items:
  - a declaration file
  - a configuration directory



Platform directory content


```
PLATFORM = projet
$(call TOSMake_include_platform,msp)

projet: $(BUILD_DEPS)
  @:
```

- projet.target
- .platform
- hardware.h
- PlatformC.nc
- PlatformP.nc

Platform directory content

- projet.target
- .platform
- hardware.h
- PlatformC.nc
- PlatformP.nc

```
push( @includes, qw(

%T/chips/msp430
%T/chips/msp430/adc12
%T/chips/msp430/dma
%T/chips/msp430/pins
%T/chips/msp430/timer
%T/chips/msp430/usart
%T/chips/msp430/sensors
%T/lib/timer
%T/lib/serial
%T/lib/power
));

push ( @opts, qw(
-gcc=msp430-gcc
-mmcu=msp430x149
-fnesc-target=msp430
-fnesc-no-debug
-fnesc-scheduler=TinySchedulerC,TinySchedulerC →
    ↪ .TaskBasic,TaskBasic,TaskBasic,runTask, →
    ↪ postTask
));
```

Platform directory content

- projet.target
- .platform
- hardware.h
- PlatformC.nc
- PlatformP.nc

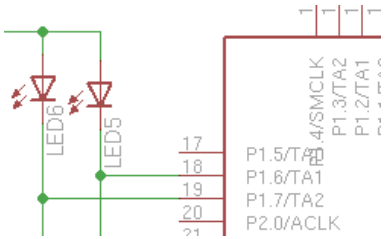
```
#include "hardware.h"

configuration PlatformC {
  provides interface Init;
}
implementation {
  components PlatformP, Msp430ClockC;
  Init = PlatformP;
  PlatformP.Msp430ClockInit -> Msp430ClockC. →
      ↪ Init;
}
```

Platform directory content

- projet.target
- .platform
- hardware.h
- PlatformC.nc
- PlatformP.nc

```
#include "hardware.h"

module PlatformP {
  provides interface Init;
  uses interface Init as Msp430ClockInit;
  uses interface Init as LedsInit;
}
implementation {
  command error_t Init.init() {
    call Msp430ClockInit.init();
    call LedsInit.init();
    return SUCCESS;
  }
  default command error_t LedsInit.init() { →
    ↪ return SUCCESS; }
}
```

Validating the platform: blinking LEDs



Application example: **apps/Blink**

```
#include "hardware.h"
configuration PlatformLedsC {
  provides interface GeneralIO as Led0;
  provides interface GeneralIO as Led1;
  provides interface GeneralIO as Led2;
  uses interface Init;
}
implementation {
  components HplMsp430GeneralIOC as GeneralIOC,
    new Msp430GpioC() as Led0Impl,
      new Msp430GpioC() as Led1Impl;

  components new NoPinC() as Led2Impl;
  components PlatformP;

  Init = PlatformP.LedsInit; // Raccorde l'event →
       ↪ init    celui de PlatformP

  Led0 = Led0Impl;
  Led0Impl -> GeneralIOC.Port16;
  Led1 = Led1Impl;
  Led1Impl -> GeneralIOC.Port17;
  Led2 = Led2Impl; // No led2 on board
}
```

Beyond the LED driver: using an LCD


```
#include "hardware.h"
configuration PlatformLcdC {
  provides interface GeneralIO as LcdData0;
[...]
  provides interface GeneralIO as LcdData3;
  provides interface GeneralIO as LcdE;
  provides interface GeneralIO as LcdRS;
  uses interface Init;
}
implementation {
  components HplMsp430GeneralIOC as GeneralIOC,
    new Msp430GpioC() as LcdData0Impl,
[...]
    new Msp430GpioC() as LcdData3Impl,
    new Msp430GpioC() as LcdEImpl,
    new Msp430GpioC() as LcdRSImpl;

  components PlatformP;

  Init = PlatformP.LcdInit;

  LcdData0 = LcdData0Impl;
  LcdData0Impl -> GeneralIOC.Port24;

[...]

  LcdData3 = LcdData3Impl;
  LcdData3Impl -> GeneralIOC.Port27;

  LcdE = LcdEImpl;
  LcdEImpl -> GeneralIOC.Port23;

  LcdRS = LcdRSImpl;
  LcdRSImpl -> GeneralIOC.Port22;
}
```

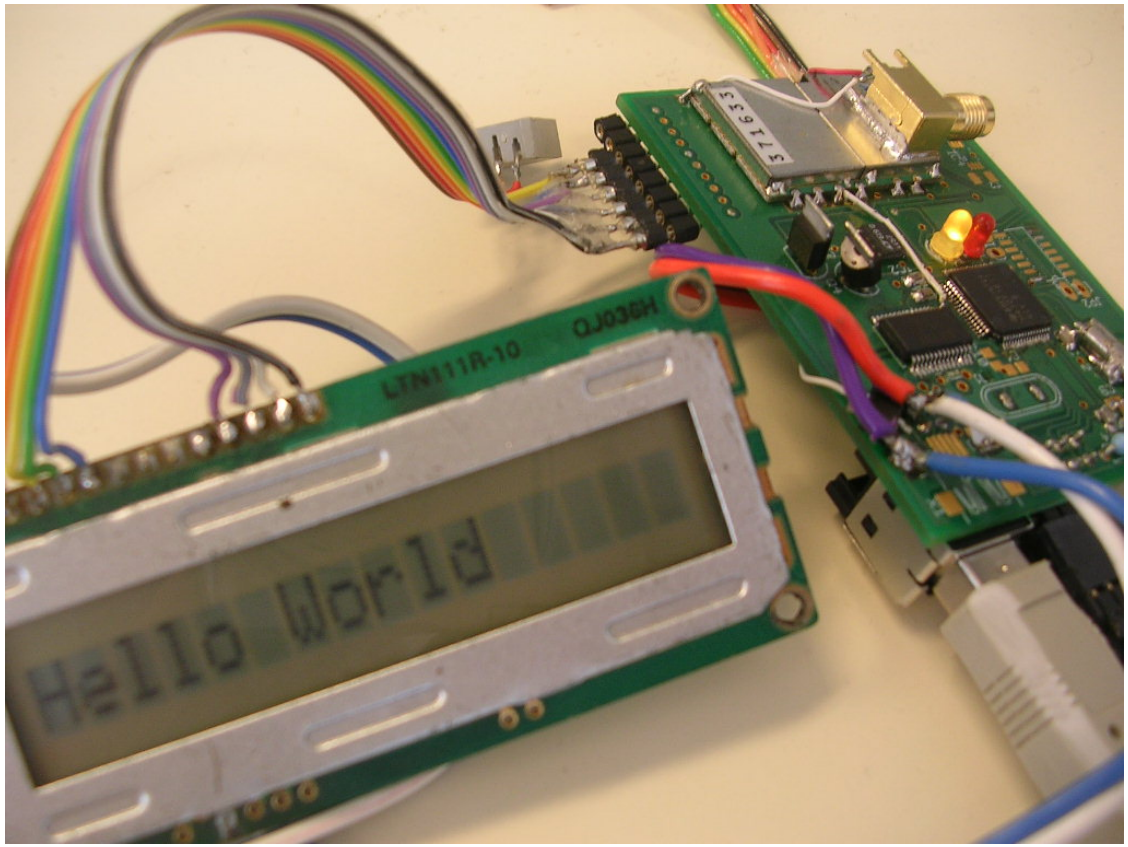
Exploiting the LCD driver

Bitwise signal manipulation

```
// Ecrit un demi octet
void writeDB(uint8_t val) {
    val = val & 0x0f;
    if (val & LCD_DATA0)
      call LcdData0.set();
    else call LcdData0.clr();

    if (val & LCD_DATA1)
      call LcdData1.set();
    else call LcdData1.clr();

    if (val & LCD_DATA2)
      call LcdData2.set();
    else call LcdData2.clr();

    if (val & LCD_DATA3)
      call LcdData3.set();
    else call LcdData3.clr();
}
```

LCD results

An interface useful for debugging and interacting with a user



Asynchronous communication

- RS232: two-way communication protocol.
- TinyOS model.
  - packets are embedded for network routing ($\simeq$ IP header on ethernet)
  - incompatible with a GPS receiver (NMEA sentences do not comply with this encapsulation)
- use of *raw* RS232 communication

RS232 communication

Raw RS232 interface

- The public interface.
- The configuration.

```
#include "hardware.h"
configuration PlatformSerialC {
  provides interface StdControl;
  provides interface UartStream;
  provides interface UartByte;
}
implementation {

  components new Msp430Uart1C() as UartC, projetSerialP;
  UartStream = UartC.UartStream;
  UartByte = UartC.UartByte;
  StdControl = projetSerialP.Control;
  projetSerialP.Msp430UartConfigure <- UartC.Msp430UartConfigure;
  projetSerialP.Resource -> UartC.Resource;
  projetSerialP.ResourceRequested -> UartC.ResourceRequested;
  components LedsC;
  projetSerialP.Leds -> LedsC;
}
```

RS232 communication

Raw RS232 interface

- The public interface.
- The configuration.

```
msp430_uart_union_config_t msp430_uart_proj_config = { {
  ubr: UBR_32KHZ_4800,
  umctl: UMCTL_32KHZ_4800,
  ssel: 0x01,
  pena: 0,
  pev: 0,
  spb: 0,
  clen: 1,
  listen: 1,
  mm: 0,
  ckpl: 0,
  urxse: 0,
  urxeie: 1,
  urxwie: 0,
  urxe: 0,
  utxe: 1} };
```

Example of an OEM GPS receiver

- Continuous position information reception
- Uses ASCII sentences (NMEA) = termination character defines the end of the sentence
- Asynchronous communication (RS232, 4800 bauds)
- Bandwidth: 144 B/s

```
$GPRMC,101236.000,A,4821.6999,N,00446.5093,W,0.94,167.02,281208,,*11
$GPGGA,101237.000,4821.6991,N,00446.5093,W,1,05,8.2,5.1,M,52.5,M →
      ↪ ,,0000*40
$GPRMC,101237.000,A,4821.6991,N,00446.5093,W,0.81,167.67,281208,,*1F
$GPGGA,101238.000,4821.6987,N,00446.5092,W,1,05,8.2,5.4,M,52.5,M →
      ↪ ,,0000*4C
```

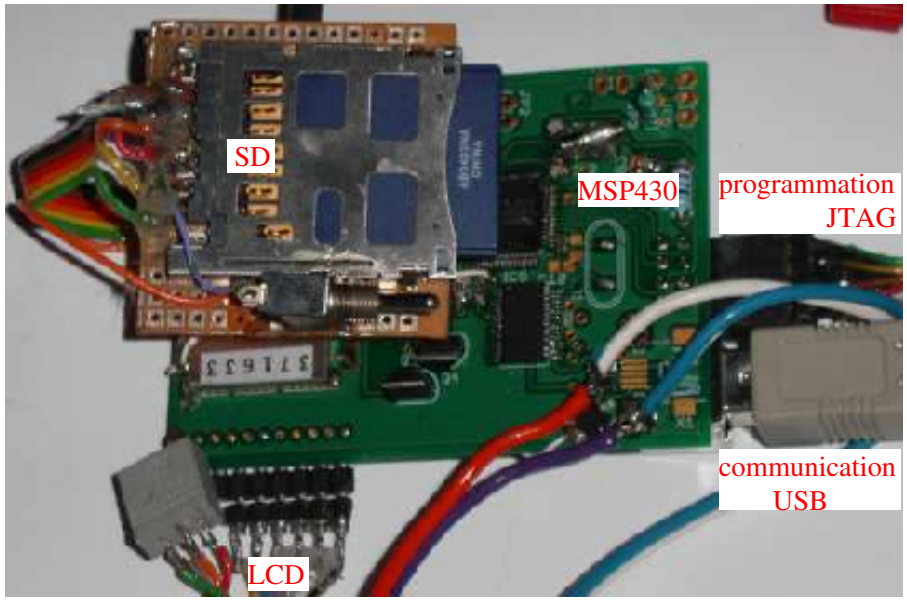
Synchronous communication

SPI: asymetric master-slave protocol.

- $N + 2$ signals to communicate with N slaves : MOSI (Master Out Slave In), MISO (Master In Slave Out) and N Chip Select signals
- high bandwidth (shared clock)
- hardware implementation on most microcontrolers.

Use of the SPI bus

Implementation of a module managing a Secure Digital card (SD) for non volatile mass storage.



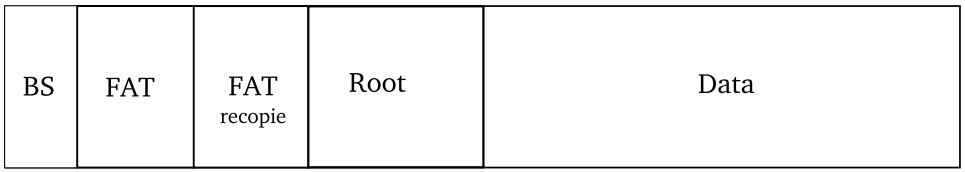
- SPI compatible
- Large storage capacity.
- Default buffer size 512 bytes, might be reduced during config.
- Lack of native drivers for TinyOS-2.x
- Blocking model selected

Filesystems

- Objectives:
  - Efficient data saving.
  - Direct data retrieval.
  - Acquired data are *structured*
- TinyOS model
- Which filesystem ?
  - simple and hence small memory footprint/power consumption
  - cross-platform.
  - designed for older personnal computers.
  - still in use.

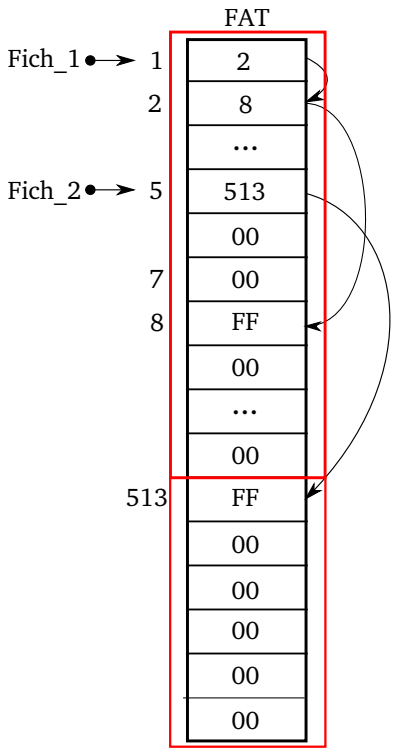
Presentation of FAT

- Access through the Master Boot Record (MBR)
- Partition
- Clusters
- File Allocation Table
- Implemented as multiple modules
- Bandwidth: 1.6 kB/s



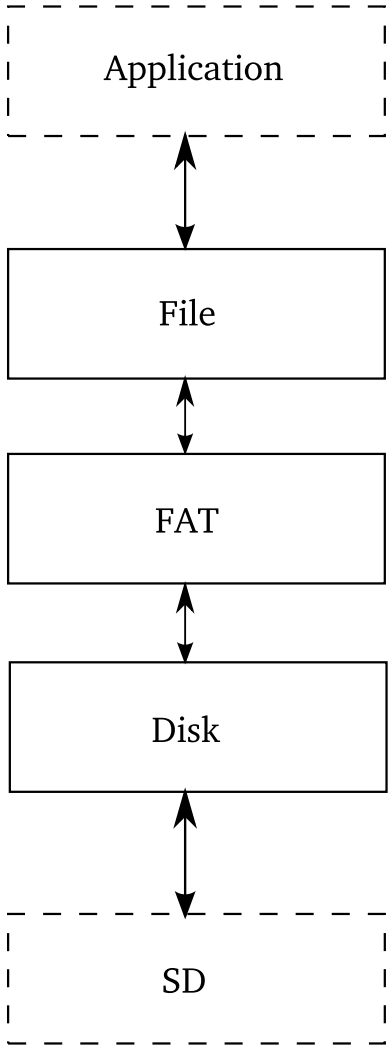
Presentation of FAT

- Access through the Master Boot Record (MBR)
- Partition
- Clusters
- File Allocation Table
- Implemented as multiple modules
- Bandwidth: 1.6 kB/s



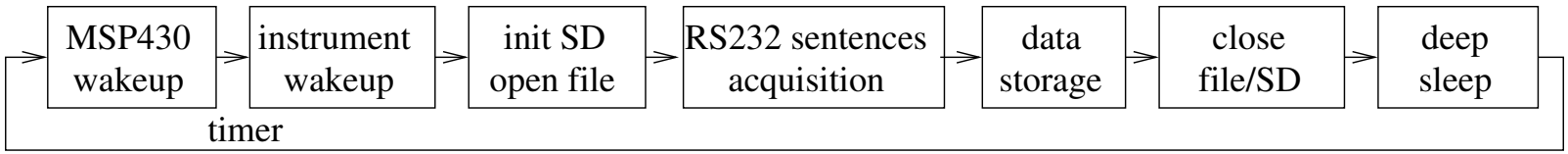
Presentation of FAT

- Access through the Master Boot Record (MBR)
- Partition
- Clusters
- File Allocation Table
- Implemented as multiple modules
- Bandwidth: 1.6 kB/s



Example: periodic acquisition and data storage in a file


```
/* Dclenchement du timer */
event void Timer0.fired() {
  call fatControl.start();
}
/* Fat fini de s'init */
event void fatControl.startDone(error_t error){
  if (error == SUCCESS) call fileControl.start();
}
/* File finit son init */
event void fileControl.startDone(error_t error) {
      if (error == SUCCESS)
    call ReadStream.postBuffer(tampon+5, (uint16_t)NBLINES);
}
/* La lecture est finie */
event void ReadStream.bufferDone(error_t result,
        uint8_t* buf, uint16_t count) {
  if (result == SUCCESS)
          call file.write(tampon,count+5); // Ecriture
  call fatControl.stop(); // Tout teindre
}
event void fatControl.stopDone(error_t error){
  if (error == SUCCESS)
    call fileControl.stop();
}
```



Results

Development of an application using the drivers for :

- Data acquisition and storage on an SD card on the embedded device
- Data transfer to computer using SD card reader
- Use of the data to generate a graph

Track characteristics

- 68004 sentences
- Duration 11 hours
- File size: 4.7 MB

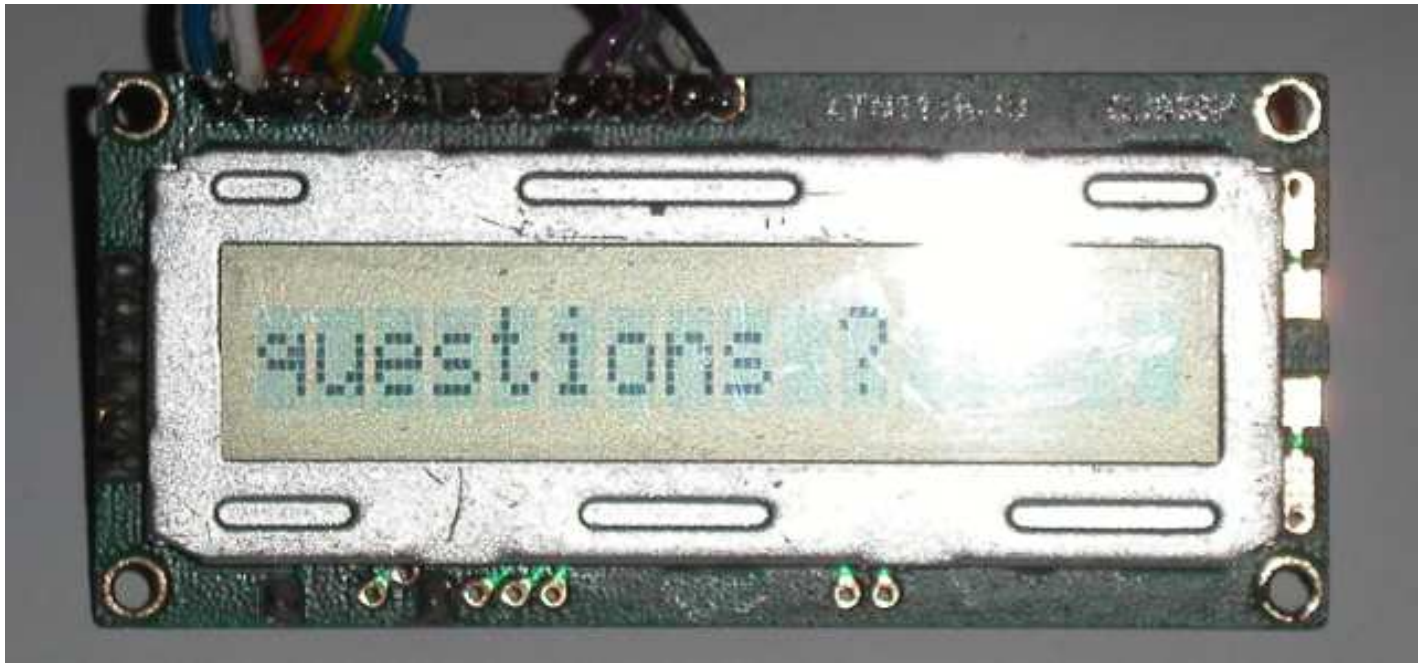
Conclusion

Working environment close to the one familiar for development under an OS

- easy port of a C program thanks to nesC
- transparent port from one platform to another
- easier code maintenance, focus on the innovative work
- our contribution: a driver for data storage (result of analog to digital conversion, RS232) on a FAT formatted SD card (SPI bus)

The driver has been validated on a Crossbow Telos2 board, **further developments** include testing on a MicaZ platform (AVR processor)

Questions



Further readings: `http://jmfriedt.free.fr/tinyOS.pdf` [in French]