Interdisciplinary Project 2011/2012
Prof. Angelo Serra

Student: Davide Quarta

Evaluation of Waspmote OWNS Platform

Index

Introduction

This paper presents the results of the evaluation of the *Waspmote* OWSN product.

The first phase of our evaluation consisted in studying the documentation supplied by Libelium and developing some simple test code.

Testing the product showed weak points that we tried to fix along the way, the result is the development of a makefile that permits to develop applications for the waspmote without having to rely on the buggy and bulky IDE supplied by Libelium and a project template that permits the integration with Visual Studio (having the makefile is really easy to integrate the product in any decent IDE under any OS where the GCC toolchain and GNU utils are available i.e. Eclipse under linux).

Since we wanted to have a reliable development environment we decided to update the toolchain supplied with the waspmote IDE and resort to the latest releases of the GCC AVR toolchain and AVRLibC library. MHV AVR tools gave us an updated environment.

The last phase consisted in developing some code to test the autonomy of the waspmote:
- reading the integrated temperature sensor
- sending data over a wireless connection using Xbee
- reading the data from the wireless connection and storing it on SD card

The data saved on the SD card has been elaborated and the results have been analyzed to evaluate the autonomy of the waspmote under different energy saving modes (hibernation and sleep).

Open Wireless Sensor Networks

OWSN are sensor networks based on *Open Wireless Sensors*. An Open Wireless Sensor is a device which is open in software and hardware.

There are multiple advantages respect to proprietary solutions.

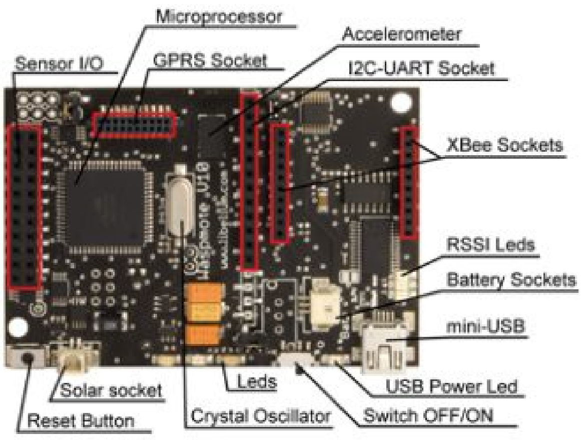
Cost: production costs are lower than proprietary solutions since being open, hardware can be modified and adapted and produced by different companies (which also benefits competition and thus lowers the final cost). Software can be developed also by other companies and with the help of the community.

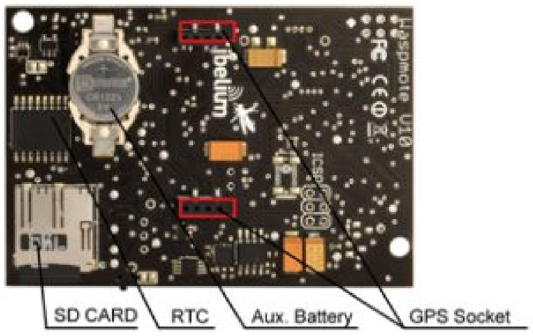
Personalization: as already said hardware can be easily adapted and modified thus leading to different solutions which presents a common base but adapted to different use cases. For an example refer to the different versions of Arduino with different characteristics present in the "*Hardware*" page of Arduino project website[1].

Independence: being hardware and software open source, a customer is independent from a single producer. Should the production of a OWSN solution be discontinued from one company others can continue to produce and support it.

1 http://arduino.cc/en/Main/Hardware

Waspmote





What's waspmote?

Waspmote is an Open Wireless Sensor Platform based on Arduino and developed by Libelium (www.libelium.com) in order to create wireless sensor networks.

Modularity, horizontal and open source approach differentiate the Waspmote from other solutions on the market (Squidbee, Sun SPOT [SUN01]) is that it's certified for commercial purposes (while squidbee is not [LIB01]) and it's architecture is highly modular.

The platform is ready for the three main certification requirements CE (Europe), FCC (US) and IC (Canada) [CHK01].

Every part of the waspmote can be connnected according to use, modules, battery, antennas, sensors and even a solar panel as power supply.

Specifications

Microcontroller:	ATmega1281
Frequency:	8MHz
SRAM:	8KB
EEPROM:	4KB
FLASH:	128KB
SD Card:	2GB
Weight:	20gr
Dimensions:	73.5 x 51 x 13 mm
Temperature Range:	[-20ºC, +65ºC]

Differences from Arduino

An *OWSN* can be also created using various shields for Arduino, like the XBee shield, but applications are somewhat limited by Arduino characteristics.
The 5V-3.3V voltage regulator in the Arduino could not be turned off resulting in a constant current draw of 50mA [CHK01]. This makes this solution not adapt for low power applications which would be desirable in a OWSN environment where we the desired autonomy for the node would be of months (even years).

Apart from that Waspmote uses the same IDE for that reason the same code is easily portable from one platform to the other with slight adjustments like adapting pinout and I/O scheme, this way Arduino users can easily switch from one platform to the other, and new waspmote users can take advantage from the fast learning curve of Arduino.

Comparing the price of a Waspmote and an Arduino board with the same capabilities on the “cooking hacks” store shows that it is pretty much the same, choosing and Arduino or a waspmote is a matter of what fits best the needs of the application developed [CHK01].

	Arduino UNO	**Arduino Mega 2560**	**Waspmote**
Board	€ 22,00	€ 41,00	
Arduino Xbee 802.15.4 + 2dBi antenna	€ 45,00	€ 45,00	
Triple axis accelerometer	€ 7,75	€ 7,75	
On Board Programmable LED + ON/OFF Switch	€ 1,00	€ 1,00	
RTC DS3234 + Button Battery	€ 16,00	€ 16,00	
uSD Adaptor	€ 20,00	€ 20,00	
Solar Panel Socket			€ 135,00
2300mAh Battery	€ 35,00	€ 35,00	€ 18,00
Total	**€ 146,75**	**€ 165,75**	**€ 153,00**

Table 1: Comparing Arduino and Waspmote prices

Comparative tables [CHK01]

Model	**Microcontroller**	**Frequency**	**RAM**	**EEPROM**	**FLASH**	**External Storage (SD card)**
Arduino	ATMega328	16MHz	2KB	1KB	32KB	-
Arduino Mega	ATMega2560	16MHz	8KB	4KB	256KB	-
Waspmote	ATMega1281	8MHz/14MHz	8KB	4KB	128KB	2GB

Table 2: Memory and microcontroller

Model	Analog In	Digital I/O	UART's	SPI	I2C	PWM	USB
Arduino	6	8	1	Yes	Yes	6	Yes
Arduino Mega	16	54	4	Yes	Yes	1	Yes
Waspmote	7	8	4	Yes	Yes	15	Yes

Consumption

Model	Consumption ON	Sleep mode	Consumption Sleep mode	Hibernate mode	Consumption Hibernate mode
Arduino	50mA	No	-	No	-
Arduino Mega	50mA	No	-	No	-
Waspmote	9mA	Yes	62µA	Yes	0.7µA

Table 3: Consumption

Model	IDE	Libraries	Electronic Certifications	Radio Certifications*
Arduino	GPL	LGPL	CE, FCC	-
Arduino Mega	GPL	LGPL	CE, FCC	-
Waspmote	GPL	LGPL	CE, FCC, IC	CE, FCC, IC

Table 4: Licenses and Legal Issues

Sensors

The waspmote exhibits an on board triple axis accelerometer and a temperature sensor integrated in the DS3231SN RTC (the temperature sensor is used by the RTC to recalibrate itself [MAX01] and can be read through the I2C bus).

As for the external sensor the waspmote platform offers 8 integration boards that offer different characteristics that depends on the possible applications for the sensors on board. These are:

- **GASES**: which is used to detected different kind of gases, besides that humidity, temperature and atmospheric pressure. Can be used for environmental control of polluting gases.
- **EVENTS**: to detect vibrations and impact, hall effect, PIR, water level, temperature. It's use is adapt for security, and control of goods applications.
- **SMART CITIES**: has a microphone, crack detection/propagation gauge, detection of dust / pm-10, ultrasound for distance measurement, temperature, humidity and luminosity. Is studied for applications like noise maps, structural health monitoring, air quality monitoring and waste management.
- **SMART PARKING**: has a magnetic field sensor (3-axis) which can be used for a smart parking environment.
- **AGRICULTURE**: serve some agriculture uses like automating a greenhouse, or irrigation system. It shows off measurement of leaf temperature and fruit diameter, soil moisture leaf wetness, solar radiation, humidity temperature, has got also an anemometer, wind vane and

pluviometer.

- **RADIATION**: has a geiger tube that permits monitoring of β and γ radiation
- **SMART METERING**: current, water flow, liquid level, load cell, ultrasound, distance foil, temperature, humidity, luminosity, and has different possible applications ranging from energy measurement to industrial automation.
- **PROTOYPING SENSOR**: is a board which permits an easy integration of any kind of sensor with the waspmote, has a bread-board like pad area, an area to permit soldering of surface mount IC, an amplification stage with a TLC272 op. amp., a current-voltage conversion stage, an ADC (the one included in the ATMEGA microprocessor used in the waspmote, which can be accessed by two pins on the board) and a relay which permits switching on and off power apparatus that can tolerate up to 10A and 250V

[CHK01]

More than this the waspmote is expressly designed to easily integrate sensors and actuators which allow to expand even more the possible applications [LIB02]:
connections to the board are done by using the 2x11 or 1x12 connectors that allow 16 digital input and output signals, of these 7 can be used as analog inputs and 1 as PWM output (DIGITAL1 pin).

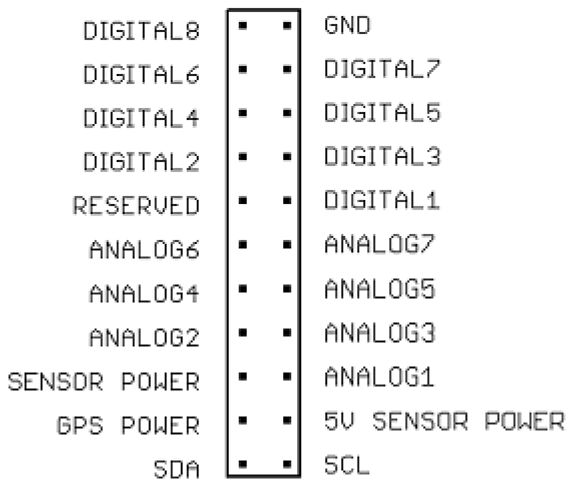


Figure 1: Sensor connector pins

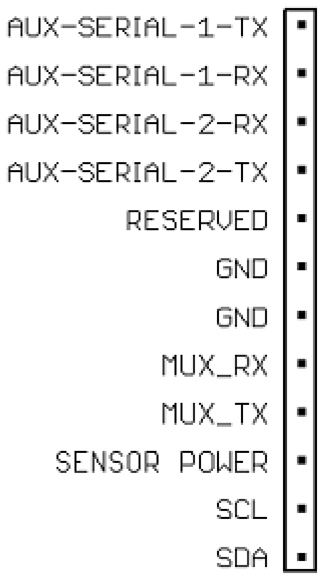


Figure 2: I2C-UART connector pins

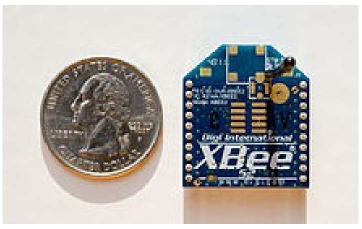
Connectivity

Waspmote solution offer several communication modules that can be connected through the XBee type socket (except for the GPRS/3G module that has it's own socket).

Each module offer support for different kind of network protocols and features, for this project the XBee 802.15.4 has been chosen as the most suitable module.

XBee

XBee is the name for a family of wireless modules sold by Digi [DIGI01]. While every model has the same form factor and is pin compatible they offer support for different radio protocols and different features (like different frequency/power output to adapt to different markets regulations, different maximum range and data rate).



Protocol	Model(s)
ZigBee	ZB, PRO ZB, ZB SMT, PRO ZB SMT
802.15.4	802.15.4, PRO 802.15.4
DigiMesh	PRO DigiMesh 900, DigiMesh 2.4, PRO DigiMesh 2.4
Wireless (802.11 b/g/n)	Wi-Fi
Proprietary	PRO XSC, PRO 900, PRO 868, 865LP*, 868LP*

Table 5: XBee models and supported protocols

*These models support *multipoint* and *digimesh* networks

802.15.4

It's a standard first ratified by IEEE in 2003 [IEEE01] (last revision in the year 2009 [IEEE02]) that defines the ***physical*** and ***MAC*** layers for a low-rate wireless personal area network (LR-WPAN).

What distinguish 802.15.4 from other WPAN solutions is that it is expressly designed towards the objectives of network flexibility, low cost, low power consumption and low data-rate.

To evaluate the autonomy of the waspmote this protocol has been used as we just need a simple point-to-point network.

Digimesh and Zigbee

Digimesh and Zigbee are two network protocols which support the mesh architecture.

Digimesh is a proprietary protocol developed by Digi which implements a peer to peer network architecture and support advanced features like support for sleeping routers and dense[2] mesh networks.

Zigbee instead is instead an open standard ratified by companies members of the Zigbee Alliance[3].

2 *A wireless mesh network having a substantial number of mesh nodes within range (i.e., geographic proximity enabling the effective communication of data) of each other is known as a "dense" wireless mesh network* http://www.faqs.org/patents/app/20090310516

3 http://www.zigbee.org/

The benefits of using the Digi proprietary protocol instead of an open one (Zigbee) are:

- support for robust mesh networking
- support for sleeping of all nodes (in Zigbee only end nodes[4] can sleep)
- easily configurable and more reliable network since all nodes are of the same type (in Zigbee we have three different kind of nodes *coordinators*, *routers*, *end devices*)

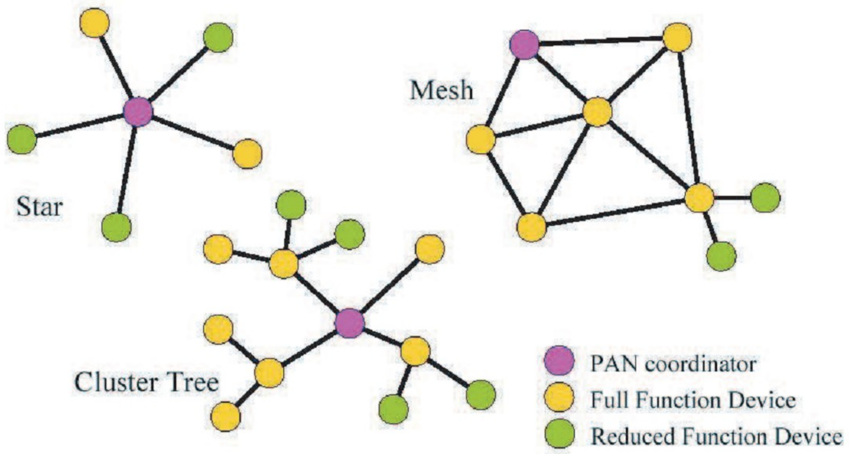


Figure 3: Different kinds of network architecture

Security

Digimesh and Zigbee seems to be both based on 802.15.4 which supports encryption trough 128 bit AES encryption (*AES-CTR*[5]).
The payload field is encrypted using the 128 bit encryption key.

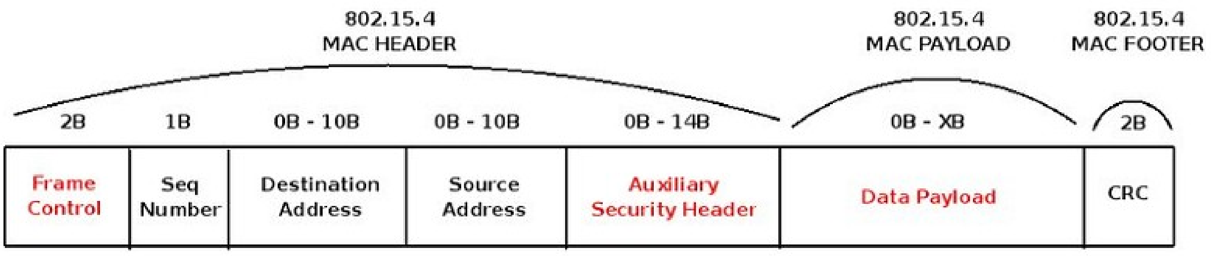


Figure 4: 802.15.4 Frame

Encryption is performed in hardware and supported by the firmware of the XBee card, this as showed in [ZEN01] permits to use encryption with a negligible difference in power consumption.

State	From OFF to ON	Time	From sleep to ON	Time
TX Unicast without encryption	890.82nAh	79.4ms	849.16nAh	76.4ms
TX Unicast with encryption	904.73nAh	79.36ms	863.07nAh	76.36ms
TX Broadcast without encryption	887.79nAh	78,7ms	846.13nAh	75.7ms
RX Broadcast without encryption	889.45nAh	78,6ms	847.79nAh	75.6ms
RX Unicast without encryption	825.52nAh	74ms	783.86nAh	71ms
RX Unicast with encryption	826.11nAh	73.92ms	784.45nAh	70.92ms
RX Broadcast without encryption	818.55nAh	73.4ms	776.89nAh	70.4ms
RX Broadcast with encryption	818.63nAh	73.4ms	776.97nAh	70.4ms

Table 6: Power Consumption

4 *End Devices* can be low-power / battery-powered devices. They have sufficient functionality to talk to their parents (either the coordinator or a router) and cannot relay data from other devices. [DIGI02]
5 Counter-mode, and highly efficient encryption mode which has been proven secure see [UCB01]

Waspmote Development

Development environment: Waspmote IDE

Waspmote's IDE is based on Arduino IDE (actually pretty much the same, just stripped out of the firmware upload feature).
One of the first problems noticed when working with this IDE is that it lacks features of every decent IDE like auto-completion and isn't really flexible when you want to decide what you exactly want compiled into the binary that will get uploaded into the on-board memory.
Another problem that has also been reported to Libelium is that the IDE is really slow when compiling, because every time the waspmote core library is rebuilt.
This last problem will be solved with a new version of the Waspmote IDE[6].

Other limits and problems

The compiler and libraries used for waspmote are the same underneath Arduino, gcc-avr and AVR-LibC.

The ones distributed with the Waspmote IDE are really old versions (WinAVR release 20081205), lots f bugs have been solved in gcc and avr-libc since then, refer to [LIBC01] for a list of changes in libc.

During the development process some bugs have been found, here we try to give a scheme of the various problems and how to avoid them.

First of all programming style guidelines proposed by Libelium have got to be followed ([LIB03]), they already describe some of the possible problems and how to avoid them:

1. declare variables using as little memory as can be used

```
uint8_t x;
x=1;
```

2. use calloc to allocate dynamic memory when large data structure are needed, actually we have found is recommendable to use it whenever we need an array because for some reason with static allocation, access to array elements give some strange problems of memory corruption

```
struct measurement{
  float temp;
  uint8_t hour;
  uint8_t minute;
  uint8_t day;
  uint8_t month;
}

struct measurement *measures;
measures = (struct measurement*)calloc(10, sizeof(struct measurement))
```

3. null pointers after using free to avoid double free problems

```
/* with reference to the precedent example */
free(measures);
measures = NULL;
```

4. avoid to call sprintf more than once in the same function, because of a bug of the AVR compiler. To solve this problem an auxiliary function can be used.

```
char mystring[20];
```

6 http://www.libelium.com/forum/viewtopic.php?p=19833&sid=acd27c095da9706cbc644e41854aeceb#p19833

```
void loop(){
  sprintf(mystring,”%s #%d”,”measure”,1);
  auxFunc();
}
void auxFunc(){
  sprintf(mystring,”%s #%d”,”measure”,2);
}
```

5. Strings printed with USB.print/XBee.print should be defined as a *#define*, otherwise will occupy lot of memory and could lead to memory exaustion problems

```
#define message "Test"
USB.print(message);
Xbee.print(message);
```

6. Free UART buffer after sending many packets via XBee calling Xbee.flush
7. due to the size of the packetXBee struct (500 byte) it is recommended to allocate it using calloc

```
#define GW_MAC "0013A20040558A2F"
void sendPacket(char* data){
  packetXBee* paq_sent;
  paq_sent=(packetXBee*) calloc(1,sizeof(packetXBee));
  paq_sent->mode=UNICAST;
  paq_sent->MY_known=0;
  paq_sent->packetID=0x52;
  paq_sent->opt=0;
  xbee802.hops=0;
  xbee802.setOriginParams(paq_sent, "5678", MY_TYPE);
  xbee802.setDestinationParams(paq_sent, GW_MAC, data, MAC_TYPE);
  xbee802.sendXBee(paq_sent);
  free(paq_sent);
  paq_sent=NULL;
}
```

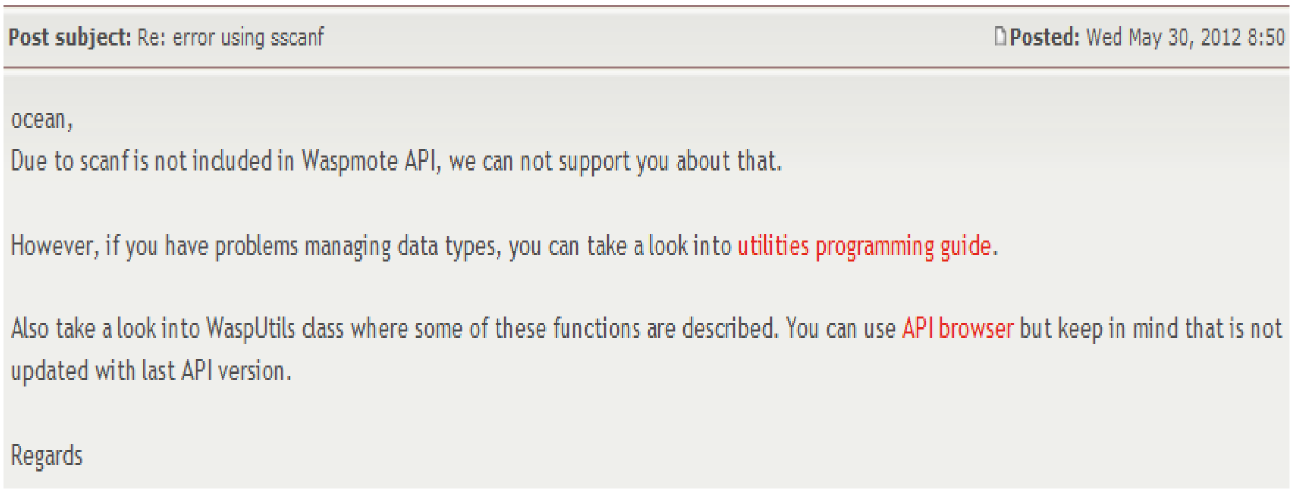
8. Free the memory allocated for a packet received by XBee after using it

```
xbee802.readXBee();
while(xbee802.pos>0)
{
  // Treat the received packet
  free(xbee802.packet_finished[xbee802.pos-1]);
  xbee802.packet_finished[xbee802.pos-1]=NULL;
  xbee802.pos--;
}
```

9. calling the function to get free memory can lead to problems similar to the ones for the sprintf function, follow the recommendation explained for sprintf to avoid problems

```
uint16_t memory=0;
Utils.getFreeMemory();
memory=Utils.freeMemory;
USB.print(“Available free memory: “);
USB.println(Utils.freeMemory);
```

Other than these we have found problems with *float* variables not addressed anywhere.
First of all there are three versions of libc that the binary can be linked against: minimal, standard and float, despite the fact that waspmote IDE links against the float version, float don't work correctly with sscanf.
Libelium only give support on their API, not the environment they supply.


Post subject: Re: error using sscanf Posted: Wed May 30, 2012 8:50

ocean,
Due to scanf is not included in Waspmote API, we can not support you about that.

However, if you have problems managing data types, you can take a look into utilities programming guide.

Also take a look into WaspUtils class where some of these functions are described. You can use API browser but keep in mind that is not updated with last API version.

Regards

All the problems and the limited support of their product can't help the feeling of just playing with a quite expensive toy instead of working with a fully fledged product.

Obtaining a good development environment

Because of a bug of the IDE an initial draft of the code was lost, this and the other problems (slow compilation times, no flexibility of the tool) lead to think of an alternative way of development for waspmote.h

A Libelium user developed a way to use Eclipse as an IDE for waspmote (the website where the process is described is not available anymore), but this alternative didn't satisfy us.

With WinAVR (the compiler environment supplied with the IDE) a makefile template is supplied, we had to modify it with the correct options for the compiler, linker and avrdude (which is used to upload the binary to the rom).
The source for the IDE is not present on Libelium's website, JavaDecompiler was used to decompile and analyze the IDE to look what command line options where to be used, and a makefile was developed.
The makefile includes also a way to ensure the binary uploaded to the waspmote is not too big (which would overwrite waspmote's bootloader):

```
@if [ `wc -c $(TARGET).hex | awk '{print $$1}'` -gt 122880 ]; then echo File is too big!; false; fi
```

After the makefile was developed we decided to use it in a Visual Studio project to get advantage of this IDE advanced features like IntelliSense, all we did is using the *"Makefile template"* of Visual Studio, enabling IntelliSense completion in the options of the project and including the path to the Waspmote SDK (which also has got to be included in the makefile's EXTRAINCDIRS variable) and Libc AVR libraries.

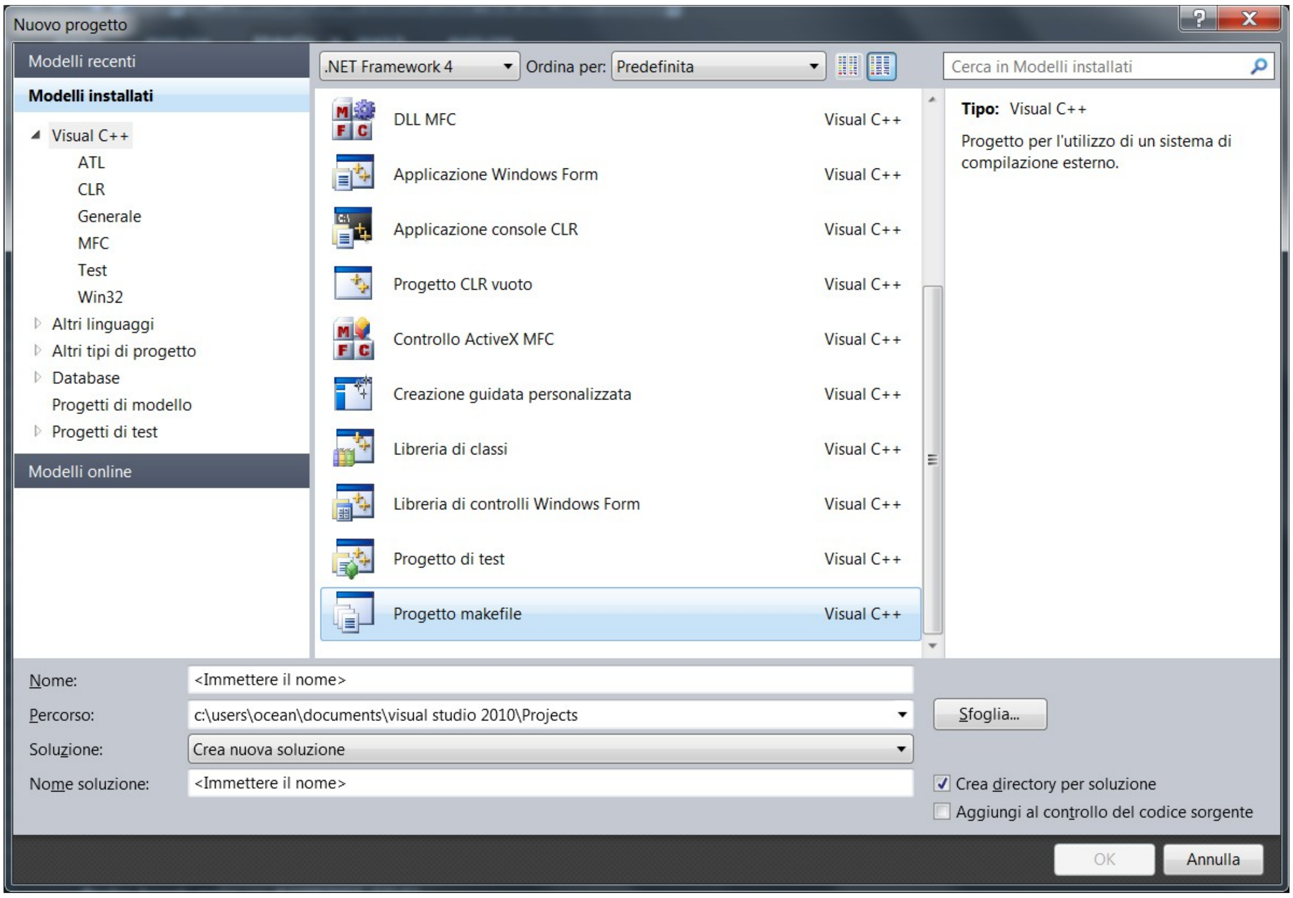
The last step is to compile an example from the waspmote IDE, and get from the build directory (in the %TEMP% for waspmote IDE v2) the core.a rename it to libcore.a and copy it to the project directory.

WinAVR is really old and not supported anymore, *MakeHackVoid*[7], an australian Hackerspace, currently develops and releases MHV AVR tools[8], that replaces WinAVR, and comes with an

7 http://www.makehackvoid.com
8 http://www.makehackvoid.com/node/578/release

updated version of the tool chain.

The installation of MHV tools should be done in a first level directory without spaces in it (i.e. *C:\MHVAVRtools*) to avoid problems (access violation errors) with *make* program.
Avrdude and avrdude.conf must be copied from the waspmote ide and overwrite the ones in the MHV tools otherwise we won't be able to upload the results of the compilation to the waspmote.
After installing the MHV AVR tools and *Core utils* and *Gawk* from GnuWin32[9] utilities paths should be eventually added to the *PATH* system environment variable.
Finally a copy of sh shell has to be copied to the bin directory of gnuwin32 or MHV tools (from either an installation of winavr or from MSYS utilities from MINGW), then our renewed development environment is ready.



.NET Framework 4 | Ordina per: Predefinita | Cerca in Modelli installati

DLL MFC	Visual C++
Applicazione Windows Form	Visual C++
Applicazione console CLR	Visual C++
Progetto CLR vuoto	Visual C++
Controllo ActiveX MFC	Visual C++
Creazione guidata personalizzata	Visual C++
Libreria di classi	Visual C++
Libreria di controlli Windows Form	Visual C++
Progetto di test	Visual C++
Progetto makefile	Visual C++

Tipo: Visual C++
Progetto per l'utilizzo di un sistema di compilazione esterno.

Nome: <Immettere il nome>
Percorso: c:\users\ocean\documents\visual studio 2010\Projects
Soluzione: Crea nuova soluzione
Nome soluzione: <Immettere il nome>
☑ Crea directory per soluzione
☐ Aggiungi al controllo del codice sorgente

Figure 5: Visual Studio 2010, Creating a new Makefile project

9 http://gnuwin32.sourceforge.net/

Waspmote, using OBJ C lists

In the tree of the waspmote IDE was found part of the libobjc library developed by the FSF, we can use this library as a way to easily implement lists, first of all the following headers have got to be included:

```
#include "objc/objc.h"
#include "objc/objc-api.h"
#include "objc/objc-list.h"
```

The structure of the list is easy to understand the head pointer points to the actual element of the list, and tail points to the next element:

```
struct objc_list {
  void *head;
  struct objc_list *tail;
};
```

calling list_cons function (the list constructor) reveals an undefined reference to the objc_malloc function when compiled, we should compile the objc library first.

At the moment there is no information about the successful porting of the objc library on ARM, it is much easier just to copy the needed code and substitute any reference to objc_malloc/free with the correspoding c functions.

Waspmote autonomy test

Since the waspmote has different energy-saving modes we have decided to test two of them and see what the typical autonomy of a waspmote is, this is necessary to carefully plan the intervals in which a waspmote should be woken up to take measures, and eventually to plan the charging/battery substitution for the nodes.
Actually the measure of the battery can also be sent to a gateway to look when it's needed to charge the battery of a node.

Developed code explanation

Our example consists of a "*sender*" node simple loop that gets data (time, battery, temperature and sends it through the Xbee, and a "*receiver*" node that will take the data and store it in the SD card.

The waspmote defines two main functions: setup to initialize the waspmote and loop, that's a infinite loop were the main program should be developed.

A struct has been defined to hold data:

```
/// structs and variables
typedef struct sTime{
       uint8_t month;
       uint8_t date;
       uint8_t hour;
       uint8_t minute;
       uint8_t second;
} stime_t;
```

```c
typedef struct sData // struct which contain sensors data
{
	uint8_t pwr_level; // battery power level
	float temperature; // temperature
	stime_t time;
	// others to be added
} sdata_t;
```

The *readSensors* function gets the data and store it in a struct that's passed as parameter, and is pretty self explanatory:

```c
/// readSensors reads data from sensors
/// and store it in our structure
void readSensors(sdata_t *sensorData)
{
	sensorData->time.month = RTC.month;
	sensorData->time.date = RTC.date;
	sensorData->time.hour = RTC.hour;
	sensorData->time.minute = RTC.minute;
	sensorData->time.second = RTC.second;
	sensorData->temperature = RTC.getTemperature();
	sensorData->pwr_level = PWR.getBatteryLevel();

	printSensorData(*sensorData);

}
```

Since the *sender* and *receiver* will share these informations and we are sending data using the waspmote api (AP2) that only permits to send strings some functions that de/serialize the data from/to strings have been developed:

```c
/// serializeData write the sensors data into the
/// "out" string, for now it's a simple serialization
/// using sprintf
bool serializeData(sdata_t sensorData, char* out,bool terminateString){

	char* format;
	int th,tl;

	if( terminateString ) format = FMT_STR_0;
	else format = FMT_STR;

	floatToInt(sensorData.temperature,&th,&tl);
	sprintf(
		out,
		format,
		sensorData.time.month,
		sensorData.time.date,
		sensorData.time.hour,
		sensorData.time.minute,
		sensorData.time.second,
		sensorData.pwr_level,
		th,
		tl
		);
	//debugPrintln(out);
}

void floatToInt(float f,int* th,int* tl){
	*th = (int)floor(f);
	f -= *th;
	*tl = (int)floor(f*100);
}

/// in the future checks on sensorData can be added
bool deserializeData(char* in, sdata_t* sensorDataOut){

	int th,tl;

	sscanf(
		in,
		FMT_STR,
		&(sensorDataOut->time.month),
		&(sensorDataOut->time.date),
		&(sensorDataOut->time.hour),
		&(sensorDataOut->time.minute),
		&(sensorDataOut->time.second),
		&(sensorDataOut->pwr_level),
		&th,
		&tl
		);

	sensorDataOut->temperature=th+(tl/100);
	return true;
}
```

We used a helper function *floatToInt* because the waspmote api starting from version 0.24 returns a float when calling *RTC.getTemperature()* and in the original development environment sscanf didn't work with floats.

Of course we defined a function to send data and store it on the SD in case the send failed, if data is found on the SD card all the data is sent togheter by the SendAllData function, if the send fails, saveData is called.

In the setup function we just init the Xbee then in the loop function we in turn read the sensors, and try to send the data as already described:

```
void loop()
{
	RTC.ON();

	sdata_t sensorData;

	readSensors(&sensorData);

	if( sendAllData(DATAFILE,sensorData)!=0 ){
		debugPrintln("Error sending data, saving to SD");
		saveData(DATAFILE,sensorData);
	} else {
		debugPrintln("Data sent succesfully!");
	}

	// here we go with power saving :)
	// If Hibernate has been captured, execute hte associated function

	// Hibernate, wake up after 10 sec. (the minimum interval defined by the
#ifdef __HIBERNATE
	PWR.hibernate("00:00:00:10",RTC_OFFSET,RTC_ALM1_MODE2);
#else
	#ifdef __DEEPSLEEP
	PWR.deepSleep("00:00:00:08",RTC_OFFSET,RTC_ALM1_MODE2,ALL_OFF);
	#else
		delay(2000);
	#endif
#endif

	// we prefer using defines to enable code only when needed,
        // this will give us advantage over code size (negligible over
        // performance in this case
        if( intFlag & RTC_INT )
        {
                Utils.blinkLEDs(1000);
                intFlag &= ~(RTC_INT);
        }
}
```

The sendAllData function will try to send eventual older data that failed to send previously and was saved on the SD card, then send the last measurement:

```
/// sendData tries to send current data
/// and untrasmitted data to the XBEE gw
/// error transmitting -> return false
/// transmission's fine -> return true
/// sensorData: defines the actual sensor data
uint8_t sendAllData(char const* datafile, sdata_t sensorData){

        uint8_t result;
        sdata_t* _fSData = (sdata_t*)calloc(1,sizeof(sdata_t)); // sensordata read from
datafile

        /*  if neighbour discovery is enabled try to find the gateway
        this could eventually improve autonomy avoiding to send
        three (or more if set differently) time the packet and waiting for the ack
        in case GW is not available */

#ifdef NEIGHBOUR_DISCOVERY
        #define GW_FOUND "Gateway found!"
        #define GW_NOT_FOUND "Gateway not found!"

        if( isGWavailable()==false ){
                debugPrintln(GW_NOT_FOUND);
                return 1;
        }
        debugPrintln(GW_FOUND);
#endif



        // char* packet;
        // packet = preparepacket(sensorData);
```

```
	int i;
	int numln = SD.numln("sensorData.txt");

	for(i=0;i<numln;i++)
	{
		deserializeData(SD.catln(datafile,i,1),_fSData);
		printSensorData(*_fSData);
		result = _sendData(*_fSData );
		if(result!=0) {
			debugPrintln(ERR_SEND_PACKET);
			// send not succesfull exit from loop
			// and retry later
			break;
		}
	}

	free(_fSData);

	if(result == 0){
		// if result==0 finally try to send actual data
		// and remove the "history" data file
		if( SD.isFile(datafile)==1 && SD.del(datafile)==0 ){
			debugPrint("ERROR: Cannot delete data file ");
			debugPrintln(datafile);
		}
		result = _sendData(sensorData);
	} else {
		debugPrintln(ERR_SEND_PACKET);
		// dontCare(...);

		// the correct behaviour should be to delete the lines that
		// have been correctly sent, for our use is highly improbable
		// and the gateway already filters the duplicates
		// hence we could improve this code if needed later
	}

	return result;
}
```

the function _sendData is what actually send the data to the receiver node, and has been prepared for the two modes of communication AP1 and AP2 (binary data unescaped and string escaped [LIB04]) using the functions available from the waspmote api:

```
uint8_t _sendData(sdata_t sensorData){

	if(++packets_sent > 5 ) XBee.flush(); // follow guidelines

#ifdef __AP1
	return xbee802.send(GWMAC,(char*)(&sensorData),0,sizeof(sdata_t));
#else
	char* data = (char*)calloc(1,MAX_DATA); // TODO serialize data from sensorData
struct!
	serializeData(sensorData, data,true);

	// using api
	packetXBee* paq_sent;      // packet to send

	// Set params to send
	paq_sent=(packetXBee*) calloc(1,sizeof(packetXBee));
	paq_sent->mode=UNICAST;
	paq_sent->MY_known=0; // not used in 802.15.4
	paq_sent->packetID=0x52;
	paq_sent->opt=0;
	xbee802.hops=0;
	xbee802.setOriginParams(paq_sent, "5678", MY_TYPE);
	xbee802.setDestinationParams(paq_sent, GWMAC, data, MAC_TYPE, DATA_ABSOLUTE);
	xbee802.sendXBee(paq_sent);

	free(paq_sent);
	free(data);
	data = NULL;
	paq_sent = NULL;

	return xbee802.error_TX;
#endif
}
```

As seen from the sendAllData function we developed also a function to check for the presence of the gateway/receiver using the node discovery of the xbee, but decided that in the end was not needed because Xbee 802.15.4 protocol can use ACKs to signal that data has been received succesfully [IEEE02].

The function isGWavailable scans the network for neighbour nodes (*brothers*) and check if the gateway is present between them:

```
bool isGWavailable(){
#define SCAN_NET "Scanning the network"
#define SCAN_B       "scanned brothers:"

       bool result;
       int8_t i,j;

       debugPrintln(SCAN_NET);

       xbee802.scanNetwork();

       debugPrint(SCAN_B);
       debugPrintln(xbee802.totalScannedBrothers,DEC);

       result = false;
       for( i=0; i<xbee802.totalScannedBrothers; i++ )
       {
              // sscanf(GWMAC,FORMAT,MAC_HIGH+0,MAC_HIGH+1,MAC_HIGH+2,MAC_HIGH+3); // get
high part of MAC
              // sscanf(GWMAC+8,FORMAT,MAC_LOW,MAC_LOW+1,MAC_LOW+2,MAC_LOW+3); // get high
part of MAC

              if( MAC_HIGH[0]==xbee802.scannedBrothers[i].SH[0] &&
MAC_LOW[0]==xbee802.scannedBrothers[i].SL[0] &&
                     MAC_HIGH[1]==xbee802.scannedBrothers[i].SH[1] &&
MAC_LOW[1]==xbee802.scannedBrothers[i].SL[1] &&
                     MAC_HIGH[2]==xbee802.scannedBrothers[i].SH[2] &&
MAC_LOW[2]==xbee802.scannedBrothers[i].SL[2] &&
                     MAC_HIGH[3]==xbee802.scannedBrothers[i].SH[3] &&
MAC_LOW[3]==xbee802.scannedBrothers[i].SL[3]    ){

                     result = true;
                     break;
              }
       }

       return result;

}
```

Test results

The two modes we decided to test are *hibernate* and *deep sleep*, we developed a spreadsheet that permits an initial estimation on power consumption, then we will see how much our empirical data corresponds.

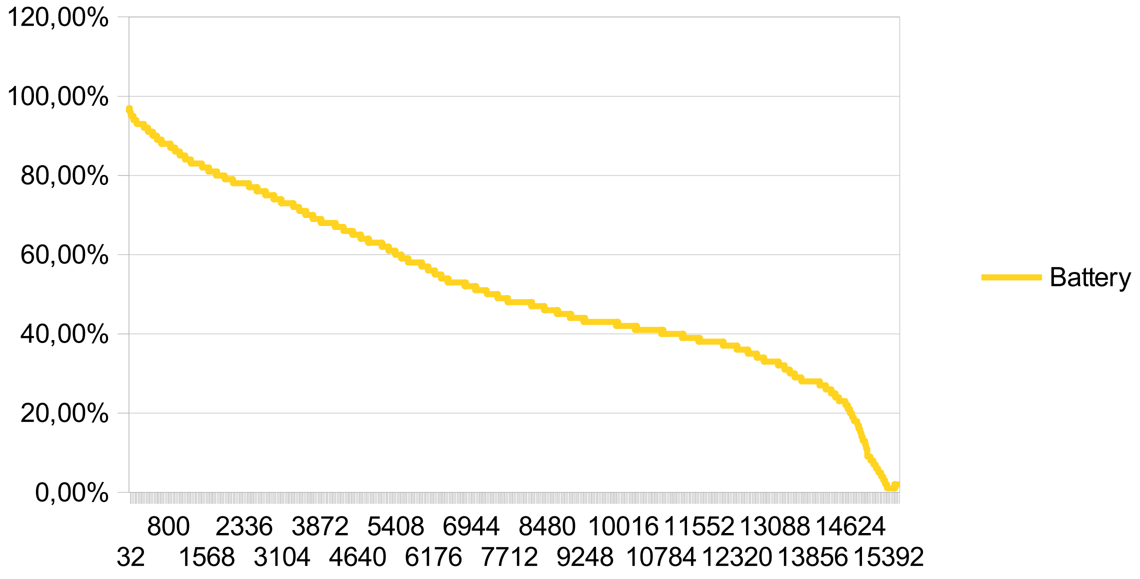
Power consumption

Waspmote	current (mA)		time (ms)	Consumption (mAh)
on	9		3000	97200
hibernate	0,0007		10000	25,2
deep sleep	0,062		0	0

Xbee	current (mA)	Data to transmit (byte)	time (ms)	Consumption (mAh)
TX	150	100	0,0024	1,296
RX/idle	55	//	3000	594000
Transmission Speed (kbpps)	250			

Total Consumption (mAh) (ON)	691201,296	Battery capacity (mAh)	1150
Total Current Draw (mA) 2(ON)	192,0004	Estimated running time (h (continuous)	17,97
Total Consumption (mAh) (hiberı	25,2		
Total Current Draw (mA) (hibern	0,007	Estimated running time (h (hibernation	77,87
Total Consumption (mAh) (sleep	0		
Total Current Draw (mA) (sleep)	0		

Table 7: Estimation of running time (hibernation)



Graph 1: Autonomy Test, hibernation

Using hibernation (set to wake up after 10 seconds) lasted 2 days 8 hours and 35 minutes (the waspmote was on for 3 seconds and in hibernation for 10 seconds)
The difference from our theoretical calculation can be explained in terms of a bigger current draw when initializing the components every time the waspmote wake up.
While the second test, for the deep sleep mode showed that the waspmote can run more than 5 days, this means a lower power consumption than expected (ALL_OFF switch was used).

Conclusions and future developments

Our autonomy test showed that despite on the documentation is written that hibernation can be used for periods higher than 8 seconds (like deep sleep) it should be used only when the hibernation period is really long.

Our conclusion is that the characteristics that could render waspmote a good product should be:
- support for debugging (AVRs can be debugged through JTAG port, not exposed on waspmote, ISP pins are available but not it's not easy to integrate in existing products)
- integration with major IDEs and AVR products (i.e. AVR studio, AVR-eclipse), also to use their debugging capabilities
- clear documentation
- mature API

in fact even if waspmote is an interesting product with characteristics not common to other competing products (Arduino boards don't have power saving modes) it's still not mature enough: problems with the development environment, documentation incomplete and unclear, API still not mature and no way to debug if not through printing messages using the serial port can bring lot of problems that takes too much time and effort to solve.

Bibliography

SUN01: , , , http://www.sunspotworld.com/
LIB01: , , , http://www.libelium.com/squidbee/index.php?title=Waspmote_vs_SquidBee
CHK01: , , , www.cooking-hacks.com/index.php/documentation/tutorials/waspmote
MAX01: Maxim IC, , http://pdfserv.maxim-ic.com/en/ds/DS3231.pdf
LIB02: Libelium, Waspmote Technical Guide,
http://www.libelium.com/documentation/waspmote/waspmote-technical_guide_eng.pdf
DIGI01: Digi.com, , , http://www.digi.com/xbee/
IEEE01: IEEE, 802.15.4-2003, 2003, http://standards.ieee.org/getieee802/download/802.15.4-2003.pdf
IEEE02: , 802.15.4-2003, 2003, http://standards.ieee.org/getieee802/download/802.15.4d-2009.pdf
DIGI02: Digi, Wireless Mesh Networking ZigBee® vs. DigiMesh™, ,
http://www.digi.com/pdf/wp_zigbeevsdigimesh.pdf
UCB01: Helger Lipmaa, Phillip Rogaway, Davide Wagner, Comments to NIST concerning AES Modes of Operations: CTR-Mode Encryption, 2000,
http://www.cs.ucdavis.edu/~rogaway/papers/ctr.pdf
ZEN01: M. Zennaro et al., Planning and Deploying Long Distance Wireless Sensor Networks: The Integration of Simulation and Experimentation 2010