Measuring with ultra-low power Wi-Fi sensor tags in LabVIEW

TOM SAVU
POLITEHNICA University of Bucharest
313 Spl. Independentei, 060042 sector 6, Bucharest
ROMANIA
tom@tomsavu.net http://www.tomsavu.net

MARIUS GHERCIOIU
National Instruments Corp.
11500 N. Mopac Expressway, Austin, TX 78759
U.S.A.
marius.ghercioiu@ni.com http://www.ni.com

Abstract: - G2 Microsystems of Campbell, California, USA, released in 2007 the first ever ultra-low power Wi-Fi System on a Chip (SoC) named G2C501. This SoC includes a 32-bit CPU, crypto accelerator, real-time clock and a versatile sensor interface that can serve as a standalone host subsystem. The G2C501 goes beyond today's basic radio frequency identification (RFID) technology to offer intelligent tracking and sensor capabilities that leverage IEEE 802.11 (Wi-Fi) networks. Due to its support for multiple location technologies, small form factor and ultra-low power consumption, the G2C502 SoC can be integrated into Wi-Fi sensor tags that lower cost of ownership and meet the needs of a variety of industries including consumer electronics, pharmaceuticals, chemical manufacturing, cold chain and more.
A battery powered, small size ultra low-power Wi-Fi wireless measurement node name IP Sensor has been built using the G2C501 SoC. Sensors for measurement of temperature, humidity, light, and vibration or motion are currently mounted on the IP Sensor board. The node is able to read a sensor and send data to the network by using an IP-based application protocol such as UDP.
This paper describes the new IP Sensor device and gives a programming methodology using LabVIEW.

Key-Words: - Wi-Fi sensors, System on a Chip (SoC), ultra-low power, LabVIEW

1 The IP Sensor

The IP Sensor is a Wi-Fi sensor tag that is battery powered and it has been designed around the G2C501 ultra-low power 802.11b 32-bit SoC. The G2C501 SoC was designed for the mobile asset tracking market that is in need of active tags with a battery life time of 1 to 5 years.
The G2C501 SoC (system on a chip) enables the development of ultra low power small form factor tags for asset location, sensing, and tracking. Designed in low cost 0.13um CMOS, it incorporates a full 802.11b PHY and MAC, the LWIP TCP/IP protocol stack, a 32-bit CPU and a hardware encryption engine. Complex power management is fully integrated to achieve maximum lifetime from a wide variety of high-energy density and low cost battery chemistries. Feature rich analog and digital interfaces allow for straightforward connection of sensors and external control. The G2C501 has been designed to be the first truly global integrated solution for MRM (mobile resource management) applications. The ultra low-power SoC solution significantly reduces the TCO (total cost of ownership) by leveraging the ubiquitous IEEE 802.11b/g Wi-Fi installed base. The G2C501 allows for the development of ultra low-power field programmable tags. The high level of integrated functionality on this single-chip solution eliminates the need for almost all external circuitry, resulting in PCBs (printed circuit board) with small footprints and low total BOM (bill of material) cost.
The work leveraged the integrated functionality of the G2C501 to build sensors. IP Sensors (Figure 1) are Wi-Fi tags that also contain sensors. IP Sensors send measurement data to Access Points or Wi-Fi enabled computers.

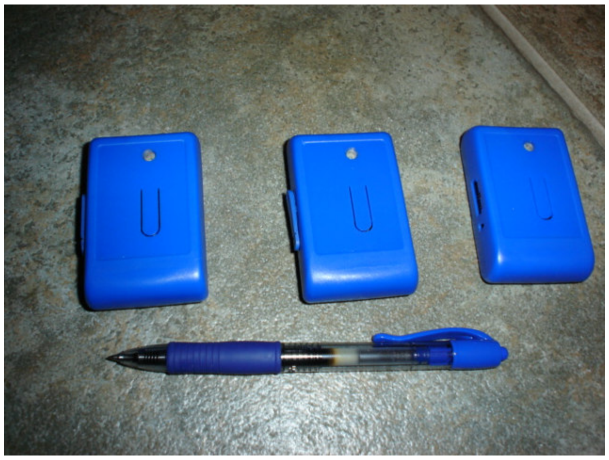


Fig. 1: IP sensors

Access Points will collate measurement data from multiple IP Sensor tags located in their neighborhood (Figure 2) and will rout the information to a data client PC. IP Sensor measurements take place mostly indoors, in places like a warehouse, factory floor, or a building facility that is connected to the Internet and hence will be covered by Access Points. Deployment of standard Access Points for local network communication to PC also works in places that are not Internet connected.

Fig. 2: IP Sensor tags next to a LinkSys Access Point

In a supply chain asset tracking application it is very useful to be able to monitor temperature to avoid overheating, especially for foodstuffs, security seals for containers, shock and acceleration for fragile goods, humidity for food, etc. The SoC used in tags that serve this market has to implement signal conditioning and digitizer elements in order to be capable of performing all these measurements. The IP Sensor is using and extending tag SoC functionality to create a multitude of measurement devices that monitor light, radiation, pressure, movement, humidity, vibrations and other physical or chemical processes. The IP Sensor is an asset tracking tag with a sensor attachment which is interfaced to the SoC using signal lines like SPI, analog-to-digital inputs, DIO and current I/O.

Wi-Fi solutions for sensing and communication are not new. Wi-Fi had been used in computers, PDAs and cellular phones for a long time. These devices though are very power hungry, use rechargeable batteries with a requirement to last for maybe eight hours between consecutive recharge cycles. In contrast with existing Wi-Fi solutions, the IP Sensor's novelty is its ultra low power Wi-Fi capability which makes it suitable for sensing applications where battery power management is critical. Further on this line of thoughts it has to be acknowledged the existence of competitive technologies in very low power sensors using the 802.15.14 (ZigBee) specification, or even Bluetooth. The difference between Wi-Fi IP Sensors and ZigBee sensors is that the IP Sensor application model does not need specially designed Access Points to bring measurement data into the network. ZigBee sensors send data to other sensors and eventually to some network Node that is capable of converting ZigBee into TCP/IP such that finally the data can be delivered to computers via the Internet. IP Sensors talk TCP/IP or UDP and therefore can use off-the-shelf Access Points by that lowering the cost of application. Same is true when compared to Bluetooth sensors, which also use more power.

2 IP Sensor hardware description

The IP Sensor device architecture is designed around an 802.11 32-bit SoC used by asset tracking tags for ultra low power consumption. The device is built on one side and it has two distinct parts, the Core and the Sensor Area. The design contribution is Size and Modularity. It was built the smallest possible PCB size of 42.5 mm x 23.5 mm as seen in the attached component placement picture (Figure 3).

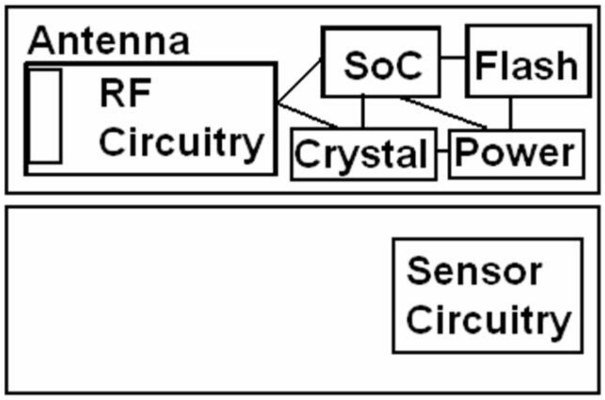


Fig. 3: IP Sensor tag Core and Sensor(s) Attachment

Also, it has been built a modular architecture to accommodate a Core Area that is fixed, and a Sensor Area that what will change depending on type of sensor(s) used in different applications (Figure 4). Modularity allows for reutilization of the same Core Area (and related Firmware) in different applications by simply changing the Sensor Area and by adding sensor drivers into the existing firmware.

The Core Area contains the SoC, a 44.0 MHz clock used by the SoC in high power mode, and another 32.768 KHz clock used for timing of sensor circuitry in low power mode. The Core Area also has:

- 1Mb EEPROM, expandable to 4Mb, used as application programming space,
- Power section that generates +3.3V and +1.4V for the Core from a battery input circuitry with detection for re-versed polarity.
- RF circuitry that includes a power amplifier and the 2.4 GHz antenna.

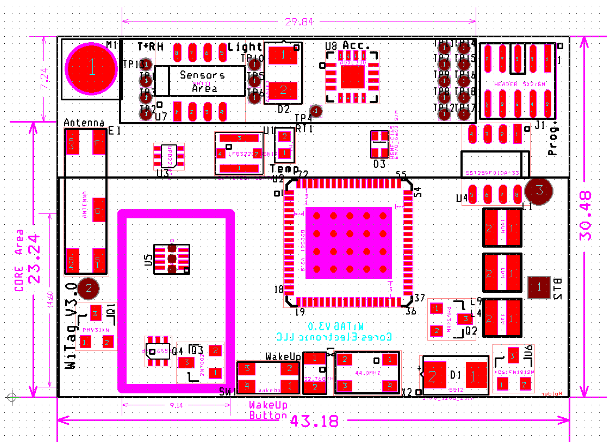


Fig. 4: IP Sensor: Component Placement

The Sensor Area can be populated with sensors that connect to the sensor bus of the SoC (Figure 5). The DEMO version is populated with circuitry for the SHT11 temperature and humidity sensor and also for the ADXL330 vibration / motion / tilt sensor from Analog Devices.

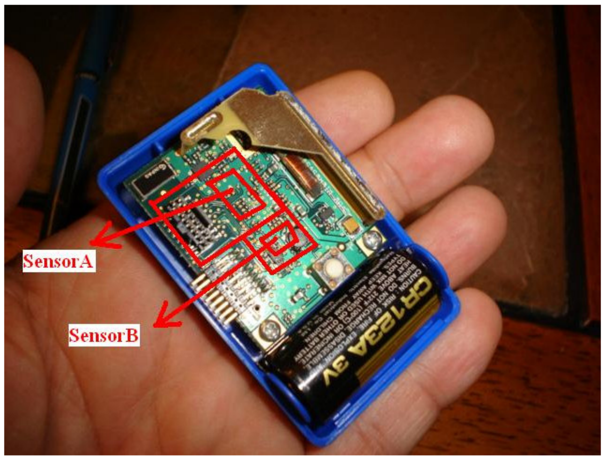


Fig. 5: IP Sensor tag Core and Sensor(s) Attachment

3 Power consumption and power management

The IP Sensor is powered by a 3.2V, lithium battery, model CR123A (Figure 6).

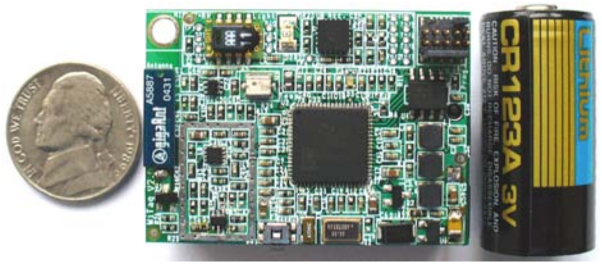


Fig. 6: IP Sensor mounted on battery holder

The IP Sensor has two modes of operation, sleep and wake-up modes.

Sleep is the low-power mode in which the IP Sensor will use an average of 100 µW of battery power. This amount of power is used by the SoC to perform sensor circuitry current sourcing, sensor readings and access of a small non-volatile memory location. uC OS does not run, communication does not run, everything regarding SoC intelligence is shut down in sleep mode.

An IP Sensor hardware timer which is pre-programmed will trigger the SoC to wake-up and enter a high-power mode state for very short intervals of time at fixed time intervals. Other causes of transition from sleep to wake are preconfigured threshold values 'seen' by the motion, temperature, magnetic and other sensors.

Each wake period the IP Sensor uses power for reception and transmission of data. The CR123 battery gives a total of 1550 mAh. At 8760 hours in a year, the total available current is 176.94 µAh. The node has a reception time of 90 msec and a transmission time of 4.7 msec for a total wake period of 94.7 msec.

Power consumption in reception (R) and rransmission (T) is given by the following formulas:

$$I_R = \frac{60mA \cdot 90m\sec}{3600\sec/h} = 1.5\mu Ah \qquad (1)$$

$$I_T = \frac{700mA \cdot 4.7m\sec}{3600\sec/h} = 0.914\mu Ah \qquad (2)$$

The report between wake and sleep is calculated at:

$$N = \frac{176.94\mu Ah}{1.5\mu Ah + 0.914\mu Ah} = 73.3 \qquad (3)$$

Based on this computation, the following power consumption scenarios may be given:

- If the IP Sensor has 73 wake periods of 95 msec every hour (equivalent with one wake period every 49 seconds), then the CR123 battery that gives 1550 mAh will last for 1 year.
- If the IP Sensor has one wake period of 95 msec every 4.11 minutes, then the CR123 battery that gives 1550 mAh will last for 5 years.
- A sensor node lifetime of 1 week to 5 years will be achieved based on selection for duration of the wake period, frequency of wake periods, and power budget.

In wake or high power mode the IP Sensor starts eCos OS on the 32-bit µC and enables full intelligence including communication capabilities using UDP.

The amount, length, and complexity of tasks performed during high-power periods vary depending on the application. From short term periods of 1ms to 10 seconds during which the IP Sensor consumes about 100mW average while it makes measurements, receives 802.11 data and executes code on the CPU, to very short periods of say 500us where the IP Sensor consumes in the range of 1W while it transmits 802.11b data via UDP. It should be mentioned here that the 802.11b communication capability of the IP Sensor operates autonomously from the CPU and it can execute direct reads/writes data to system RAM in low-power mode.

During a lifetime period of 1 month to 5 years the IP Sensor may repeat the task of each waking up for high-power mode event every 10 to 300seconds. IP Sensor power consumption is dictated by battery capacity and the frequency and length of the wake or high-power mode states of the application.

There is no explicit step for node assignment to an Access Point in an application that uses Wi-Fi sensors. The sensor registers with an Access Point and uses the DHCP protocol to request dynamic allocation of an IP address from a DHCP server that may reside in the Access Point or in a networked PC. If the IP Sensor does not have access to a wireless network, it could store sensor data, complete with timestamp. Measurement data will be uploaded to a PC or Access Point Data Server when Wi-Fi access becomes available again.

4 Firmware

The IP Sensor contains a 32-bit processor (or CPU) that runs on a 44MHz external clock. The slower and lower power sensor measurement operation is controlled by an external 32 KHz oscillator. The CPU incorporates full 802.11 PHY, MAC and encryption engine. The IP Sensor CPU has internal 80Kbytes of RAM and 320Kbytes of ROM that comes loaded with the following firmware components:

- Boot loader
- eCos OS
- TCP/IP stack
- 802.11b stack
- Encryption and decryption support
- Application/deployment specific sensor drivers and communication protocol
- Power-saving support

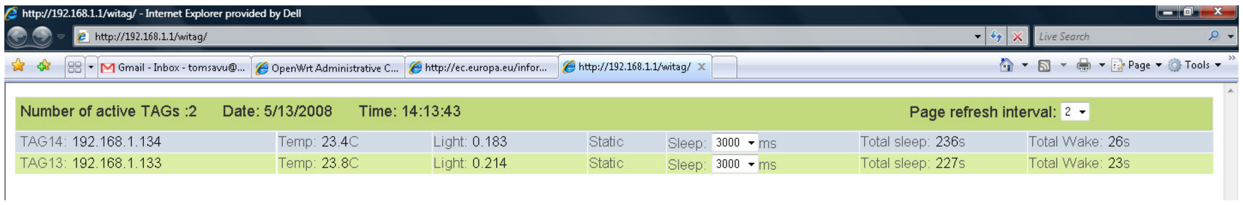
802.11b communication capability of the IP Sensor operates autonomously from the CPU and it can execute direct reads/writes data to the system RAM. All this is part of the node firmware. Besides this infrastructure related firmware, there is more firmware that is related to user applications.

User code is downloaded in external Flash, the 1Mb EEPROM memory that is controlled by the CPU using an SPI interface. An example of user code is the I2C driver imple-menting communication with the SHT11 sensor, and a time loop that wakes-up the sensor every User Defined number of seconds. After each sleep period of time, the sensor node wakes up timer triggered, and communicates with the AP to find out if there is any data in its buffer that needs to be red. If there is none, then the node will perform a temperature and humidity SHT11 sensor read and it will send the data to the (client) IP address via the AP. All this is described by the user application firmware that is stored in the 1Mb EEPROM memory location.

5 Data Server Use Case

The use case scenario that makes most sense has several IP Sensor measurement nodes associated with an AP Data Server to which the nodes send information on a periodic basis and from which the nodes may receive commands. The AP Data Server collates measurement data from these multiple sensor tags and presents them as a web page to a Data Client PC user or users. The following demo could be imagined:

- User opens a Web Browser on the Data Client PC,
- User browses to the IP address of the AP Data Server,
- Data Server sends Data Client PC a web page containing controls that allow the user to configure each IP Sensor for measurement related parameters including alarms to parameters that are monitored and frequency of data transmission to the AP.
- User sets measurement, alarm and transmission parameters using controls displayed by the web page,
- Data from IP Sensors appears in indicators and graphs displayed by the web page panel (Figure 7).


http://192.168.1.1/witag/ - Internet Explorer provided by Dell
http://192.168.1.1/witag/
Live Search
Gmail - Inbox - tomsavu@...
OpenWrt Administrative C...
http://ec.europa.eu/infor...
http://192.168.1.1/witag/
Page
Tools

Number of active TAGs :2 Date: 5/13/2008 Time: 14:13:43 Page refresh interval: 2

TAG14: 192.168.1.134	Temp: 23.4C	Light: 0.183	Static	Sleep: 3000 ms	Total sleep: 236s	Total Wake: 26s
TAG13: 192.168.1.133	Temp: 23.8C	Light: 0.214	Static	Sleep: 3000 ms	Total sleep: 227s	Total Wake: 23s

Fig. 7: IP Sensors Data in the webpage panel

During the tests described above there were used two IP Sensors having three transducers each:

- a temperature sensor;
- a light intensity sensor;
- an on / off motion sensor.

6 LabVIEW Tools

Many users in the test and measurement industry are programming their computer based measurement applications from LabVIEW. Wireless sensor networks are seen by these users as extensions of existing measurement systems based on plug-in PC boards, PXI, cRIO chassis, etc., which are all LabVIEW platforms. LabVIEW users will program the IP Sensor wireless measurement application from LabVIEW by calling UDP Read/Write VIs.

An IP Sensor sends UDP messages every three seconds to IP address 192.168.1.1, port 50007.

Each message contains the current state of the tag sensors and is always 28 bytes long, seven int32 values using big endian ordering.

The seven int32 values are the following

0 - the fixed value 1

1 - sleep period in milliseconds(initially 3000)

2 - total seconds spent powered down

3 - total seconds spent powered up

4 - temperature in Deg C * 1000

5 - light sensor voltage value * 1000 (range 0-599, corresponding to 0 to 0.599 V)

6 - 0 = no motion, 1 = in motion

To change the sleep period of an IP Sensor (currently default setting is 3 seconds), a message needs to be sent to the address it sent the message from (IP, UDP port). The message must contain three integer values in big endian encoding:

0 - 0xABCDEF00 (code to denote write message)

1 - new sleep period value in milliseconds

2 - 1

The diagram of a simple LabVIEW Virtual Instrument (VI) used for reading the IP Sensor message is presented in Figure 8.

The VI is designed to return only the message sent from a specified address (from a specified IP Sensor). Messages sent from other addresses will be ignored.

The VI is first using an UDP Open function for opening an UDP socket on port 50007.

The function creates a network connection refnum that uniquely identifies the UDP socket and which is subsequently used by an UDP Read function.

The UDP Read function returns the address of the IP Sensor which wrote the datagram in the UDP socket.

The specified IP address from where the message is allowed to be read is converted to a net address using the String To IP function and then compared with the address returned by the UDP Read function.

If the two addresses are not the same, then the current iteration of the VI's While loop is not going to be the last iteration, which means that the message read in this iteration will be ignored.

The network connection refnum is further used by an UDP Close function for closing the UDP socket. If the UDP Close function returns an error, the VI's While loop will continue to the next iteration and so the current read message will be ignored.

The message returned by the UDP Read function, in String format, is converted to an array of int32 integers in big-endian, network order, using an Unflatten From String function. It has to be specified, using a boolean False constant, that data does not include array or string size.

If the two above mentioned IP addresses are the same, this array is going to be further sent through the data flow. If the addresses are different, then the array will be replaced, using a Select function, by an array containing seven NaN (null) values. The second array is created by an Initialize Array function.

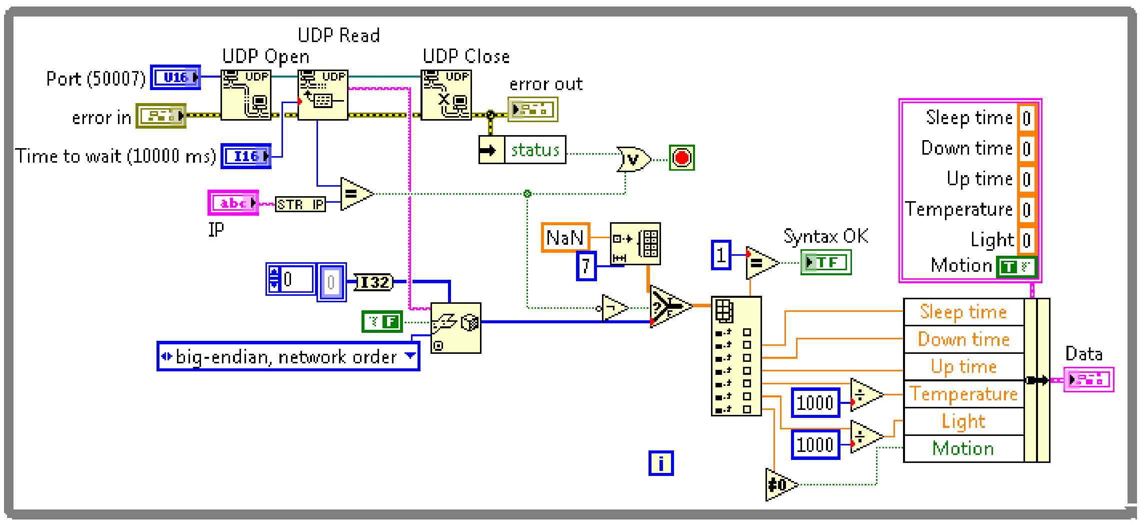


Fig. 8: LabVIEW VI for reading IP Sensor's message

The seven values from the data array are separated using an Index Array function.
The first value is compared with 1 for verifying the message syntax. The result of the comparison is passed as a VI's boolean output value.
Values representing temperature and light sensor voltage are scaled by dividing them with 1000.
0 or 1 value describing the motion state is converted to a boolean value using a Not Equal To 0? function.
Six of the values are grouped into a named cluster, using a Bundle By Name function. The data structure is passed to the function using a cluster constant. The cluster containing the six grouped values is passed as the VI's second output value.
For testing the response time of the above described VI, two Tick Count functions (returning the value of the millisecond timer) were placed in its diagram (Figure 9).
The first function is registering the time when the While loop starts.
The second function was placed inside the While loop, in the True window of a Case structure, so it's registering the time when the While loop ends.
The whole VI's diagram was placed in a For loop executing ten iterations. After each iteration of the For loop, the execution time of the While loop and its number of performed iterations are stored in a cluster, using a Bundle function.
The clusters resulted from all the ten iterations of the For loop are passed as an array to an indicator on the VI's control panel.

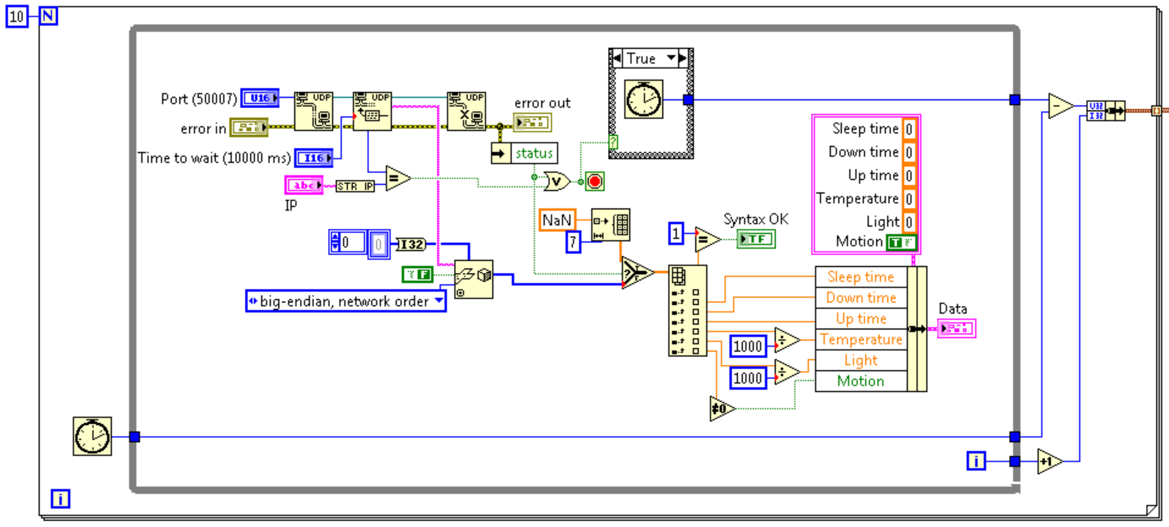
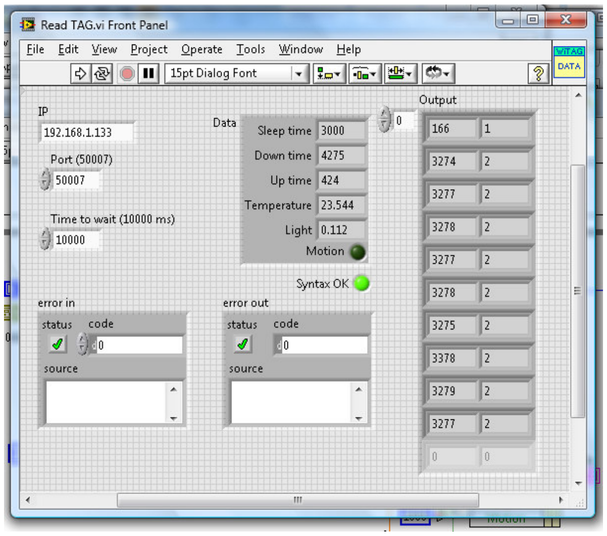


Fig. 9: Modified diagram for testing purposes

Data obtained during testing one IP Sensor show that the VI's response time is usually no longer than 110 % of the sensor's sleeping period (Figure 10, Table 1).


Read TAG.vi Front Panel

IP
192.168.1.133
Port (50007)
50007
Time to wait (10000 ms)
10000

Data
Sleep time 3000
Down time 4275
Up time 424
Temperature 23.544
Light 0.112
Motion
Syntax OK

error in
status code
0
source

error out
status code
0
source

Output
0

```
166   1
3274  2
3277  2
3278  2
3277  2
3278  2
3275  2
3378  2
3279  2
3277  2
0     0
```

Fig. 10: Test data

Table 1

Test no.	Time [ms]	Attempts
1	166	1
2	3274	2
3	3277	2
4	3278	2
5	3277	2
6	3278	2
7	3275	2
8	3378	2
9	3279	2
10	3277	2

Another LabVIEW VI developed for the IP Sensors is one to be used for discovering which sensors are in the communication range (Figure 11).
The VI's logic is assuming that, at each iteration of the While loop, when the UDP socket is opened, it is not known which IP Sensor will communicate.

Only the sensor's address is extracted from the UDP Read function, and is transformed in string format using the IP To String function with dot notation.
A string array, initially void, is sent from one iteration to another using a loop's shift register. This array will finally contain the IPs of the discovered sensors.
If the UDP socket is closed without errors, then the sensor's address is searched in the solutions array using a Search 1D Array function.
The function is returning an unsigned index if the address was found or a negative value otherwise.
If the address wasn't found, then it is added to the solution array using a Build Array function (in the True window of the Case structure, Figure 12). If the address was found, then a number of attempts is incremented.
The While loop stops either if an error was returned or if the number of possible attempts was reached.
The solution array is then sorted and send as a VI's output value.
A front panel image of the VI discovering two IP sensors is presented in Figure 13.

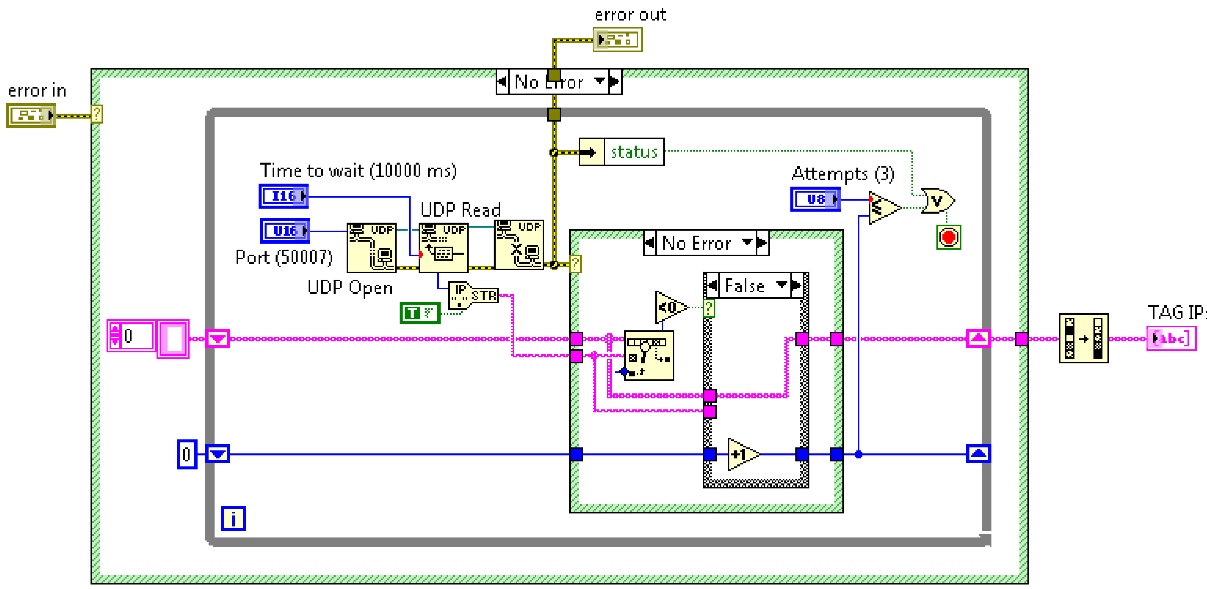


Fig. 11: Diagram for the discovering sensors VI

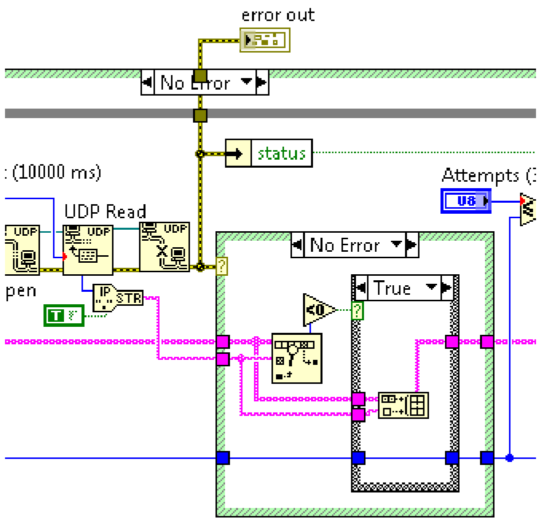
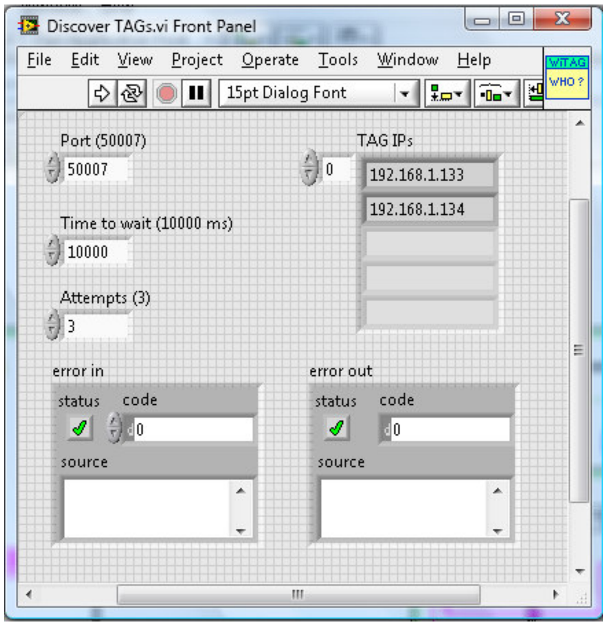


Fig.12: Partial diagram for not found address


Discover TAGs.vi Front Panel
File Edit View Project Operate Tools Window Help
15pt Dialog Font

Port (50007)
50007
Time to wait (10000 ms)
10000
Attempts (3)
3
TAG IPs
0
192.168.1.133
192.168.1.134
error in
status ✓ code 0
source
error out
status ✓ code 0
source

Fig. 13: Results of sensors discovering VI

For testing the response time of the sensors discovering VI, two Tick Count functions (returning the value of the millisecond timer) were placed in its diagram (Figure 14).
The first function is registering the time when the While loop starts. The second function was placed inside the While loop, so it's registering the time when the While loop ends.
The whole VI's diagram was placed in a For loop executing ten iterations. After each iteration of the For loop, the execution time of the While loop and the number of attempts are stored in a cluster, using a Bundle function.
The clusters resulted from all the ten iterations of the For loop are passed as an array to an indicator on the VI's control panel.

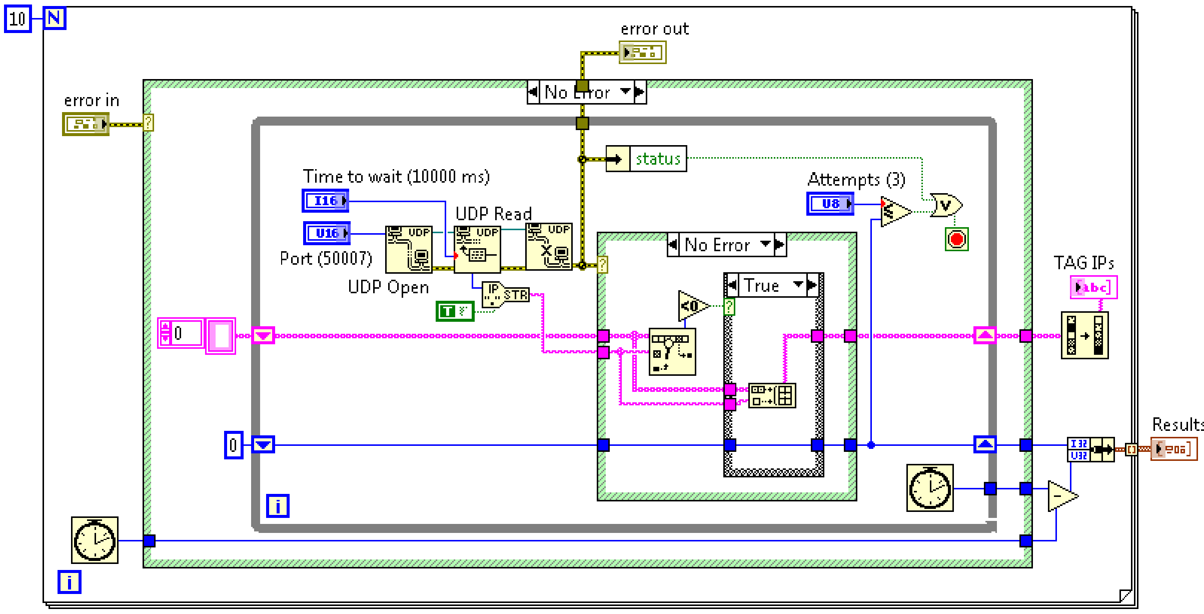


Fig. 14: Test VI for sensors discovering

If an error occurs during an iteration of the For loop, then the number of attempts is set to zero (Figure 15).

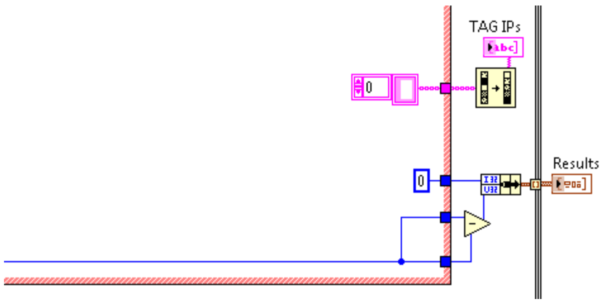
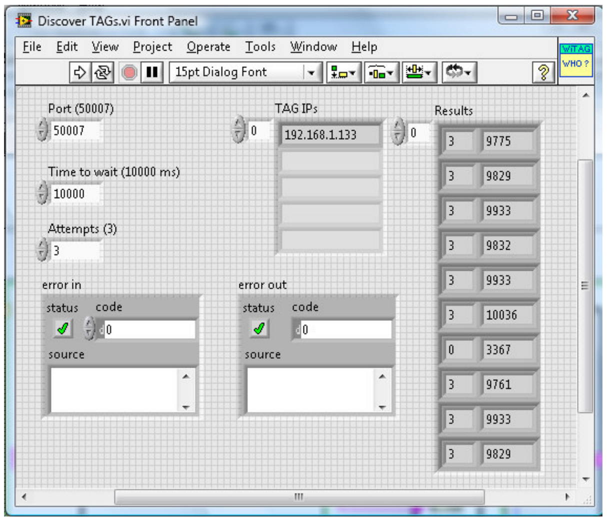


Fig. 15: Setting the number of attempts to zero because of an error

Test results for a situation when only one IP sensor was in the range are presented in Figure 16 and Table 2. Test results for a situation when two IP sensors were in the range are presented in Figure 17 and Table 3.
Not so obvious at a first look, the time needed for discovering two sensors is smaller than the time spent for the discovery of only one sensor.
Neglecting the 7th iteration from the first case, each time three attempts were made, which means that, in each iteration of the For loop, the VI had to find three times an already existing sensor.
When only one sensor is in range, the time for finding it is approximately 10 % bigger than the sleep period of 3 s, as in Table 1.
When two sensors are in range, because their sleep periods do not start in the same time, the time for finding one of the two sensors decreases.


Discover TAGs.vi Front Panel
File Edit View Project Operate Tools Window Help
15pt Dialog Font
WiTAG WHO ?
Port (50007)
50007

Time to wait (10000 ms)
10000

Attempts (3)
3

TAG IPs
0
192.168.1.133

error in
status ✓ code 0
source

error out
status ✓ code 0
source

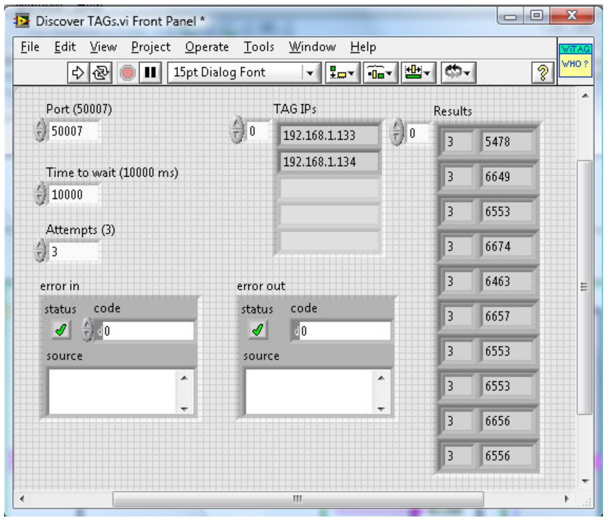
Results
0

3	9775
3	9829
3	9933
3	9832
3	9933
3	10036
0	3367
3	9761
3	9933
3	9829

Fig. 16: Test results for one sensor in the range

Table 2

Test no.	Time [ms]	Attempts
1	9775	3
2	9829	3
3	9933	3
4	9832	3
5	9933	3
6	10036	3
7	3367	0
8	9761	3
9	9933	3
10	9829	3


Discover TAGs.vi Front Panel *

File Edit View Project Operate Tools Window Help

15pt Dialog Font

Port (50007): 50007

Time to wait (10000 ms): 10000

Attempts (3): 3

TAG IPs: 0 | 192.168.1.133, 192.168.1.134

error in: status ✓ code 0 source

error out: status ✓ code 0 source

Results: 0

3	5478
3	6649
3	6553
3	6674
3	6463
3	6657
3	6553
3	6553
3	6656
3	6556

Fig. 17: Test results for two sensors in the range

Table 3

Test no.	Time [ms]	Attempts
1	5478	3
2	6649	3
3	6553	3
4	6674	3
5	6463	3
6	6657	3
7	6553	3
8	6553	3
9	6656	3
10	6556	3

The last LabVIEW tool to be presented is a VI for setting the sleep time for a specified sensor (Figure 18). After opening the UDP connection on the usual port and building the necessary command string (using the Flatten To String function, with big-endian network order, the UDP Write function is used ten times, in a For loop, for achieving a high probability that the command string will be read by the sensor during it's wake time.

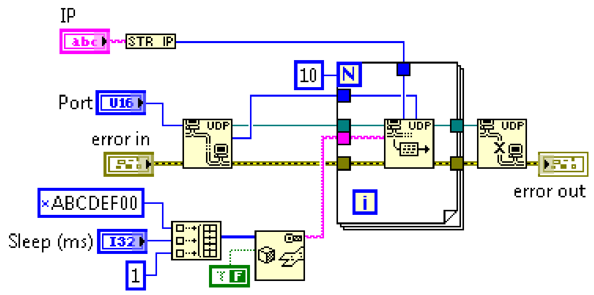


Fig. 18: Setting sensor's sleep time

7 Application example

The diagram of an example application VI is presented in Figure 19.

The sensor discovery VI is first used for obtaining an array containing the IPs of the sensors in range. For each IP in the array, which means for each sensor in range, the data reading VI is used to obtain the measured values of temperature and light intensity and also the boolean value of the sensors motion state. The boolean value is then transformed into 0 or 1 values.

The above mentioned values are grouped and sent to a waveform chart indicator in the application's front panel.

Results obtained during a test sessions are graphically displayed in Figure 20.

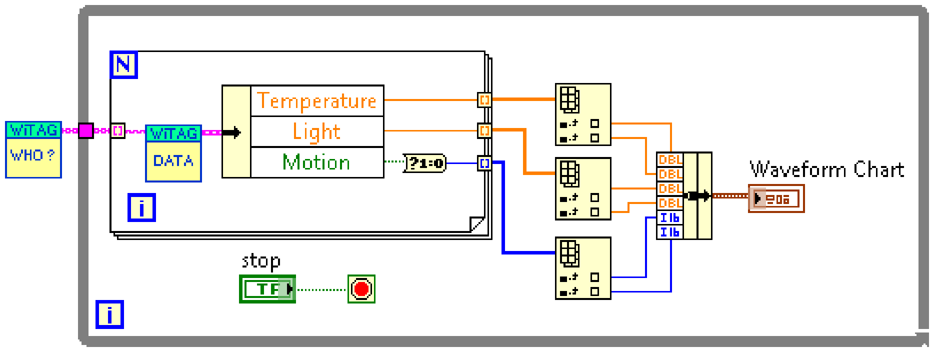


Fig. 19: Diagram of an example application VI

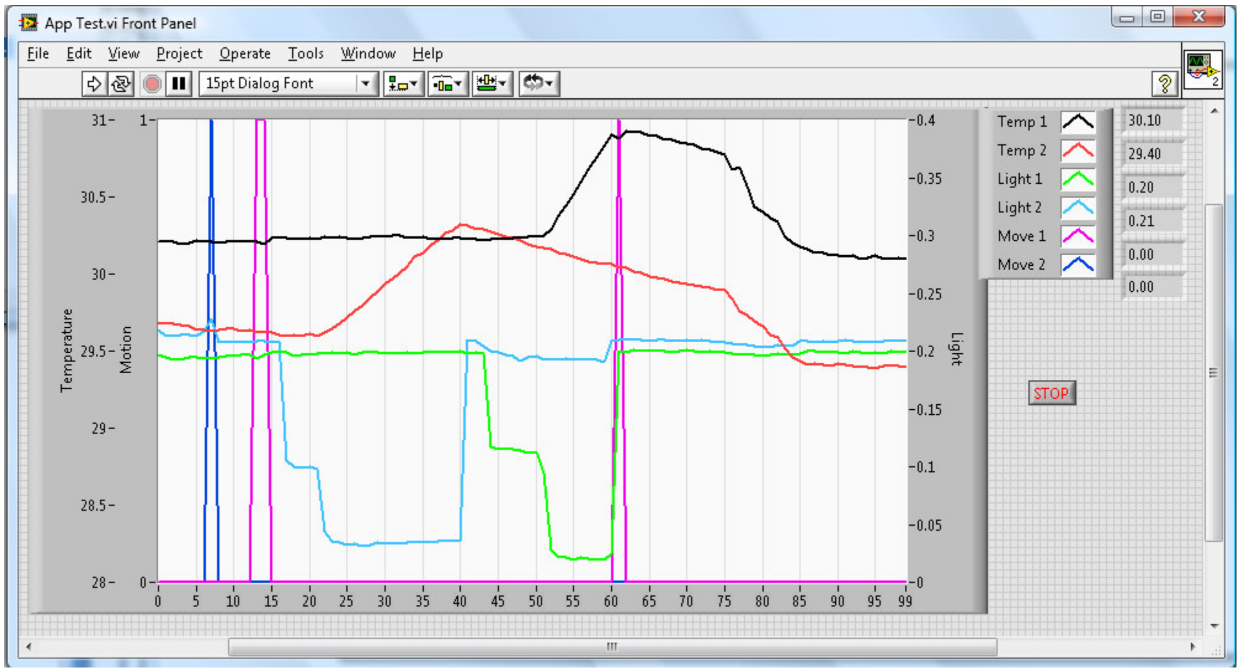


Fig. 20: Results during a test session of the application example

The second sensor was moved at iteration 6, so a value of 1 for its motion state was obtained (dark blue line). First sensor was also moved at iteration 12 (violet line).
Starting with iteration 16, the second sensor was shadowed, so an abrupt decrease in its light intensity signal can be observed on the light blue line.
When the operator's hand was placed over the second sensor at iteration 21, covering it, both a decrease in its light intensity signal and an increasing slope of the temperature signal (red line) may be observed.
At iteration 40 the second sensor was uncovered, so its light intensity signal immediately went back to the original value and the temperature signal begin to decrease.
The same procedure was then applied to the first sensor, shadowing it at iteration 42 (decrease in light intensity, green line), completely covering it at iteration 50 (less light intensity and an increase in temperature due to operator's hand, black line) and uncovering it at iteration 60 (light intensity immediately at initial value and temperature begining to decrease).
When uncovering the first sensor at iteration 60, a little shock was applied to it by mistake and it was registered on the corresponding violet line.
At iteration 75, air started to be blown over the two sensors, so it can be observed a modification in the decreasing speed of the temperature because the two sensors were cooling faster.
At iteration 85 the air flow was stopped and the speed of the temperature variation returned to normal.

8 Conclusion

The new technologies embedded in the IP Sensors make them suitable for a wide range of applications where the low-power consumption and the wireless communication are compulsory.
Developing LabVIEW applications, both for testing the IP Sensors' characteristics and for providing usefull subVIs for further developments will be continued.

References:

[1] Weilian Su, Bassam Almaharmeh, "QoS Integration of the Internet and Wireless Sensor Networks", WSEAS Transactions on Computers, Issue 4, Volume 7, April 2008
[2] J. Levendovszky, A. Bojárszky, B. Karlócai, A. Oláh, "Energy Balancing by Combinatorial Optimization for Wireless Sensor Networks", WSEAS Transactions on Communications, Issue 2, Volume 7, February 2008
[3] Kuang Xing-Hong, Shao Hui-He, " Localization Assisted by the Mobile Nodes in Wireless Sensor Networks", WSEAS Transactions on Communications, Issue 8, Volume 6, August 2007
[4] Marius Ghercioiu, Silviu Folea, Jani Monoses, "The IP Sensor - a WiFi Sensor TAG", ICOMP-07, Las Vegas, Nevada, USA, June 25-28, 2007
[5] G2 Microsystems, "G2C501 for Ultra Low-power Wi-Fi", http://www.g2microsystems.com/ as on June 2008
[6] http://www.tag4m.com/ as on June 2008