NanoQplus 2 (Nano OS)
User's Guide

2007. 11. 23

Sensor Network OS Research Team

Embedded S/W Division

ETRI 한국전자통신연구원
Electronics and Telecommunications Research Institute

Contents

List of Figures

1. Supporting Hardware Platforms

1.1. ETRI-SSN (or NANO-24)

Sensor nodes should be designed, considering several factors that involve the sensor network environments. In addition, they have some constraints and limited capabilities. For example, they are small in size, and their hardware cost and power consumption must be kept low. In most sensor hardware platforms, they are battery-operated, the processor clock speeds are slow (1-16MHz), and have a small amount of memory (2-10KB) without memory protected hardware devices such as MMU (Memory Management Unit).

Figure 1 depicts the main module of ETRI-SSN (or Nano-24). ETRI-SSN and Nano-24 are almost identical except pin-connections and additional external memory. The modules of ETRI-SSN (or Nano-24) are small in size (AA-battery size in height). This module has an Atmega128L CPU (one of AVR product series from ATMEL) with 4KB of SRAM, 128KB of FLASH memory and 4KB of EEPROM, and CC2420 RF module (from Chipcon) for wireless communication.

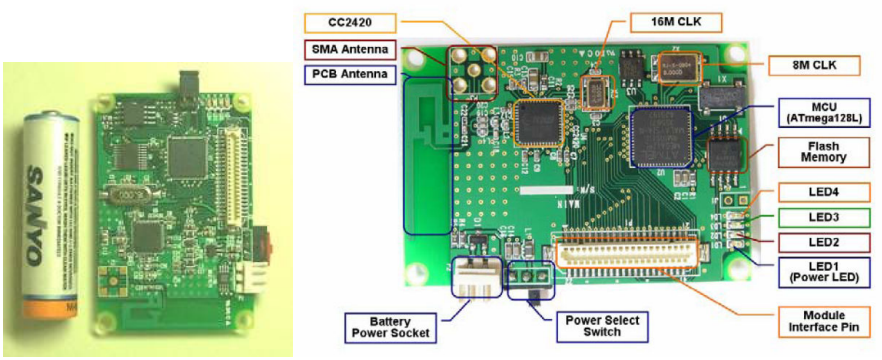


Figure 1. Main module of ETRI-SSN

Component	Model	Description
Sensor Board	ETRI-SSN (or Nano-24)	WSN Hardware Platform
CPU	Atmega128L	- Low-power - 8 bit micro-processor - 8MHz clock
Memory	FLASH	Internal 128KB External 512KB
	SRAM	Internal 4KB External 32KB
	EEPROM	4KB

RF	Chipcon CC2420	2.4GHz Zigbee 250Kbps data rate
USB	RS-232	1Mbps
	RS-485	3Mbps
	TTL	3Mbps

ETRI-SSN (Nano-24) has 4 types of modules; 1) main module (computation and wireless communication), 2) base interface module (connection with PC and main module), 3) sensor module (data collection), and 4) actuator module (actuating an external devices). The combination of those modules is referred to as a *sensor node*. The ETRI-SSN (smart sensor node) is shown in Figure 2.

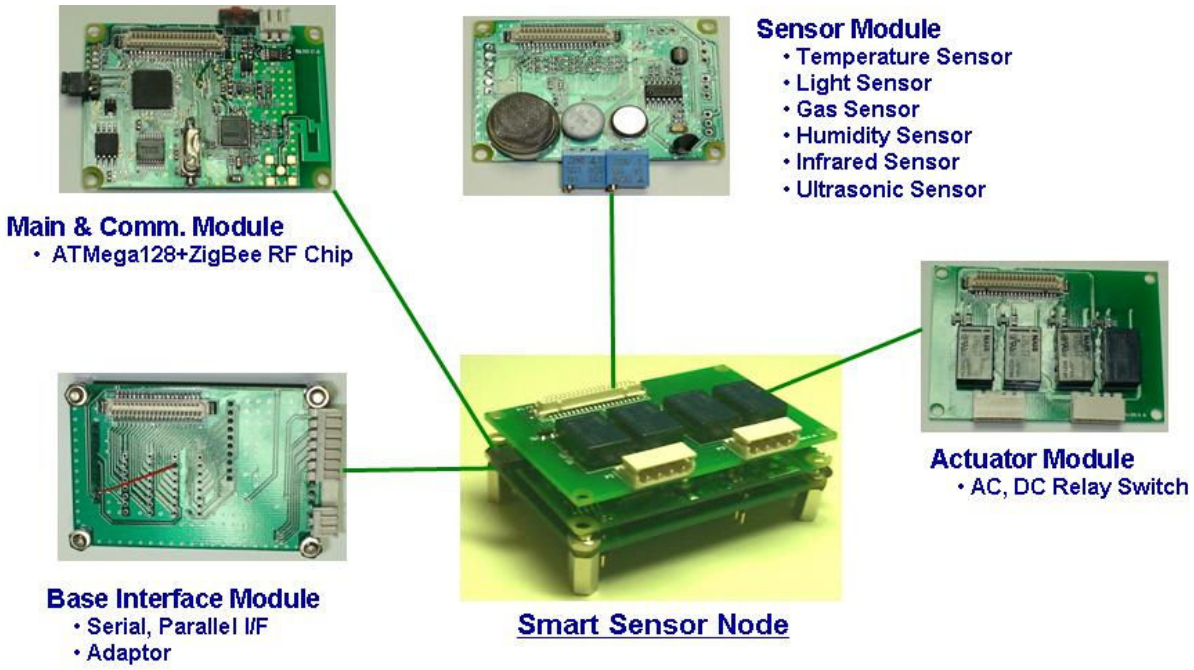


Figure 2. ETRI-SSN components

1) Main module : the core of the sensor node. It has the main CPU and RF chips for computation and communication. All sensor nodes must have the main module.

2) Base interface module : the connection module between PC and the main module. The base interface module can be used to transfer a new application program image from a PC to the main module. This is referred to as “cross development environment”. It also provides the serial communication interface between PC and the sensor node.

3) Sensor module : the sensing module that collects environmental data. Some detected data are stored in the memory of the main module or transferred to the PC through wireless

communication. Sensor modules in ETRI-SSN are humidity sensor, light sensor, gas sensor, temperature sensor, ultra-sonic sensor, and infrared sensor.

4) Actuator module : the actuating module that signals external devices for the application purpose. For example, when an event occurs in a sensor environment, the actuator module may send a command to an external device.

These modules being combined, ETRI-SSN node becomes one of the following nodes; sensor node (sensor module + main module), routing node (main module), sink node (base interface module + main module), or actuator node (actuator module + main module). An example scenario of a sensor network application using these ETRI-SSN nodes is shown in Figure 3. In this scenario, the sensor node sends the sensor module's data to the sink node through wireless communication. Several routing nodes may exist between a sensor node and a sink node. The sink node receives the sensor node data, and sends to the PC connected to the end user or the actuator node. The actuator node analyzes the transmitted data for application purposes (e.g. turning on a fan or controling a lamp).

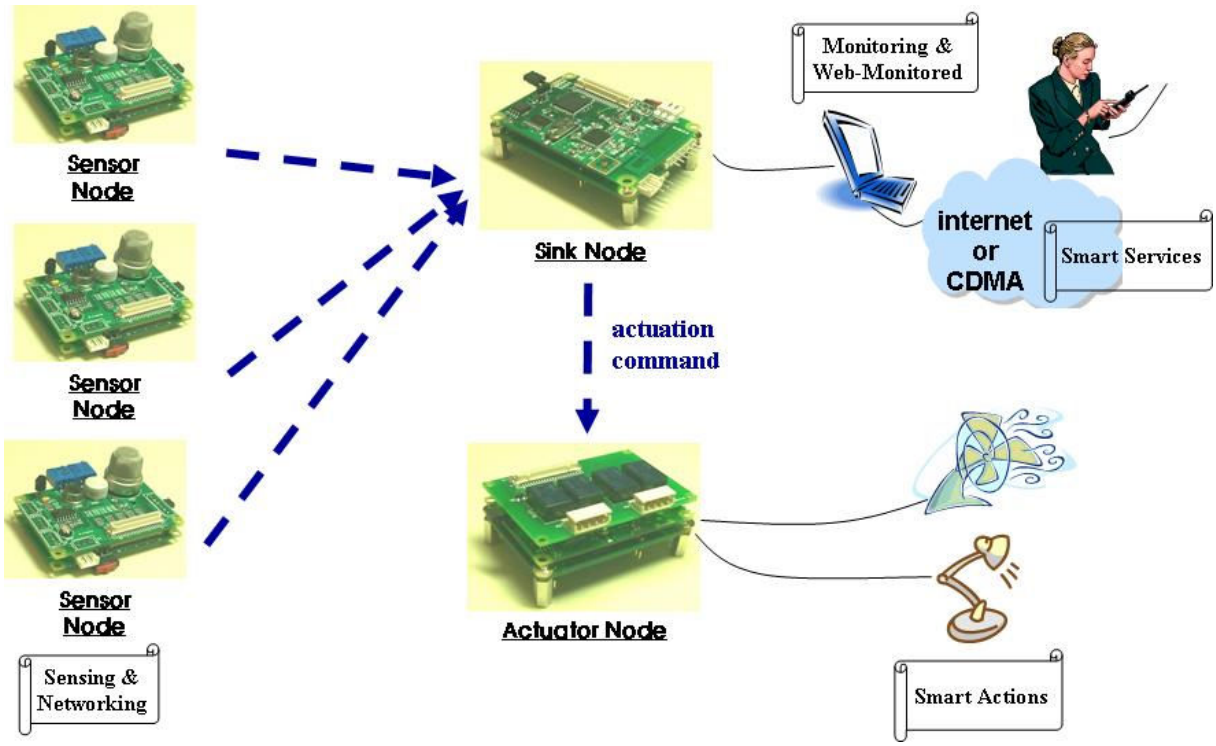


Figure 3. example of sensor network application with ETRI-SSN nodes

1.2. MICAz

The MICAz is a 2.4 GHz, IEEE 802.15.4 compliant, Mote module used for enabling low-power, wireless, sensor networks. The MICAz Mote features several new capabilities that enhance the overall functionality of Crossbow's MICA family of wireless sensor networking products. The MICAz and its interface board are shown in Fig.4.

Figure 4 MICA-Z Sensor Board and Interface

Component	Model	Description
Sensor Board	MICA-Z	WSN Hardware Platform
CPU	Atmega128L	- Low-power - 8 bit micro-processor - 8MHz clock
Memory	FLASH	Internal 128KB External 512KB
	SRAM	Internal 4KB External 32KB
	EEPROM	4KB
RF	Chipcon CC2420	2.4GHz Zigbee 250Kbps data rate
USB	RS-232	1Mbps
	RS-485	3Mbps
	TTL	3Mbps

MICAz has the following characteristics.

• IEEE 802.15.4, Tiny, Wireless Measurement System

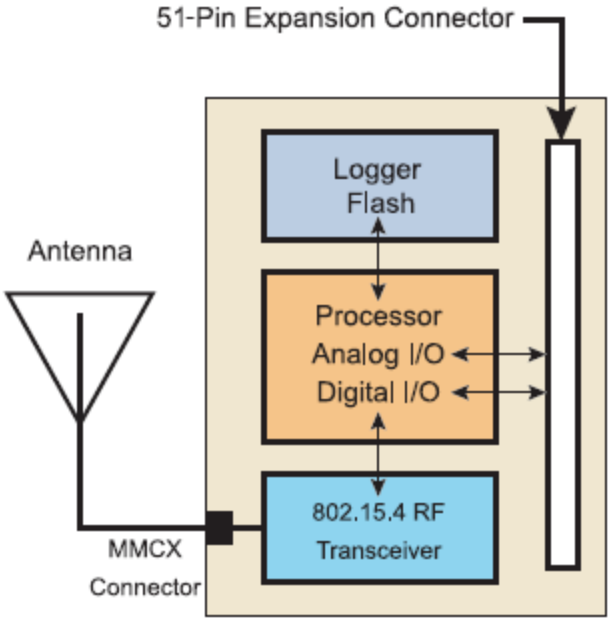
• Designed Specifically for Deeply Embedded Sensor Networks

• 250 kbps, High Data Rate Radio

• Wireless Communications with Every Node as Router Capability

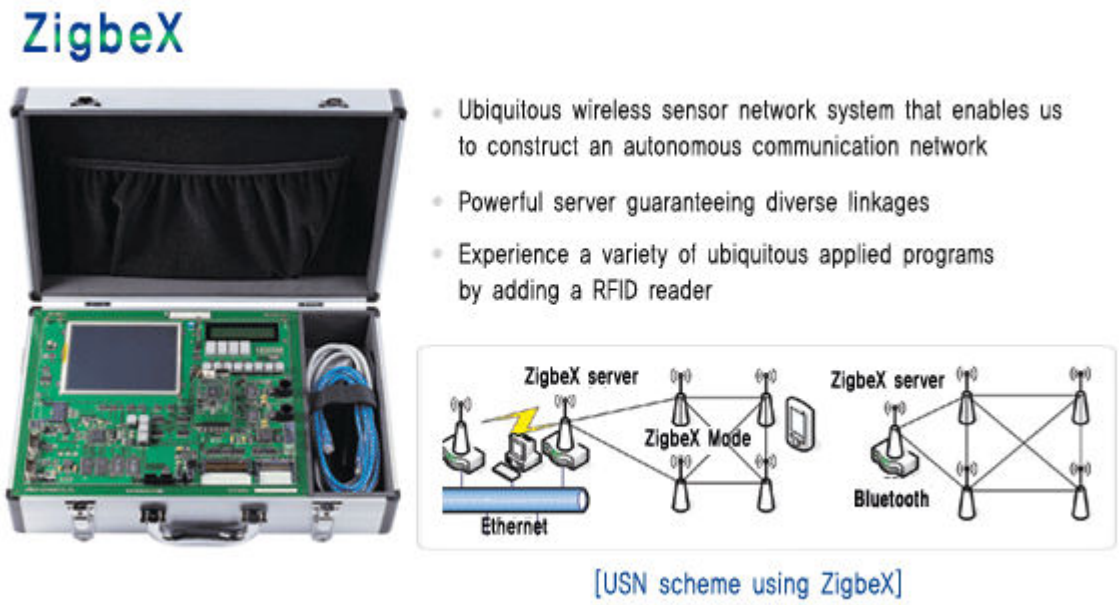
• Expansion Connector for Light, Temperature, RH, Barometric Pressure, Acceleration/Seismic, Acoustic, Magnetic and other Crossbow Sensor Boards

The block diagram of MICAz (MPR2400CA) shown below.

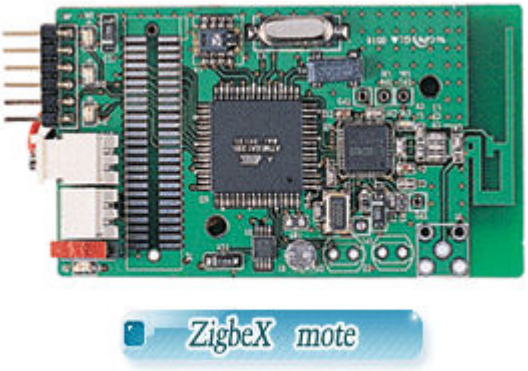


1.3. ZigbeX

The sensor network combining sensor system, embedded system and RF system is used wirelessly in connection with the existing network and is recognized as a leading system that will provide a solution suitable for the ubiquitous age. Therefore, in the educational field, the demand for the educational board of the wireless sensor network is on the increase. ZigbeX of Hanback Electronics Co., Ltd provides diverse linkage and storage devices by loading X-scale CPU of recognized powerful performance. Also, it provides an authentic wireless sensor network that connects the sensor information on the Zigbee supporting mote to a farther server by constructing an autonomous communication network. Besides, it can be connected to any sensor or actuator as it is designed in consideration of the capacity of sensors



[USN scheme using ZigbeX]



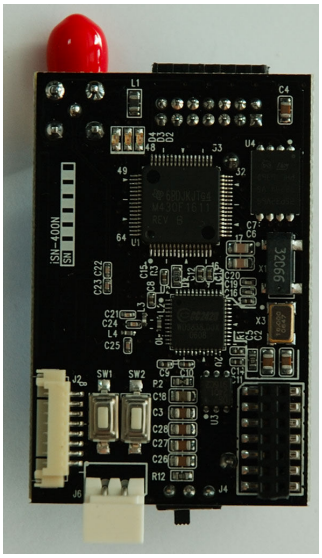
ZigbeX has the following chief features.

. Use 400MHz X-scale CPU with powerful performance
. Stable high performance platform (used as an embedded platform in colleges nationwide)
. Provide diverse linkages (USB, LAN, Bluetooth, serial, IrDA, RF)
. Provide diverse storage facilities (USB, SDRAM, FLASH)
. Provide a large-scale LCD for GUI(640x480)
. Touch screen method
. Supported with PS2 keyboard
. Supported with diverse USB 2.0 devices (Wireless, Bluetooth, memory, Hard disk) . Supported with Zigbee
. Equipped with REID
. Capable of being geared with SEM 21, RF, and Bluetooth

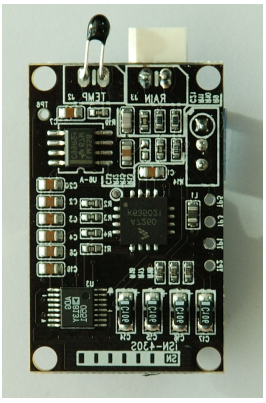
Hardware Item	Description
Micro Controller	ATmega128 (program 128Kbyte SDRAM 2KB EEPROM 4KB AD 10bit 8 Channels)
RF part	CC2420 2.4GHz Zigbee(IEEE 802. 15. 4)
Security	DSSS
Transfer BPS	Maximum 250K BPS
Base sensor	Basically equipped with temperature, illumination and humidity sensors, and RTC
Power	1.5V AA 2ea or 1.2V Rechargeable battery 2ea
Length	40mm × 70mm

1.4. iSN-400N

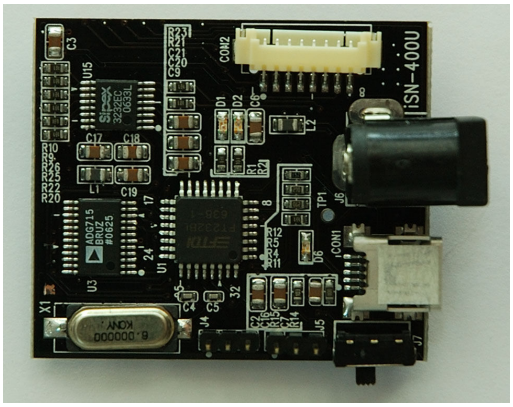
iSN-400N is a low-power and high performance mote, which includes temperature, motion and rain sensors. It uses 250Kbps 2.4GHz IEEE 802.15.4 Chipcon CC2420 RF chip.



Component	Model	Description
Node Board	iSN-400N	Hardware Platform
CPU	MSP430F1611	- Low-power - 8 bit micro-processor - 32.768 [Khz] clock
Memory	FLASH	48KB
	SRAM	10KB
Battery	CR123	3[V] , 1400[mAh]
RF	Chipcon CC2420	2.4GHz Zigbee



Component	Model	Description
Sensor Board	iSN-430S	Hardware Platform
Temp	NTC-103F343F	**center-Temp 25.0℃/10.00KΩ** β-Tolerance **1.00%** -40 ~ +120 ℃
Rain	iSN-202R	60[mm] circle sensor
Motion Sensor	MMA7260Q	X, Y and Z axis of accleration sensor



Component	Model	Description
Inteface board	iSN-400U	Hardware Platform

Temp	NTC-103F343F	**center-Temp 25.0℃/10.00KΩ** β-Tolerance **1.00%** -40 ~ +120 ℃
USB	FT232B	Serial Debug
Power	USB or DC jack	USB or DC jack power supply input DC 5~ 12[V]

1.5 usb-msp430

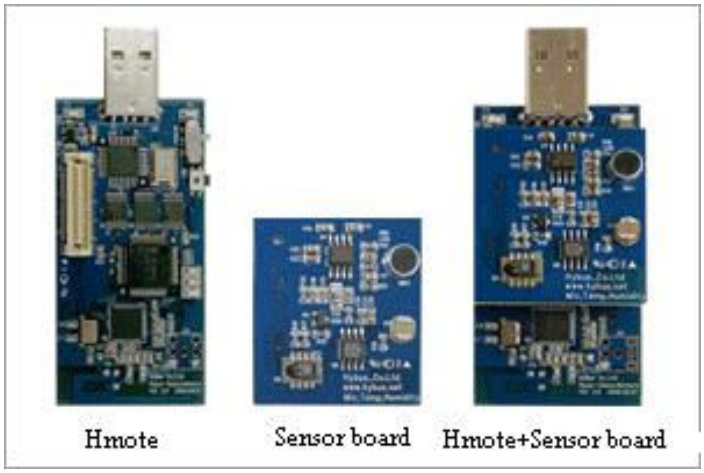
Hanback Electronic develops the new ultra-low power wireless sensor platforms (HBE-Ubi-MSP430, Ubimote) based on TI MSP-430 CPU and CC2420 RF. HBE-Ubi-MSP430 can be utilized in the education, research and performance evaluation area related the ubiquitous technology. Users can easily learn wireless sensor networks, MAC embed system and sensing control mechanisms by the platform and its various examples.

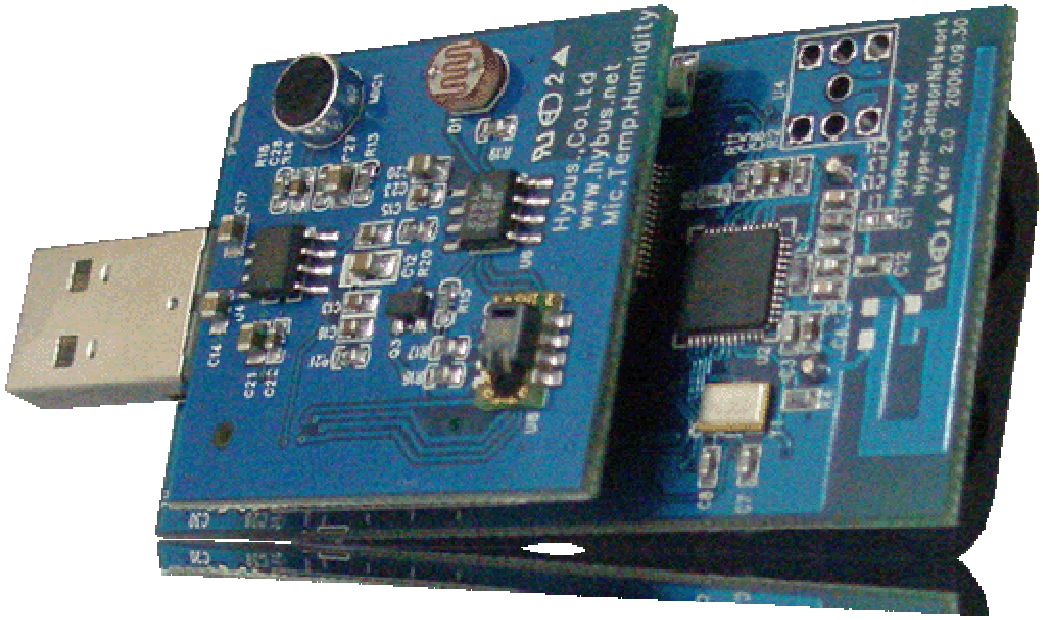
. Public MSP430 CPU
. Stable 8bit high performance platform
. External data storage (SDRAM, FLASH)
. PCB pattern antenna
. Size : 40 mm x 70 mm
. 1.2v recharge battery
. Support external extend port
. Support external antenna port to enhance a communication distance
. Support various sensors (Temp. Humi, Photo, Light)

Item	Description
Micro Controller	MSP430F1611(program 48Kbyte RAM 10Kbyte AD 8ch, DA 2ch)
RF part	CC2420 2.4GHz IEEE 802.15.4 PHY
Security	DSSS
Tranfer BPS	Maximum 250Kbps
Base Sensor	Temperature, Humidity and Photo Sensors
Power	1.5V AA 2ea or 1.2V Rechargeable battery 2ea
Length	40mm × 77mm

1.6 Hmote2420

Hmote2420 is a low-power and high performance mote, which includes temperature, humidity, light and tone sensors. It uses 250Kbps 2.4GHz IEEE 802.15.4 Chipcon CC2420 RF chip.





The features of Hmote2420 are shown below.

- Interoperable with other IEEE 802.15.4 devices for processing information
- Using Texas Instruments MSP430 microcontroller (10k RAM, 48k Flash)
- Onboard antenna (50m range indoors / 125m range outdoors)
- Ultra-low power consumption

- Fast wakeup from sleep mode (<6us)
- Enabling to attach a variety of sensor boards though connectors
- Including optional SMA antenna connector
- Programming and data collection via USB

Item	Description	count
H/W	TI MSP430 16bit Processor - Speed : 8MHz/32KHz - Program Space : 48kb, RAM : 10kB	1
	RF Chip CC2420 - Frequency : 2400~2483MHz - Data rate : 250kbps	1
	Internal antenna	1
	External antenna Interface	1
	USB Interface	1
	Extension connector	1

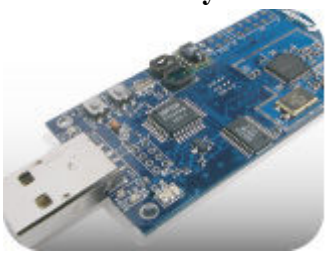
1.7 Ubi-coin

Ubi-Coin, developed by Hanback Electronics Co ., Ltd., is a micro USN unit based on the low power micro processor MSP430. It is equipped with a photo sensor and CC2420 RF chip th at provides the IEEE 802.15 .4 PHY function . Ubi-Coin configures a wireless sensor network between nodes based on the IEEE 802.15.4 MAC d eveloped by Hanback Electronics Co., Ltd. It enables download and serial communications with PC through the USB port. Use a micro USN equipment & low-power hardware, and convert various modes through buttons. HBE-Ubi-Coin is designed to be embedded on various hardwares, and its size is very small. (3cm x 3cm). It is equipped with a luminance sensor basically. Besides it, various sensors can be mounted on it, too.

CPU	MSP430 1611
Flash	48KByte + 256Byte
RAM	10KByte
RF MODEM	CC2420
RF Power	0dBm
RF range	Maximum 120m(usually 60m)-outdoor
Push Button	1ea

Color LED(option)	Full Color LED 1ea(12 colors indicated)
Beep(option)	Piezo(96 stage)
Power	3V DC

1.8 Tmote-Sky



Reliable low-power wireless sensor networking eases development and deployment.

Tmote Sky is the next-generation mote platform for extremely low power, high data-rate, sensor network applications designed with the dual goal of fault tolerance and development ease. Tmote Sky boasts the largest on-chip RAM size (10kB) of any mote, the first IEEE 802.15.4 radio, and an integrated on-board antenna providing up to 125 meter range. Toward development ease, Tmote Sky provides an easy-to-use USB protocol for programming, debugging and data collection.

Tmote Sky offers a number of integrated peripherals including a 12-bit ADC and DAC, Timer, I2C, SPI, and UART bus protocols, and a performance boosting DMA controller. Tmote Sky offers a robust solution with hardware protected external flash (1Mb in size), applications may be wirelessly programmed to the Tmote Sky module. In the event of a malfunctioning program, the module loads a protected image from flash.

Key Features

- 250kbps 2.4GHz IEEE 802.15.4 Chipcon Wireless Transceiver
- Interoperability with other IEEE 802.15.4 devices
- 8MHz Texas Instruments MSP430 microcontroller (10k RAM, 48k Flash)
- Integrated ADC, DAC, Supply Voltage Supervisor, and DMA Controller
- Integrated onboard antenna with 50m range indoors / 125m range outdoors
- Optional Integrated Humidity, Temperature, and Light sensors
- Ultra low current consumption
- Fast wakeup from sleep (<6us)
- Hardware link-layer encryption and authentication
- Programming and data collection via USB
- 16-pin expansion support and optional SMA antenna connector
- TinyOS support : mesh networking and communication implementation
- FCC modular certification : conforms to all US and Canada regulations

2 Nano OS Installation

To install Nano OS, some preliminary work is needed. Currently, Nano OS has been installed in MS-Windows environments. However, it can be installed on Linux systems very easily because Nano OS is based on cygwin environments. The following packages should be pre-installed for Nano OS.

1. Cygwin : a linux emulator for Windows

2. Compiler : You should select one of the following compilers depending on your hardware
(1) **[ETRI-SSN, Nano-24, MICAz, ZigbeX, SKY-Z200]** Winavr : avr-gcc compiler for Windows
(2) **[ISN-400N, Ubi-msp430, Ubi-coin, Hmote2420, Tmote-Sky]** Mspgcc : mspgcc compiler for Windows

3. In-System Programming tool for fusing : You should select one depending on your hardware
(1) **[ETRI-SSN, Nano-24]** PonyProg2000 (parallel cable connection) **(ponyprog)**
(2) **[ETRI-SSN, Nano-24]** Avrdude (serial connection through USB interface board)
(3) **[MICAz]** Uisp (serial connection through USB interface board)
(4) **[ISN-400N]** Mspfet (serial connection through USB interface board)
(5) **[Ubi-msp430, Ubi-coin, Hmote2420]** Mspbsl (serial connection through USB interface board)

2.5 Cygwin [Common]

Cygwin is a windows program from GNU, which enables us to emulate linux environment on Windows systems. Because it provides many unix-compatible utilities, developers are able to use them (e.g. X window, vi editor, gnu make, latex, etc.) in MS-windows. To install Cygwin, go to the website, http://www.cygwin.com, and download necessary packages. Cygwin has a lot of optional packages. For Nano OS, it is recommended to download editor tools (vi or emacs), programming development tools (gcc, make), and shells (bash or csh). If you are not an expert for installing Cygwin, it is desirable to download and install all packages. However, it will take much time. The common path of installation is “c:\cygwin”.

2.6 WinAVR (avr-gcc compiler) [ETRI-SSN, Nano-24, MICAz, ZigbeX, SKY-Z200]

WinAVR is an integrated tool for AVR series. It contains an avr-gcc compiler, which is necessary to compile Nano OS application programs. You can download it at the website,

http://winavr.sourceforge.net. Download the latest WinAVR program (later than April. 4th, 2006). Otherwise, you will see many warning/error messages. The common path is "c:\WinAVR".

2.7 mspgcc compiler [ISN-400N, Ubi-msp430, Ubi-coin, Hmote2420, Tmote-Sky]

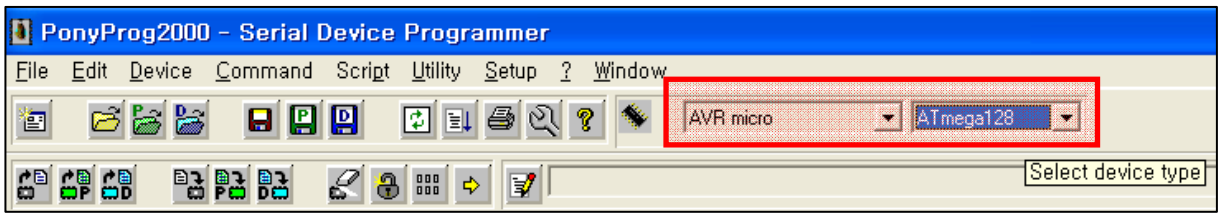
MSPGCC is a completely free and unlimited C compiler for TI's MSP430 series of microcontrollers. There are NO limitations. This is a port of the GNU C Compiler (GCC) and GNU Binutils (as, ld) for the embedded processor MSP430. Tools for debugging and download are provided (GDB, JTAG and BSL). It is necessary to compile Nano OS application programs. You can download it at the website, http://mspgcc.sourceforge.net . Download the latest mspgcc-win32 program. The common path is "c:\mspgcc". And copy c:\cygwin\bin\cygwin1.dll to c:\mspgcc\bin.

2.8 PonyProg [ETRI-SSN, Nano-24]

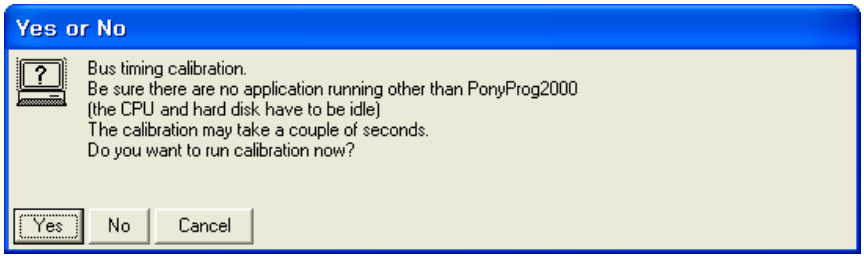
When you compile our sensor program, a program image (hex file) is created. Finally, you need to store the program in a sensor hardware board. In order to put the program image developed on a PC into a sensor board, you need to use an extra Windows program. For this task, AVR chips supports ISP (In-System Programming) that enables you to directly program the FLASH memory. One of the most popular ISP programs for AVR chips is "PonyProg" series. You can use a PonyProg2000 program to insert your program image into the ETRI-SSN sensor hardware. The program can be downloaded at the website, http://www.lancos.com/ppwin95.html. The common path is "c:\Program Files\PonyProg2000" After installing it, you should make some preparations for its operation.

Refer to the following procedure for ETRI-SSN (or Nano 24).

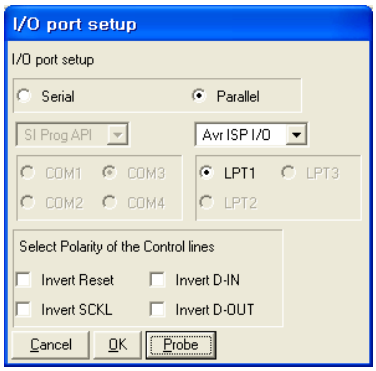
(1) The type of Chip : AVR micro → ATmega128



(2) The calibration → Select "Yes"



(3) Interface setup → Select “Parallel” and “avr ISP I/O”



2.9 Nano OS [Common]

Nano OS package is provided as a form of nos-x.y.z.tgz. Uncompress this file at the location that you want to install. For example, if you have a nos-xy.yz.tgz file at c:/ path.

$ cd c:/
$ tar xzvf nos-x.y.z.tgz

Then, you will see nos-x.y.z at c:/ directory.

2.10 Cygwin Setup [Common]

Pop up a cygwin window, and edit the /etc/profile (or ~/.bash_profile) file for your environment. NOS_HOME is the path of home directory for Nano OS, and the ISPPATH is the path of PonyProg2000. You need to change the version part of nos-x.y.z into the one that you would like to install.

$ vi ~/.bash_profile
NOS_HOME=”**/cygdrive/c/nos-2.0.0**” # change the version part, x.y.z in nos-x.y.z
ISPPATH=”**/cygdrive/c/Program Files/PonyProg2000**” # this is for **[ETRI-SSN, Nano-24]**

PATH=$PATH:$ISPPATH # this is for **[ETRI-SSN, Nano-24]**
export **NOS_HOME PATH**

2.11 Hyperterminal Setting [Common]

In PC environments, after Windows starts, the hyper-terminal program is installed. This program is located in the path, Windows Start → Program → Accessories → Communication → Hyperterminal. The hyper-termial is a program for serial communications. When debugging a sensor program, the hyper-terminal program can be useful (It can print useful information from sensor board). Thus, this progam is a prerequisite for Nano OS. For UART communication, there must be a connection between PC and a sensor node by, for example, a serial cable. The notable point is that the hardware control is set to “none” The bit rate or other options can be changed if necessary in Nano OS. In most cases, the following setting is enough: **9600** baud rate, **8** data bits, parity **none**, **1** stop bit and **no** hardware flow control.

3 Developing Procedure

This includes instructions how to develop Nano OS application programs.

1. Make your own application directory. Typically, application programs have been written under "$NOS_HOME/apps/" or "$NOS_HOME/test-apps/"
2. Make an application program using 'C' language.
3. Link '$NOS_HOME/Makefile.kconf' to './Makefile'.
4. Configurate your application program for code optimization.
   - Type 'make menuconfig' for this step. 'kconf.h' will be generated automatically. The 'kconf.h' is the configuration file that describes Nano OS modules required by application programs. Nano OS consists of several modules that can be plug-in or plug-out. Please select only modules that you will really use in your application programs for code optimization.
5. Make a library for your application. Type 'make lib'. (Note that you can omit this process.)
6. 'make' will compile your codes and generate hexa image file ('your_appl.rom')
7. Type 'make burn port=<port-option>' to load your application program image into your sensor node. The <port-option> depends on the platform and interface that you have.
   (1) **[ETRI-SSN, Nano-24]** – use '**make burn port=lpt1**' for parallel connection. In case of use connection by USB interface board, you should use '**make burn port=usb**'
   (2) **[MICAz]** – use '**make burn port=comX**', where X is the serial port number connected.
   (3) **[ISN-400N]** – move to use '**make burn**'. Then, you will see MspFet.exe, which is a window program for downloading intel hexa-formated file. Run this program. Follows the next steps for downloading.
      a) select a proper model : (msp430x1611)
      b) select an application hexa-file (*.hex) by clicking 'Open' tab.
      c) click 'Erase' and 'Program' tabs. Or you can do that at once by clicking 'Auto'
   (4) **[Ubi-msp430, Ubi-coin, Hmote2420]** – use '**make burn port=comX**', where X is the serial port number connected.

Write your own application programs, e.g. led.c. In Nano OS, sensor application programs are written in C and the coding format is shown in Fig. 8. At the uppermost of application program code, the "nos.h" header file must be included in the program. In addition, the NOS_INIT() must be placed before writing your own codes in the main program. This is the common coding format that all Nano OS application programs must adhere to.

```
#include "nos.h"

int main (void)
{
    nos_init();

    ……
   /* your codes */
    ……

    return 0;
}
```

Figure 5. Coding format of Nano OS application programs

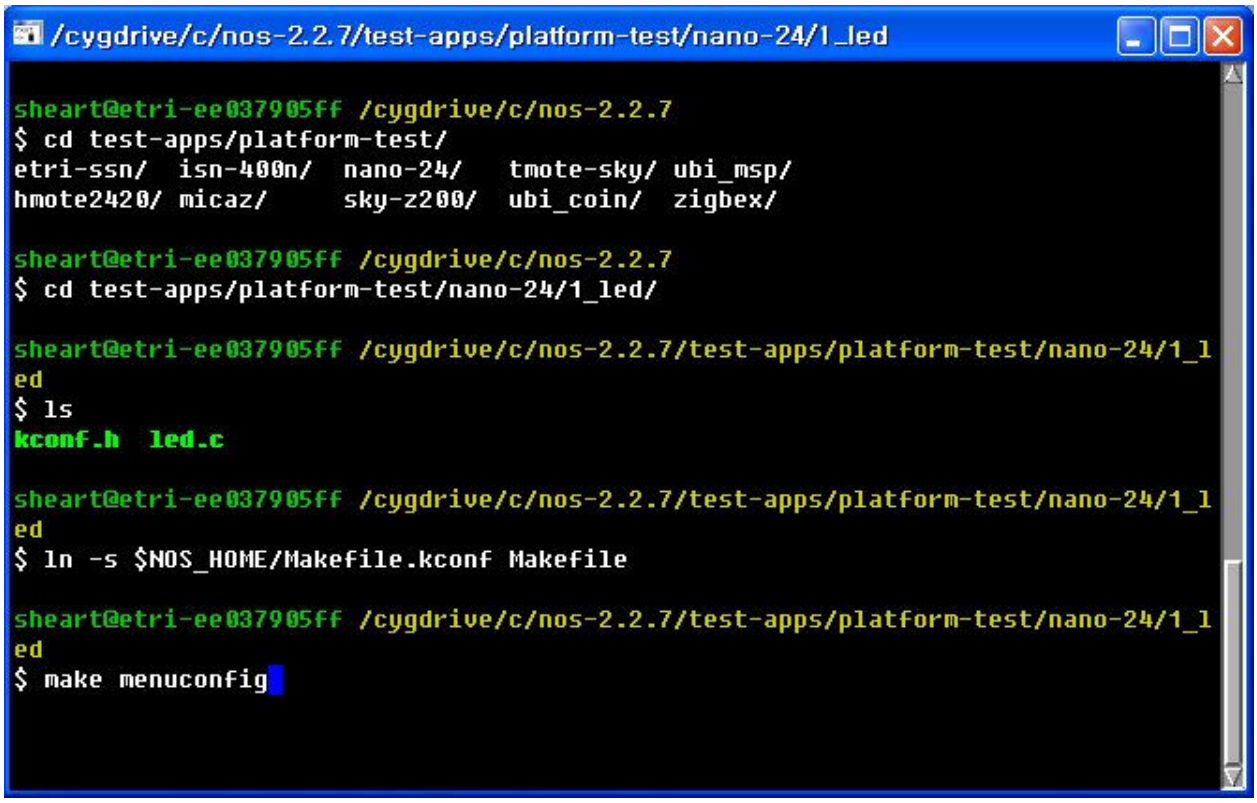


Figure 6. The development step 1, 2, 3, 4.

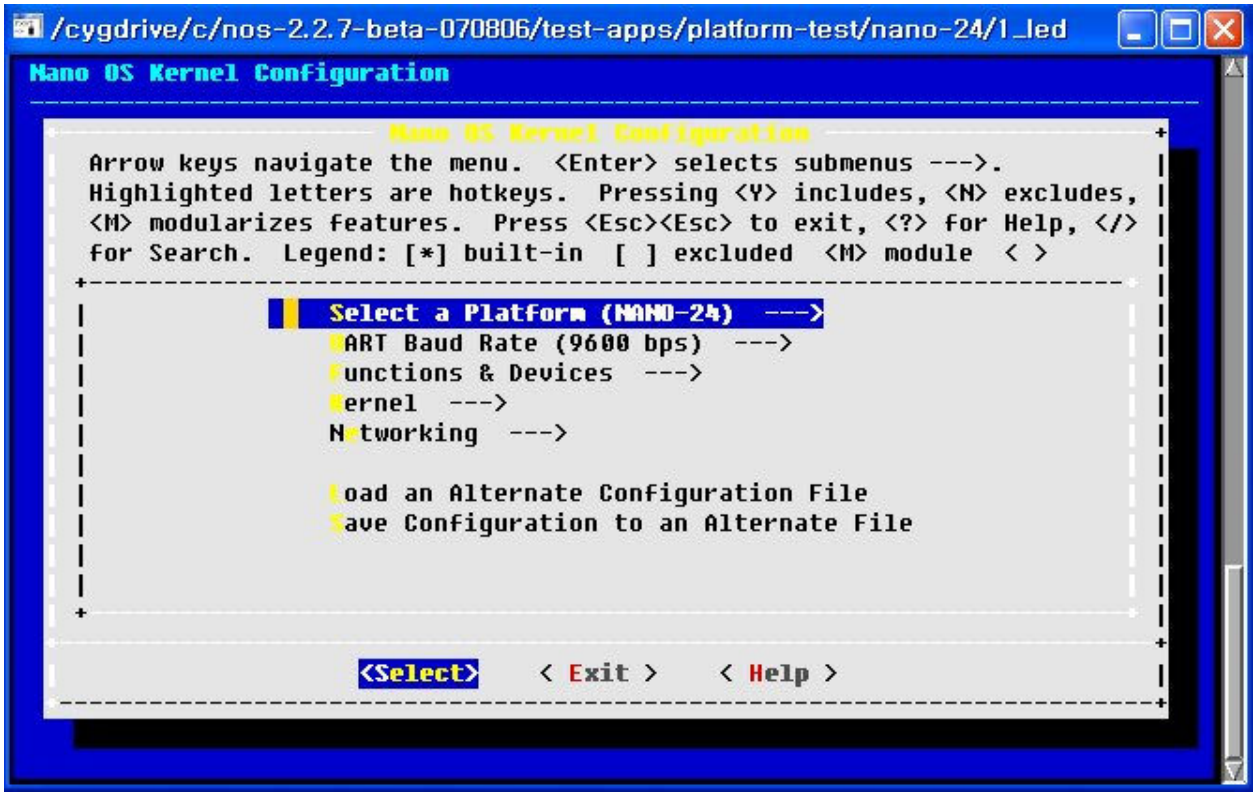


Figure 7. More details of step 4.

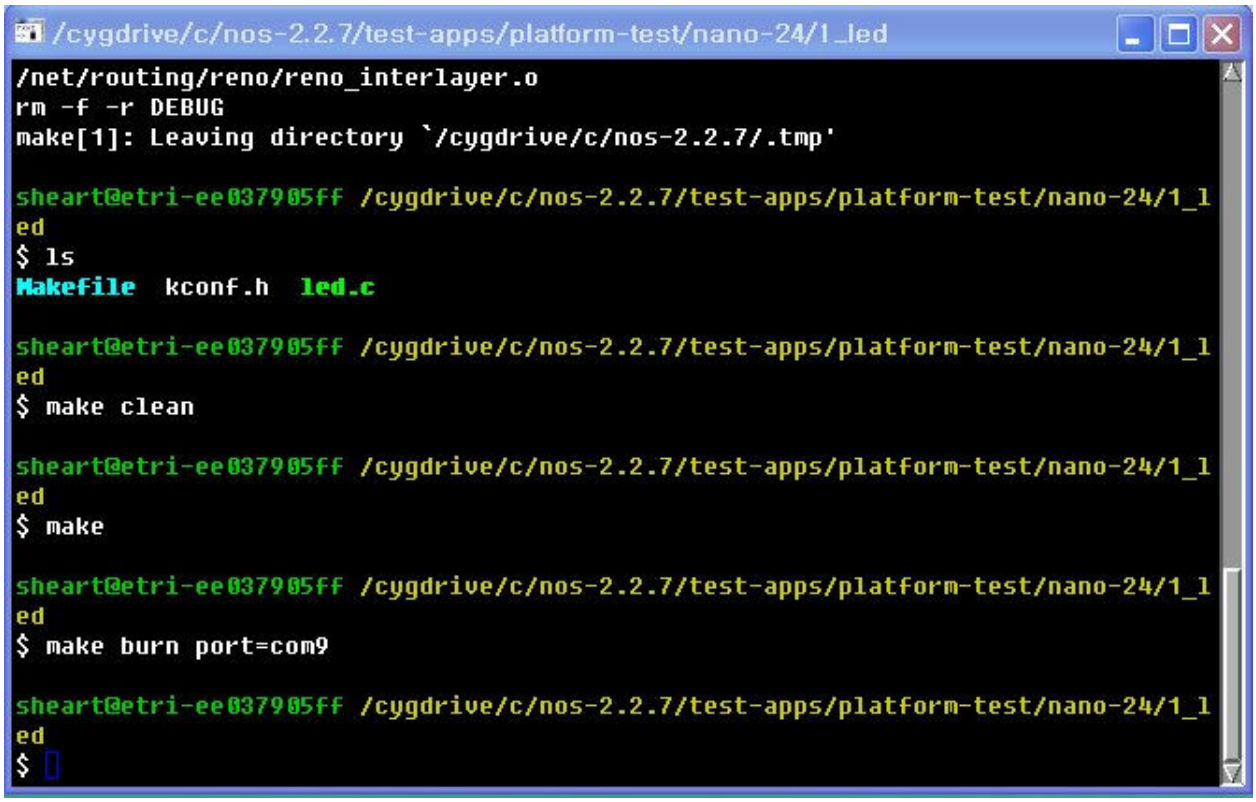


Figure 8. The development step 5, 6, 7.

Please make sure that PC is connected to the sensor node by a parallel cable when you upload the image to the sensor node. “PonyProg2000” immediately moves the ROM image to sensor node. If there are any problems at this time, check if PonyProg2000 program has been properly installed in the previous section or if the status of parallel cable, sensor node, etc. are OK.

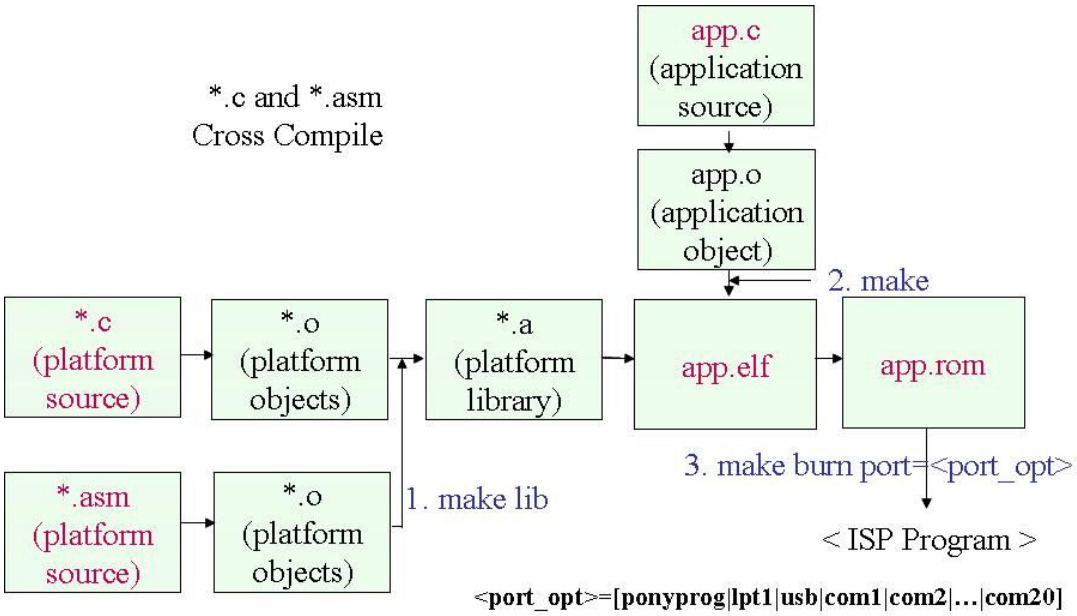


Figure 9. Development Process

4 Sensor Programming with Nano OS

Using a variety of functions that Nano OS is provided with, you can write sensor network appl. PGMs(application programs). Here is a short description of writing simple sensor appl. PGMs

4.5 Basic Tests ($NOS_HOME/test-apps/platform-test/$PLATFORM/1_led)

In Nano OS directories, there can be seen various test programs that you can look into. In this section detailed explanations for some of these programs. Code analysis will be followed by the execution results.

4.6 LED Test

```
#include "nos.h"

int main (void)
{
    nos_init();

    while(1)
    {
        // turn on LD1 (red)
        led_on(1); // RED
        led_off(2); // GREEN
        led_off(3); // YELLOW
        delay_ms(200); // delay for 200ms

        // turn on LD2 (green)
        led_off(1);
        led_on(2);
        led_off(3);
        delay_ms(200); // delay for 200ms

        //turn on LD3 (yellow)
        led_off(1);
        led_off(2);
        led_on(3);
        delay_ms(200); // delay for 200ms
    }

    return 0;
} ? end main ?
```

(1) Code Analysis

This code is written for testing LEDs on a sensor board (Here, ETRI-SSN). It is assumed that there are three LEDs in the sensor board. “led_on(n)” and “led_off(n)” macros will turn on and off n-th LED, respectively. This example shows that three LEDs on the sensor board blink repeatedly.

(2) Result

Run this example, you will see three LEDs are blinking in the following pattern; OXX, XOX and XXO, where O (LED on) and X (LED off). The delay_ms() function delays CPU for the given period. Without this function, the state changes of the LEDs will occur so fast that we cannot see them blinking.

4.7 UART Sending Test

```
#include "nos.h"

int main (void)
{
	UINT8 i;
	nos_init();
	led_on(1);
	// print string
	uart_printf("%s", "\r\n*** Nano OS ***\r\n");

	// print UINT8, UINT16 (0~65535)  --> printf(%u) or putu()
	uart_printf("UINT16 : %u~%u\r\n", 0, 65535);
	uart_putu(0);
	uart_puts("~");
	uart_putu(65535);

	// print INT8, UINT8, INT16 (-32767~32767)  --> printf(%d) or puti()
	uart_printf("\r\nINT16 : %d~%d\r\n",-32767, 32767);
	uart_puti(-32768);
	uart_puts("~");
	uart_puti(32767);

	// (unsigned) deci, hexa, octa, char
	uart_printf("\r\ndecimal = %d,  hexa = %x octa = %o char = %c\r\n", 65,65,65,65);


	while (1)
	{
		// print character
		for (i=0; i<5; i++)
		{
			uart_putc('-');
			delay_ms(200); // wait for 200 ms
		}
		for (i=0; i<5; i++)
		{
			// Backspace key handling
			uart_putc(_BS);  // Backspace character '_BS' = 0x08
			uart_putc(' ');
			uart_putc(_BS);
			delay_ms(200); // wait for 200 ms
		}

	}

	return 0;
} ? end main ?
```

(1) Code Analysis

This code is written for testing UART transmission. It shows the various execution results of “uart_putc”, “uart_puts”, “uart_puti”, “uart_putu” and “uart_printf”. In the ‘while’ loop, there are two ‘for’ statements. One ‘for’ statement prints 5 hyphen(‘-‘) characters, while the other

prints a backspace (0x08 in ASCII) and blank characters. The latter erases the hyphen(‘-‘) character. The uart_puts() and uart_putc() functions print characters on UART terminal.

(2) Result

After printing some test results, hyphen(‘-‘) characters are sequentially printed to and removed from the UART terminal.

4.8 UART Recving Test

```
//-------------- Terminal setting ----------------------
// Data bit    : 8 bit
// Stop bit    : 1 bit
// Parity : none
//------------------------------------------------------

#include "nos.h"

#define MAX_STRLEN 10 // You can change this value as you want.

INT8 str[MAX_STRLEN];

void uart_rx_handler (UINT8 rx_char)
{
    if (rx_char == '1')
    {
        led_toggle(1);
        uart_printf("\nLED1 has toggled.");
    }
    else if (rx_char == '2')
    {
        led_toggle(2);
        uart_printf("\nLED2 has toggled.");
    }
    else if (rx_char == '3')
    {
        led_toggle(3);
        uart_printf("\nLED3 has toggled.");
    }
    else
    {
        led_off(1);
        led_off(2);
        led_off(3);
    }
} ? end uart_rx_handler ?

int main (void)
{
    nos_init();
    led_on(1);
    uart_puts("\n\r*** Nano OS ***\n\r");

    uart_getc_callback (uart_rx_handler);
    enable_uart_rx_intr();

    while (1)
    {
        // uart_rx_handler is not called while "nos_uart_gets" is calling. Input 'Enter' for termination.
        uart_puts("\n\n\rType anything (up to 9characters), and then press 'Enter'.\n\r");
        uart_gets(str, sizeof(str));
        uart_puts("You typed : ");
        uart_puts(str);
        uart_puts("\n\r");

        // You can change LED state  for 10sec by calling "uart_rx_handler"
        uart_puts("\r\nType '1', '2', '3' to change LED state.");
        delay_ms(10000);
    }

    return 0;
} ? end main ?
```

(1) Code Analysis

This code tests UART reception. “uart_rx_handler” is called when UART interrupt occurs. UART ISR callback function is registered by “uart_getc_callback”. Note that you must call “enable_uart_rx_intr” to call the registered function or “disable_uart_rx_intr” to disable it. Even if “enable_uart_rx_intr” is called, “uart_gets” overrides UART callback setting (by

disabling it) and then receives a string from a keyboard until ‘ENTER” key is pressed. After that, UART callback setting is resotred.

(2) Result

You can type a string of 10 characters. Type ‘ENTER’ to terminate. After that, you have 10 seconds to turn on or off LEDs.

4.9 Sensor Test

4.10 Light, Temperature, Humidity, Gas Sensors

These four sensors use ADC (Analog Digital Converter) to get sensor data. Sensor data is available after ADC conversion is completed. If the data from one of the sensors is detected by polling, MCU takes the data immediately.

/cygdrive/c/nos-2.2.5/test-apps/platform-test/etri-ssn/sensor/gas

```
//====================================================================
//
// gas.c (@sheart)
//
//====================================================================
// Copyright 2004-2010, ETRI
//====================================================================
#include "nos.h"

int main(void)
{
        nos_init();
        led_on(1);
        uart_puts("\n\r*** Nano OS ***\n\r");

        gas_power_on(); // power on GAS sensor

        while(1)
        {
                uart_printf("Current Density of Gas : %u (PPM)\n", gas_get_data());
                delay_ms(1000); // Do sensing every 1 second
        }

        gas_power_off();        // power off GAS sensor

        return 0;
} // main
```

(1) Code Analysis

To activate each sensor, call the XXX_power_on(), where XXX is one of light, temp, gas, hum or so. Likewise, to deactivate each sensor, you should call XXX_power_off(). These two functions are used in pair. The delay_ms(1000) delays the CPU clock for 1 second to delay the program. The XXX_get_data() obtains sensor data from the ADC, where XXX is one of light, temp, hum and gas or so. It uses ADC to get sensor data.

(2) Result

You can test each sensor as follows;

1) For light sensor, cover the light sensor by your hand so that light cannot reach the sensor. The light sensor value decreases.
2) For temp. sensor, hold your finger onto the temperature sensor. The heat of your hand will be transferred to the temperature sensor, which will increase the sensor data.
3) For humidity sensor, blow your breath softly on the humidity sensor. The vapor in your breath will increase the sensor data.
4) For gas sensor, move the gas sensor close to the lighter gas. The gas of the lighter will increase the sensor data.

4.11 PIR sensor

/cygdrive/c/nos-2.2.5/test-apps/platform-test/etri-ssn/sensor/pir

```
#include "nos.h"

// Interrupt occurs when detected object is gone.
void pir_intr_handler(void)
{
        uart_puts("\n\r\tPIR Detected!\n\r");
}

int main(void)
{
        nos_init();
        led_on(1);
        uart_puts("\n\r*** Nano OS ***\n\r");

        pir_power_on(); // power on PIR SENSOR sensor
        pir_callback(pir_intr_handler);

        while (TRUE)
        {
                uart_putc(_BS);
                uart_putc('-');
                delay_ms(200);
                uart_putc(_BS);
                uart_putc('\');
                delay_ms(200);
                uart_putc(_BS);
                uart_putc('|');
                delay_ms(200);
                uart_putc(_BS);
                uart_putc('/');
                delay_ms(200);
        }

        pir_power_off(); // power off PIR SENSOR sensor

        return 0;
}
```

(1) Code Analysis

Unlike the previous sensors, the pir sensor works by an interrupt. After the pir sensor is activated, the program just runs the loop until an interrupt occurs. If the pir sensor detects an object, an interrupt occurs and the pir_intr_handler() function will be invoked. The pir_intr_handler() is a callback function that programmers have to write.

(2) Result

If you run this program, the program will print '- \ | /' repeatedly on the UART terminal until an interrupt occurs. If your hand is moving over the pir sensor, the interrupt handler will be invoked to print "PIR Detected" on the UART terminal.

4.12 Ultrasonic sensor

/cygdrive/c/nos-2.2.5/test-apps/platform-test/etri-ssn/sensor/us

```
#include "nos.h"

void us_intr_handler(void)
{
        uart_puts("WnWrDistance : ");
        uart_putu(us_get_data());
        uart_puts("cmWnWr");
}

int main(void)
{
        nos_init();
        led_on(1);
        uart_puts("WnWr*** Nano OS ***WnWr");

        us_power_on();  // turn on US sensor
        us_callback(us_intr_handler);

        while(1)
        {
                uart_puts("WnWrTriggering..");
                us_trigger();           // Turn interrupt on while triggering
                delay_ms(1000);
        }
        us_power_off(); // turn off US sensor
        return 0;
} // end of main
```

(1) Code Analysis

The US sensor also works by an interrupt. Entering the while loop, it calls the us_trigger() function, which triggers a US transmission signal into the air, and waits until an interrupt occurs. If the US signal is returned, the interrupt handler us_intr_handler() is invoked. This handler should be written by the programmer. Then, the us_get_data() function will get the distance from the US sensor to the object. This function calculates the distance using the round trip time of the signal.

(2) Result

When you run this program, you will see the UART terminal prints US sensor data repeatedly. US sensor measures the distance to object, so if your hand gets closer to the US sensor, the sensor value decreases.

4.13 Kernel Test

In kernel test, we introduce only thread testing.

```
#include "nos.h"

void do_loop(UINT8 id)
{
    UINT8 i,j;
    for (i=1; i<10; ++i)
    {
        ENTER_CRITICAL();
        uart_printf("\nTask%d is now working. - ", id);
        for (j=0; j<i; ++j)
        {
            uart_puts("*");
        }
        EXIT_CRITICAL();
        delay_ms(100);   // To slow down uart printf speed
    }
}

void new_task(void *args)
{
    UINT8 my_id = get_thread_id();
    UINT8 my_value = *((UINT8 *)args);
    uart_printf("\n===A new Task%d has been created by Task%d.===", my_id, my_value)
    do_loop(my_id);
    uart_printf("\n===Task%d has exited.===\n", my_id);
    //"thread_exit()" will be called.
}

void task1(void *args)
{
    UINT8 my_id = get_thread_id();
    while (1)
    {
        do_loop(my_id);
        if (thread_create(new_task, &my_id, 0, PRIORITY_NORMAL) < 0)
            uart_printf("\n===Task%d has failed to create a new thread.===",my_id);
    }
}

void task2(void *args)
{
    UINT8 my_id = get_thread_id();
    while (1)
    {
        do_loop(my_id);
        if (thread_create(new_task, &my_id, 0, PRIORITY_NORMAL) < 0)
            uart_printf("\n===Task%d has failed to create a new thread.===",my_id);
    }
}

void task3(void *args)
{
    UINT8 my_id = get_thread_id();
    while (1)
    {
        do_loop(my_id);
        if (thread_create(new_task, &my_id, 0, PRIORITY_NORMAL) < 0)
            uart_printf("\n===Task%d has failed to create a new thread.===",my_id);
    }
}


int main (void)
{
    nos_init();
    led_on(1);
    uart_puts("\n\r*** Nano OS ***\n\r");

    thread_create(task1, NULL, 0, PRIORITY_NORMAL);
    thread_create(task2, NULL, 0, PRIORITY_NORMAL);
    thread_create(task3, NULL, 0, PRIORITY_NORMAL);

    sched_start();

    return 0;
```

(1) Code Analysis

3 threads are created in main function. Each thread will execute task1(), task2() and task3(), respectively. The stack size of each thread has a fixed value, DEFAULT_STACK_SIZE that is defined in “arch.h”. If a thread uses the thread stack with larger size than DEFAULT_STACK_SIZE, the stack size must be increased (change ‘0’ of arg3 to any number of additional stack size). Though, in most cases it will be enough.

Note that the 3 threads have the same priorities. This implies that threads will be run in a round robin fashion. Afterwards, when the sched_start() function is called, the threads will start to run, while performing context switching,.

4.14 MAC Test

Two sensor nodes are required to test MAC applications. One is the sender node, while the other is the receiver node. Thus, two source codes (tx.c and rx.c) are prepared, each of which should be installed to their own nodes.

/cygdrive/c/nos-2.2.7/test-apps/net-test/nano-mac/tx

```
#include "nos.h"

NMAC_TX_INFO tx_info;   // Simple mac Tx information
UINT8 tx_payload[NMAC_MAX_PAYLOAD_SIZE];        // Simple mac max payload : 115 byte

int main (void)
{
        UINT8 n;

        nos_init();
        led_on(1);
        uart_puts("\n\r*** Nano OS ***\n\r");

        tx_info.dest_addr = 0x5678; // destination node id = 0x5678 (B)
        tx_info.payload_length  = 4; // NMAC_MAX_PAYLOAD_SIZE(115);
        tx_info.payload_ptr = tx_payload;

        // Write dummy data to tx_payload[1]~[114].
        for (n = 1; n < NMAC_MAX_PAYLOAD_SIZE; n++)
        {
                tx_payload[n] = n;
        }
        n = 0;

        mac_init(0x1a, 0x2420, 0x1234); // channel=26, PAN id = 0x2420, source node id = 0x1234 (A)
        //mac_set_rx_range(0x0000, 0xffff); // default (from mac_init)
        //mac_rx_on();          // default : on (from mac_init)
                                        // ack packet can be received even if rf has been t
urned off.
        //nmac_set_rx_cb(void (*func)(void)); // default : NULL. RX interrupt callback function

        while (TRUE)
        {
                led_toggle(2);
                tx_payload[0] = n;      // write the packet number to tx_payload[0]
                // Sends packet. Requests autoack
                if (mac_tx(&tx_info))
                {
                        uart_printf("\nTX (success): %u", n);
                }
                else
                {
                        uart_printf("\n\tTX (fail) : %u", n);
                }
                ++n;

                // Sends same packet but do not request autoack.
                mac_tx_noack(&tx_info);
                uart_printf("\n\t\t\tTX without ACK : %u", n);
                ++n;
        }
        return 0;
} // main
```

/cygdrive/c/nos-2.2.7/test-apps/net-test/nano-mac/rx

```
#include "nos.h"

NMAC_RX_INFO rx_info;   // Simple mac Rx information
UINT8 rx_payload[NMAC_MAX_PAYLOAD_SIZE];

void read_frame(void)
{

        if ( mac_rx(&rx_info) )         // Get a packet from the queue;
        {
                led_toggle(3);
                uart_printf("RX : %u\n",((UINT8*)(rx_info.payload_ptr))[0] );
                //uart_printf("RX : %u\n",rx_payload[0]);
        }
 }

int main (void)
{
        nos_init();
        led_on(1);
        uart_puts("\n\r**** Nano OS ****\n\r");

        rx_info.payload_ptr = rx_payload;

        mac_init(0x1A, 0x2420, 0x5678); // channel=26, PAN id = 0x2420, node id = 0x5678 (B)
        //mac_set_rx_range(0x0000, 0xffff); // default (from mac_init)
        mac_set_rx_cb( read_frame );    // default : NULL, sets a callback function for RX interrup
t

        while (TRUE)
        {
                delay_us(100);  // do nothing
        }
    return 0;
}
```

(1) Code Analysis

As a preliminary task, each node should initialize MAC to set the related variables(ie; RF channel, PAN ID, node ID, etc) to their default values. In this example, the common channel is 26, PAN ID is 0x2420. The sender node gets an ID of 0x1234, while the receiver node's ID is set to 0x5678. The mac_init() function performs this task.

On sender side : The sender node sends 10 bytes of numeric characters with an ACK request. Since the receiver does not want to receive ack messages, the mac_tx_noack() function is called. In the while loop, the mac_tx() function sends data and checks the ACK packet to see if the data is transmitted correctly.

On receiver side : To receive data, mac_rx() function is called. After receiving the rf message, the receiver node prints the values in the message.

(2) Result

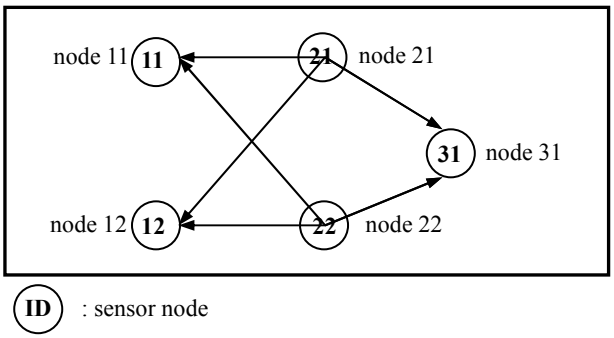
Prepare two sensor nodes and see results after downloading the application program image. Please try it by yourself! The programs will run as explained.

4.15 Advanced programs ($NOS_HOME/apps)

There are several programs to help you to write a complex sensor network program.

4.16 Bi-directional Routing Test ($NOS_HOME/apps/reno_5nodes)

Five sensor nodes are required to execute this routing test application. Three nodes (11, 12 and 31) exchange messages with each other through two router nodes (21 and 22) (Refer to the figure shown below). Five source codes - “sender11.c”, “sender12.c”, “router21.c”, “router22.c” and “sender31.c”, are prepared accordingly. Each code should be installed in the corresponding node.



For instance, the source code of “sender31.c” is as follows..

```
#include "nos.h"

UINT8 data1[ROUTING_MAX_PAYLOAD_SIZE]; //108, actually you need only 2 byte for this appl.
UINT8 data2[ROUTING_MAX_PAYLOAD_SIZE]; //108, actually you need only 2 byte for this appl.
UINT8 data[ROUTING_MAX_PAYLOAD_SIZE];

void led_control(void *payload)
{
    switch (((UINT8*)payload)[0])
    {
        case '1':
            led_toggle(1);
            break;
        case '2':
            led_toggle(2);
            break;
        case '3':
            led_toggle(3);
            break;
        default:
            led_off(1);
            led_off(2);
            led_off(3);
            break;
    }
} ? end led_control ?

//Receives string and toggles LEDs.
void rx_callback(void)
{
    UINT8 src_id;
    UINT8 length;

    while ( nwk_rx(&src_id, &length, data) )
    {
        print_route_table();
        print_payload(0, 2, data);
        led_control(data);
    }
}
```

```
// TX thread : sends string to node 11 every about 0.5sec
void task1(void* args)
{
    UINT8 dest_id, data_length;

    dest_id = 11;
    data1[1]= '\0';
    data_length = 2;

    while (TRUE)
    {
        print_local_clock();
        data1[0]='2';
        nwk_tx(dest_id, data_length, data1);
        thread_sleep_ms(200);
    }
}

// TX thread : sends string to node 12 every about 0.5sec
void task2(void* args)
{
    UINT8 dest_id, data_length;

    dest_id = 12;
    data2[1] = '\0';
    data_length = 2;

    while (TRUE)
    {
        data2[0]='3';
        nwk_tx(dest_id, data_length, data2);
        thread_sleep_ms(200);
    }
}

int main(void)
{
    UINT8 channel = 26;
    UINT16 pan_addr  = 312;
    UINT8 node_id = 31;
//--------for multi-hop test ----------------
    UINT8 min_permit_id = 21;
    UINT8 max_permit_id = 49;
//---------------------------------------

    nos_init();
    led_on(1);
    uart_puts("\n\r*** Nano OS ***\n\r");

    // Initializing  [ Channel:0x0B~0x1A (11~26), PAN addr : 0x0000~0xfffE(0~65534), Node ID(MAC short address) : 0~255 ]
    nwk_init(channel, pan_addr, node_id, min_permit_id, max_permit_id);
    nwk_set_rx_cb(rx_callback);

    uart_printf("RF Channel : %d\nPAN address: %u\nID : %d\nReceivable Range : %d ~ %d\n",
                channel, pan_addr, node_id, min_permit_id, max_permit_id);

    thread_create(task1, NULL, 2, PRIORITY_NORMAL);
    thread_create(task2, NULL, 2, PRIORITY_NORMAL);

    sched_start();

    return 0;
} ? end main ?  // main
```

(1) Code Analysis

In this routing test, there are five nodes. The RF channel and pan ID of all these nodes are set to 26 and 312, but they have unique IDs – 11, 12, 21, 22 and 31. As shown in the following figure, the nodes 11, 12 and 31 exchange messages with each other through nodes 21 and 22.

Each node can receive data from only the node ID of which value is between ‘min_permit_id’ and ‘max_permit_id’, which are the parameters of nwk_init() function. The following table shows the range of permitted ID of nodes in this example.

Node ID	min_permit_id	max_permit_id

11	0	29
12	0	29
21	11	39
22	11	39
31	21	49

The nodes 11 and 12 execute task1 or task2 respectively. The task1 periodically sends '2' messages to the node 31, and the rf_callback() receives RF messages and toggles LEDs according to the received message types.

The node 31 executes two tasks in parallel – task1 and task2. The task1 sends '2' messages to the node 11, and the task2 sends '3' messages to the node 12. Finally, the task3 receives RF messages and toggles LEDs.

The nodes 11 and 12 forward messages that are generated by the nodes 11 and 12 to the node 31, and vice versa. The tasks do not handle the packet routing. The routing function is implicitly carried out in the routing module of Nano OS (The routing codes does not appear explicitly in the application code).

If a node cannot handle the received data immediately due to high data rate, the data is temporarily stored in an internal queue for post-processing.

(2) Result

Prepare five sensor nodes. After compiling each source codes, install the generated images in the nodes. Turn on all the nodes. Then the LEDs of nodes 11, 21 and 31 will start blinking.

——————————————————— END ———————————————————