# A SoC-based Sensor Node: Evaluation of RETOS-enabled CC2430

Sukwon Choi†, Hojung Cha†, SungChil Cho††

Department of Computer Science, Yonsei University
Seodaemun-gu, Shinchon-dong 134, Seoul 120-749, Korea
{sukwon, hjcha}@cs.yonsei.ac.kr

†† Hybus Co. Ltd.
Guro-dong 212-8, Guro-gu, Seoul 152-790, Korea
soami@hybus.net

*Abstract*—**Recent progress in Wireless Sensor Networks technology has enabled many complicated real-world applications. Some of the applications demand a non-trivial amount of computation; some run multiple tasks concurrently on a sensor node. Supporting highly concurrent, heterogeneous, and computation-oriented sensor applications may require adequate functionality of the operating system. Multi-threaded operating systems for sensor networks have recently been developed to offer an alternative programming environment to the conventional event-driven system, and the operating principle of the multi-threaded system is considered suitable for this category of applications. Although a multi-threaded operating system provides many advantages, its efficient implementation, especially on a resource-limited sensor node, is a big challenge; hence, powerful hardware with low energy consumption is always sought. The latest development of SoC (System-on-Chip) technology has enabled some of the interesting processors that are suitable for the sensor node platform. The Chipcon's CC2430 processor, for instance, is equipped with a high-performance processor core and IEEE802.15.4-compliant radio in a single chip. In this paper, we describe the development of a CC2430-based sensor node, especially from the viewpoint of system software and its performance. We have successfully ported the multi-threaded RETOS operating system on the hardware, and the implementation details are discussed in the paper. As the primary objective of our work is to understand the performance of the developed hardware, we have evaluated the system with extensive experiments. Our experiences show that the CC2430-based sensor node is powerful and energy-efficient, and a suitable platform, especially for multi-threaded sensor operating systems.**

*Index Terms*— **wireless sensor network, operating system, SoC**

## I. INTRODUCTION

WSN(Wireless Sensor Network) technologies have enabled sensor nodes to be deployed in quantity to gather data and to detect certain events by using inexpensive microcontrollers and low-cost RF hardware. Various WSN applications are found in construction [1], science [2], and oceanography [3]; they are becoming complex and consequently require high computing power or even multiple applications to be executed on the nodes.

Multi-threading programming is adequate for the concurrency control required for complex applications since the technique supports automatic state management. Multi-threading has a more natural programming architecture than event-driven models, because control flow is clear in multi-threading programming [4]. These advantages have led to the development of TinyThread [5], for example, which supports thread abstraction for TinyOS, and the development of multi-threaded operating systems such as MANTIS [6] and RETOS [4,7,8]. However, multi-threaded operating systems incur overhead compared to event-driven models since context switching is required for operating systems to be preemptive and to block threads. In addition, kernel service such as system timer handling is necessary for scheduler and time quantum updates. RETOS, for example, offers several techniques to enhance threading performance. Threading, in general, can be managed efficiently in high-performance hardware as the threading overhead decreases proportionally as hardware performance improves.

Diverse efforts have recently been made in the development of sensor platforms. Mica2 [9], MicaZ [10], and Telos [11] motes are among the widely used hardware platforms. The ECO System [12] of UC Irvine offers motes as small as 1 cubic centimeter by using the nRF24n microcontroller. Yale's XYZ [13] and Intel's IMote2 [14] offer high computing performance based on the ARM processor, but the cost and energy consumption are high.

Recent technology advances in the SoC (System On Chip) have made it possible to have data processing, memory, digital signal processing, analog signal processing, and RF combined in a single Microprocessor Unit (MPU). The SoC-based MPUs have been developed for specific devices such as CDMA or network multimedia, but microcontrollers such as the nRF24n [15], CC1010 [16], and CC2030/31[17] have also been developed to meet the low-power consumption requirements for the sensor applications. In these platforms, all the necessary functionalities are performed in a single SoC-based MPU, and thus, no additional circuits such as communication bus between chips and shared memory are needed. This reduces power consumption as well as the size of the overall hardware platform. The SoC-based MPU is therefore adequate for sensor networking hardware due to the minimized size, cost, and energy consumption.

CC2430 [17] is a System-on-Chip (SoC) MCU that includes both a processor core and IEEE 802.15.4 RF (Radio Frequency); the former supports the standard 8051 instructions, and the latter supports the low-frequency communication standard. The inner clock is 32 MHz, and the peak power consumption in operation is 7.0 mA. Radio transmission and reception consume 24 ~ 27 mA, and support a 4-level low-power mode. The CC2430-based sensor nodes outperform MicaZ, Telos, and other existing sensor nodes in terms of computing power and power consumption. We have recently built CC2430-based S-Mote sensor nodes and implemented the RETOS operating system on them. S-Mote is designed to meet the requirement of sensor applications that demand a high data sampling rate with low energy consumption. To validate the functionality of S-Mote, sensor applications were implemented and analyzed using RETOS. We also compared the overall performance of S-Mote to other sensor nodes.

The rest of this paper is organized as follows: Section 2 discusses the motivation to build S-Mote. Section 3 describes the hardware feature of S-Mote, and Section 4 explains the detailed implementation of RETOS on S-Motes. Section 5 analyzes the performance of the implemented system. Section 6 concludes the paper.

## II. S-Mote: CC2430-based Sensor Platform

### A. Motivation

Sensor applications such as the Distributed Source Locating System (DSL) [18] and Structural Health Monitoring System (SHMS) [19] have currently been developed with RETOS. The applications run on TI MSP430-based TMote Sky only with on-board sensors, and do not use digital signal processing hardware. As data reliability and real-time are the important requirements of DSL and SHMS, DSL should detect the exact time of sound occurrence, and SHMS should monitor the stress of a building. This can be achieved by gathering data on the designated sensor field, and processing and transmitting them to the destination or neighbor nodes. In these applications, data are obtained from MIC(Microphone) and strain gauge through ADC(Analog to Digital Converter) in the MCU.

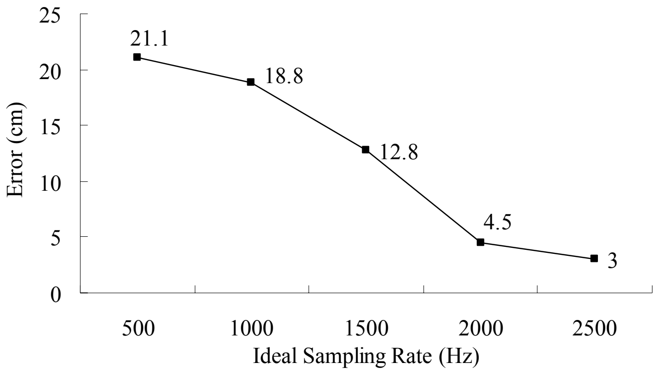


Fig. 1. Sampling rate vs. DSL error

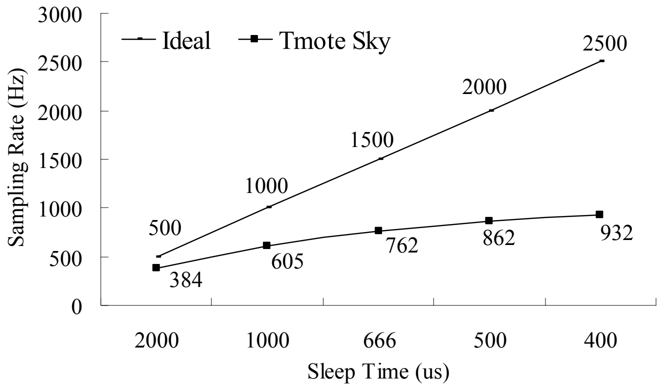


Fig. 2. Ideal sampling rate and real sampling rate on Tmote Sky

Sensor nodes used for DSL do not use separate DSP, thus the access to ADC directly influences the amount of CPU usage. To understand the relationship between the sampling rate and the accuracy of DSL, the sampling rate of sensors that use ADC and the error rate of the DSL system were measured. All the peripherals were turned off during the sampling process to raise the accuracy of the measurement. As seen in Fig. 1, the accuracy of DSL increases as the sampling rate increases. However, the CPU usage increases, too, and results in the delayed response time of the overall system. Fig. 2 shows the possible ADC sampling rates of the Tmote Sky. The low response time does not meet the required sampling rate, and the sampling rate reaches only up to 932 Hz. This can significantly damage the reliability of the obtained data. The problem originates in the performance bottleneck of the ADC access for data sampling and data processing. A high-performance MCU can solve this problem. Sensor nodes using a high-performance MCU, however, consume much energy, and clock maintenance requires an additional circuit, resulting in the size overhead of the nodes. Thus, a new system architecture for sensor nodes is necessary to meet high performance, low cost, and low-power consumption simultaneously.

Table I Hardware Comparison [9, 11, 12, 15, 17]

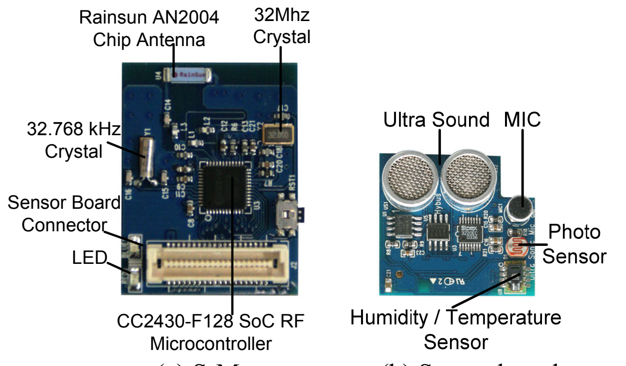
| | MICAZ | Tmote Sky | XYZ node | ECO System | S-Mote |
|---|---|---|---|---|---|
| CPU type | Atmel | MSP430 | ML-67Q500 | nRF24e1 | CC2430 |
| Instruction | 8-bit CISIC (AVR) | 16-bit RISC (MSP) | 32-bit RISC (ARM7) | 8-bit CISC (8051) | 8-bit CISC (8051) |
| CPU Clock | 16Mhz | 8Mhz | 60 MHz | 20MHz | 32Mhz |
| Minimum Voltage | 2.7V | 1.8V | 2.25V | 1.9V | 2.0V |
| Active Power | 8 mA | 2 mA | 40mA | 3mA | 7 mA |
| Sleep Power | 20uA | 27uA | 20 uA | 2uA | 0.9uA |
| MCU + RX Power | 23.3mA | 21.8mA | 69.8 mA | 10.5 mA | 27 mA |
| MCU + TX Power | 21.0mA | 19.5mA | 57.5 mA | 19.0mA | 24.7 mA |
| RADIO Chip | CC2420 | CC2420 | CC2420 | nRF24e1 | CC2430 |
| Price (MCU+RF) | $9.20 | $9.50 | $8.43 | $3.25 | $3.90 |

## B. S-Mote Platform

In Table 1, we compare several existing systems to evaluate sensor hardware that satisfies the applications' high-performance requirement. The XYZ node uses ML67Q500x, which is an ARM-based 32bit processor, but consumes excessive power. The ECO System uses an nRF24e1 processor and SoC-based 2.4 GHz RF. The platform is close to our objective: the peak clock is only 20 MHz, which is lower than the clock of CC2430. The processor performance is not quite comparable to Tmote Sky.

CC2430 uses 8051-based 8-bit instructions at a 32 MHz core clock, and adopts enhanced instruction processing architecture, outperforming other sensor nodes such as MicaZ or Telos. In addition, the advantage of SoC architecture removes unnecessary bus status and provides several features for low-power consumption, such as systematic DMA, MAC timer, and low-power timer. Our CC2430-based sensor node is shown in Fig. 3. To support low-power consumption, we added a 32.768 KHz crystal oscillator to the node so that it can reduce power consumption. We minimized the space for mounting an antenna and supported the omni-directional feature by employing a Rainsun AN6502 ceramic chip antenna. The node can be connected to a sensor board or programming board through a connector. The developed sensor board has an ultrasound sensor, MIC, Photo sensor, and temperature/humidity sensor.

This paper primarily evaluates if S-Mote can be used for applications that require high-performance processing power, such as found in DSL. We have ported the RETOS operating system to S-Mote, and analyzed the performance by using benchmark programs and real applications on the nodes.


Rainsun AN2004 Chip Antenna
32Mhz Crystal
32.768 kHz Crystal
Sensor Board Connector
LED
CC2430-F128 SoC RF Microcontroller
Ultra Sound
MIC
Photo Sensor
Humidity / Temperature Sensor

(a) S-Mote (b) Sensor board

Fig, 3. CC2430-based S-Mote mote and sensor board

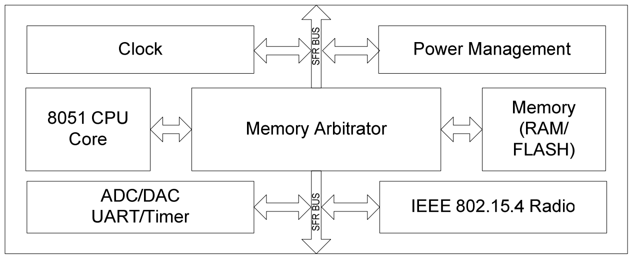


Fig. 4. CC2430 block diagram

## III. RETOS Implementation on S-Mote

The RETOS operating system has been designed to satisfy three goals: supporting a multi-threading programming environment, offering a robust system by protecting the kernel from application errors, and giving functional extensibility to the sensor operating system. RETOS protects the system from application errors by adopting a dual mode and kernel protection mechanism. The kernel and applications are developed in Standard C language. The applications written by developers are loaded on the operating system only after their codes are proved to be robust. RETOS supports basic functionality of a standard operating system, a static kernel such as hardware-dependent codes, shared library, and dynamic kernel modules for various kernel functionalities. The following section describes the implementation issues of RETOS on S-Mote.

## A. The CC2430 Hardware

The main difference between the CC2430 microcontroller and other 8051-based microcontrollers is the memory architecture due to the SoC architecture. Fig. 4 shows the block diagram of CC2430, which consists of 6 blocks with a Memory Arbitrator in the center. The Memory Arbitrator manages the data transaction between the CPU and the memory. Data exchange between clock, power source, peripherals, and radio is done by the SFR bus. We should consider this shared bus and memory architecture

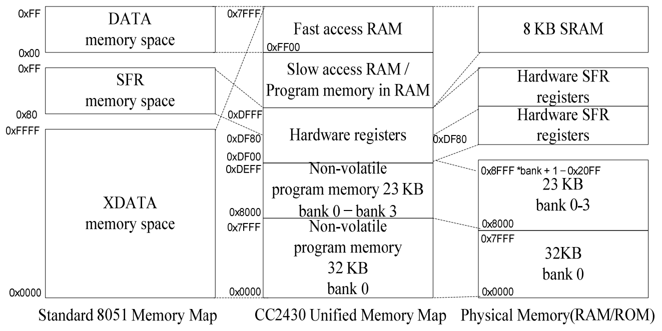


Fig. 5. CC2430 memory map

to maximize the performance of the processor while porting the operating system.

Unlike typical 8051 processors, the CC2430 consists of four independent memory spaces: CODE, DATA, XDATA (External DATA), and SFR (Special Function Register memory map). There are two different methods of CODE area mapping: non-unified mapping and unified mapping. Non-unified mapping is compatible to the 8051 processor, and unified mapping can use CC2430-optimized functions. We used unified mapping to maximize the performance, which is shown in Fig. 5. Efficient memory usage is possible by partitioning the area: the upper half for fast memory space and the lower half for slow memory space. The upper 4 KB are designated as the DATA memory space and used during code execution, whereas the lower 4 KB can be accessed via the XDATA area.

One advantage that comes with using the CC2430 in a sensor network environment is its capability of direct management of radio-related interrupts. The CC2430 contains 18 interrupt sources. They are categorized into six groups and are given priorities accordingly. The radio-related interrupt group is given the highest priority, and even in the same group, each interrupt is given a polling priority. The highest priority is given to the radio buffer-related interrupt, and the next is given to general RF interrupts. In other words, the processor is designed to have data transmission and reception take precedence over anything else.

## B. *Dual-mode Operation*

RETOS is a dual-mode operating system, which provides both kernel mode and user mode operations, based on a preemptive multi-threaded kernel. RETOS protects the kernel from invalid operations by applications. RETOS adopts a single kernel stack to provide memory-efficient threading. Application codes are run in the user stack during the execution, and system calls and interrupt service routines jump to the kernel stack and use it after saving the stack pointer of the currently executing applications in TCB. We maximized the use of the CC2430's memory architecture to provide mode switching, as shown in Fig. 5. This area is designated as the fast memory area and does not lose data even with a change of power mode. This feature guarantees immediate execution in any situation when the stack pointer is updated properly. The slow memory area is used for the mode change. Since slow memory does not support indirect read/write, access should be made through the XDATA area. This is useful since the kernel can use the XDATA area as an auxiliary memory without interfering with the DATA area.

## C. *Relocation*

RETOS uses a relocation method to support address-independent application loading and dynamic kernel libraries. A relocation method requires accurate manipulation of memory space at the point of adding dynamic codes, and necessitates a thorough understanding of codes and data area. The CC2430 can use 55 KB out of 128 KB flash memory, which is unified memory mapping. The lower 32 KB of the 55 KB are used for actual program memory, including boot and kernel codes. The remaining 73KB are used to store data. Applications and libraries should be recognized within the lower 32 KB. As shown in Fig. 5, code is written in the 32 KB of physical flash memory area, and the operation should not be a problem when an accurate address is given at the point of code uploading. However, the upper 23 KB memory addresses have a different flash memory bank, which requires an additional operation. Our work aims to maximize the use of code area so that a multitude of dynamic kernel libraries can be run. To achieve this goal, 23 KB is also used when 32 KB are not sufficient to run all the kernel libraries. For this purpose, the relocation method was adopted, which matches all the CODE area to the XDATA area. The XDATA's capability of using all the memory map of the CC2430 has made a relocation method possible with a single unified interface of code and data.

## D. *Networking*

The CC2430 has a particular advantage in networking functionality. The RETOS networking abstraction [20] consists of three-level layering: the MAC & Physical Link Layer (MLL), the Networking Support Layer (NSL), and the Dynamic Networking Layer (DNL). This layering was designed to give application developers an easy development environment. The NSL and the DNL are independent of the hardware and were ported to the CC2430 without modification, whereas the MLL was modified to fit the CC2430.

The physical network provided by the CC2430 is identical to the CC2420, which operates at 250kbps, 2.4GHz DSSS and modulates with O-QPSK, offering compatibility with the IEEE 802.15.4 MAC. Transmission frames conform to the IEEE 802.15.4 format, and basic software control is sufficient for proper operation. We developed the RETOS MAC protocol, packet format, and basic functionality for the NSL such as CCA and LQI retrieving routines based on this architecture. The whole network device driver architecture follows the RETOS R-DDA (RETOS Device Driver Architecture) standard. DMA and a CC2430 MAC Timer were used to optimize network performance. The CC2430 supports RADIO DMA trigger, which triggers when data arrives at the RXFIFO (Radio Receive Data FIFO Buffer) or data are read. With this, data reception and processing are possible without waking up the CPU by connecting the RXFIFO and DMA. This guarantees low power consumption and high utilization of processors. The MAC timer, which is designed to support the CSMA-CA implementation, makes accurate execution and overflow management possible. A sleep timer and 32.768 KHz clock are supported as well to reduce power consumption.

## *E. Power Management*

Power management is a necessary feature for battery-powered wireless sensor nodes. To support power management, the CC2430 offers several operations to reduce power consumption. RETOS has a thread-based time scheduling model, and hence variable timer function is used for power management. We used the power-saving operations of the CC2430 and the variable timer when implementing RETOS. The CC2430 provides four levels of low-power modes, from PM0 through PM3, to reduce static and dynamic power consumption. PM0 functions with clock oscillators and the voltage regulator on. PM1 functions with 32.768 kHz oscillators and the voltage regulator on. PM2 runs with 32.768 kHz oscillators on and the voltage regulator off. PM3 runs with all clock oscillators and the voltage regulator off.

A variable timer can reduce power consumption since the timer is called only at the request of threads. In our experiments, mode switching was implemented based on the last interrupt. PM2 was set to the default mode, and either PM0 or PM1 was used based on the ADC usage. A sleep timer was used to maintain the interrupts and system timer during the sleep mode because a sleep timer runs with a 32.768 kHz power-saving oscillator.

DMA is another feature that can reduce power consumption. DMA can switch the mode of the CPU to

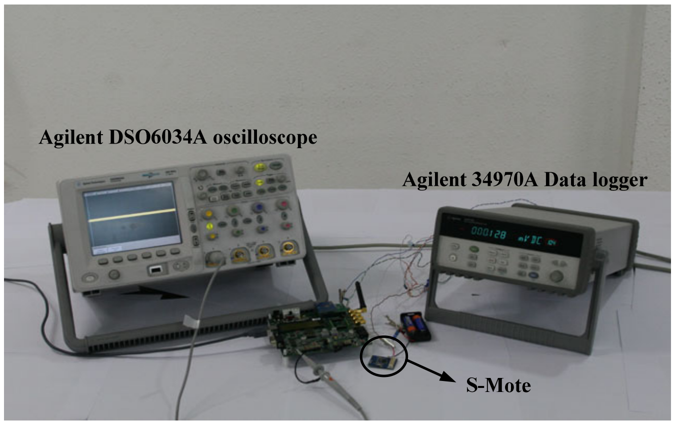


Fig. 6. Experiment setup

sleep mode even during sensing and radio transmission and reception to reduce power consumption. In our implementation, the power-mode decision is made based on the I/O operation blocking and DMA interrupts.

## IV. Evaluation

We conducted various experiments to evaluate the performance of S-Mote. First, we analyzed system factors that affect the performance of RETOS. Next, we measured the data processing time and energy consumption of applications that require computation power. Finally, we executed a real-world application and evaluated the feasibility of S-Mote.

We connected S-Mote to the Chipcon Smart RF04EB Development Board [21] for debugging purposes and analyzed the results with the IAR Embedded Workbench. With the Agilent DSO6034A oscilloscope, we measured the execution time and performance, and with the Agilent 34970A Data logger, we measured the current through a register connected to a battery pack (Fig. 6). The MSP430-based Tmote Sky [11] sensor node is chosen as a comparative sensor node to S-Mote, since the MSP430-based sensor node is a popular sensor node supported by TinyOS and has the same radio characteristics and low power management. All benchmarking methods for Tmote Sky are the same as S-Mote's.

## *A. Component Analysis*

RETOS experiences some overhead due to mode-switching and thread handling while operating a dual-mode operation. To evaluate how S-Mote reduces these overheads, we measured kernel performance and the corresponding mechanisms. We used a multi-threading application, shown in Fig. 7, for this experiment. The application creates three individual threads. Two receive a

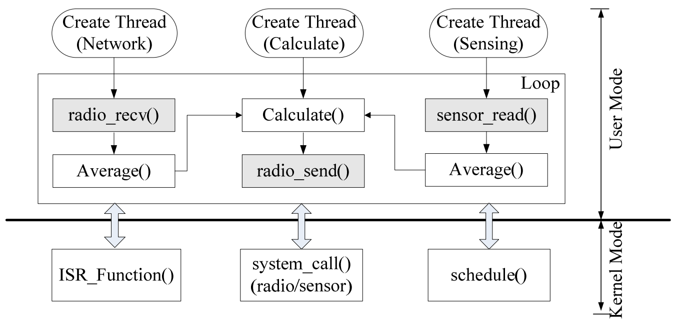
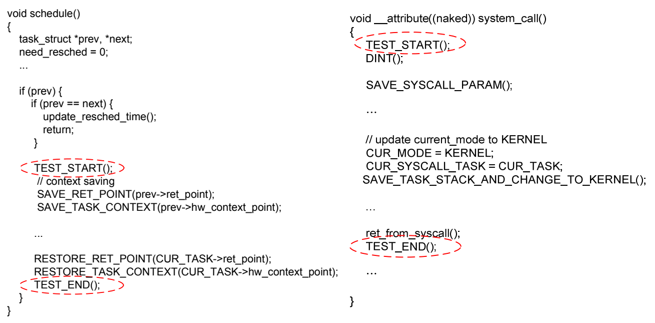


Fig. 7. Benchmarking application



```
void schedule()
{
    task_struct *prev, *next;
    need_resched = 0;
    ...

    if (prev) {
        if (prev == next) {
            update_resched_time();
            return;
        }

        TEST_START();
         // context saving
         SAVE_RET_POINT(prev->ret_point);
         SAVE_TASK_CONTEXT(prev->hw_context_point);

        ...

        RESTORE_RET_POINT(CUR_TASK->ret_point);
        RESTORE_TASK_CONTEXT(CUR_TASK->hw_context_point);
        TEST_END();
    }
}
```

```
void __attribute((naked)) system_call()
{
    TEST_START();
    DINT();

    SAVE_SYSCALL_PARAM();

    ...

    // update current_mode to KERNEL
    CUR_MODE = KERNEL;
    CUR_SYSCALL_TASK = CUR_TASK;
   SAVE_TASK_STACK_AND_CHANGE_TO_KERNEL();

    ...

    ret_from_syscall();
    TEST_END();

    ...

}
```

(a) Mode switch (b) Thread switch

Fig. 8. Mode switching and thread switching overhead

radio packet from a neighbor node and sample the data from a sensor device. The third thread calculates data from other threads and returns the result to the neighbor.

Three threads are continuously switched by the kernel scheduler. Some system calls such as *radio_send*, *radio_recv,* and *sensor_read* cause mode-switching. Two radio system calls request an ISR function to handle the radio device interrupt. We measured the operation time of this part to evaluate the overhead of a threading/dual-mode operation, radio send/recv, and sensing devices. To measure, we added the benchmarking code shown in Fig. 8 to the kernel code. Each operation is kept track of by the added code.

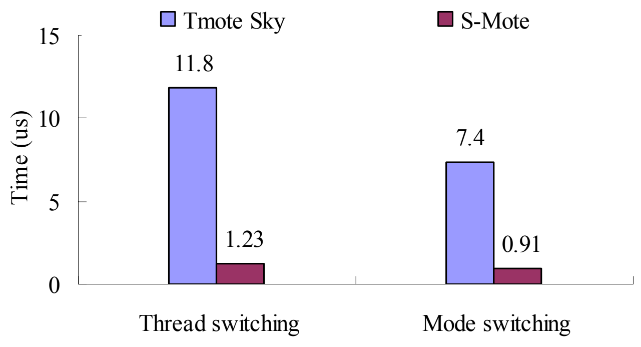


Fig. 9. Kernel performance for mode and thread switching

Fig. 9 shows the threading overhead. The figure shows that S-Mote has little overhead during operation, which affects system performance. S-Mote spends 1.23us in thread-switching, which means that the system is able to switch threads 813,008 times per second.

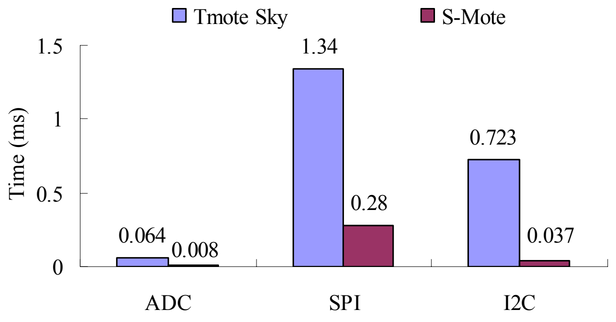


Fig. 10. Kernel performance for data sensing

Sensing is a major portion of system operation. In general, data sensing is obtained in three different ways: ADC, I2C, and SPI. ADC converts an analog signal to a digital value, and I2C reads the digital value directly. SPI read a sensing data from a serial. MIC, strain gage, and photo sensor use ADC, and temperature and humidity sensors use I2C. SPI is used by a device that supports a communication protocol such as a GPS device. Fig. 10 shows the sensing time needed to read the data from each type of sensor interface. The high-rate sampling frequency of ADC significantly affects system performance. As clock speed is a major performance factor to read data from ADC devices, the CC2430 has better performance compared to the MSP430-based mote.

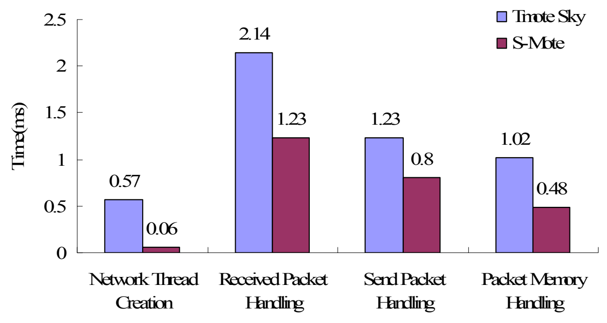


Fig. 11. Packet handling performance

Fig. 11 compares the performance for network packet handling. To evaluate packet handling performance, we traced the ISR operation time. We measured the time from a packet's arrival to data copy to a buffer in the kernel, ignoring data transmission time in order to measure overhead due to system performance. Reducing the memory copy from the radio to the system buffer is the

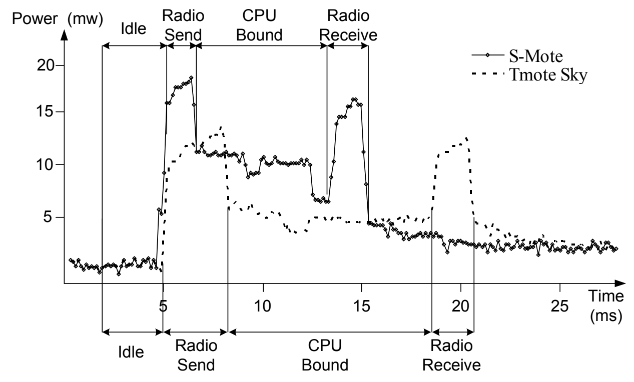


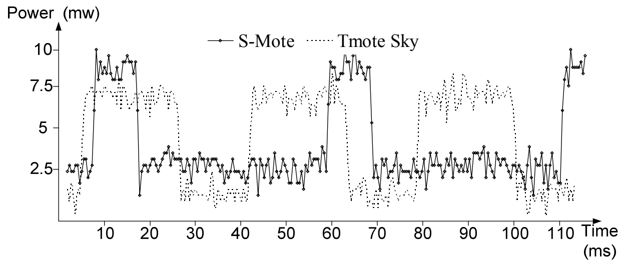
Fig. 12. Power consumption

primary reason for the performance improvement of S-Mote. Generally, a processor that has a high data-rate operates the memory copy fast. However, S-Mote operates the copy instruction in-between an internal-bus system, so it reduces the operation time dramatically.

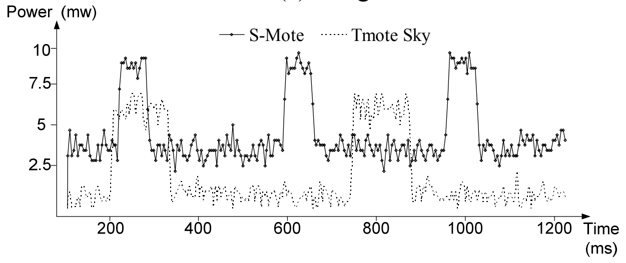
The high clock speed of a processor leads to high power consumption. In a battery-powered sensor network, reducing power consumption is a major issue. Evaluation of the sleep period is particularly important because a sensor system operates and sleeps periodically, and an experiment to measure this aspect was conducted. For fine-grain measurement of the MCU's energy consumption, we measured the total energy cost while running the application, as shown in Fig. 12. The experiments lasted 25 milliseconds, and the application sent a packet after a three second-idle time, operated an intensive calculation and entered a radio-receive mode. Fig. 12 shows the trace of the overall energy consumption and shows that S-Mote consumes more energy but finishes early. The overall energy consumptions are similar to each other.

A 8bit/16bit microcontroller is generally optimized for handling interrupts rather than calculations. However, computation power is important when a sensor node does much in-network processing. Processing power is related to process usage and affects power consumption. We measured operation time and energy consumption while a sensor node operates a trilateration algorithm [22], which is frequently used in a tracking application. A floating point calculation is also important for an application using mathematical calculations. The microcontroller for a sensor node typically does not support the floating point calculation, so a compiler links a corresponding library in compile time. Experiments were conducted for both integer and floating point calculations, and the results are shown in Fig. 13. Compared to MSP430-based hardware, the CC2430 runs the calculation quickly and enters an idle mode early, hence reducing energy consumption.

S-Mote consumes little energy with a short operation time, while Tmote Sky consumes slightly little energy during integer calculation. To explain this difference, we analyzed four arithmetic operations (addition, subtraction,



(a) Long



(b) Float

Fig. 13. Performance trace of the trilateration algorithm

multiplication, and division) in each platform. The MSP430 has large cost for operating a multiplication operation due to its RISC architecture, which operates many instructions for multiplication. A microcontroller using RISC architecture does not have an advanced technique, such as a pipeline technique, so many operations are needed for calculation. The CC2430 has a benefit in operating mathematic operations because they are built upon CISC architecture. When an application requires lots of computation, the CC2430 is a better choice, because of better throughput and energy consumption than the MSP430. The trilateration code has a loop portion of three percent, calculation thirty percent, and variable allocation and shift operation sixty-five percent. The code has a small amount of multiplication operation.

Our experimental results show that S-Mote reduces overhead while operating multi-threading and has better throughput and radio transmission over the traditional sensor node, especially in operating ADC devices and sampling rate. S-Mote consumes slightly more energy, but the shortcoming can be overcome. S-Mote can be a reasonable platform for sensor networks when operating with a proper idle time scheduling, even where high-rate sample frequency is required.

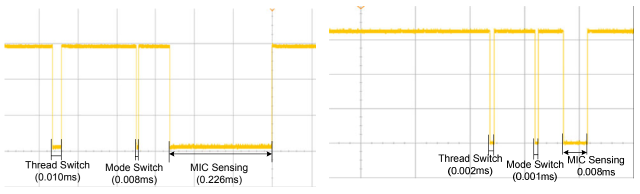
### *B. Running DSL Application*

To evaluate the feasibility of S-Mote for sensor network application and to understand the relationship between component analysis and real-world applications, we implemented the distributed sound localization (DSL) application [18]. DSL is a sensor network application that detects the location of an acoustic source. The system requires high-performance sensor nodes for MIC sampling, data processing, and networking. Therefore, energy issues must be taken into account for this application. We first describe the mechanism of the DSL application. DSL runs over a sensor network with randomly deployed nodes. Each node uses an inexpensive MIC sensor. The base station acquires the real-world position of the sound source, which is calculated by the network nodes. The system dynamically establishes a group to perform a distributed algorithm, which is used for locating the acoustic signal generated within the sensor network at a random time. The group is formed with nodes that have listened to the same acoustic signal. In the group, one leader, nearest to the source, is elected. The members of the group exchange data about the detected acoustic signal with each other. Based on the data, the nodes estimate the source location and, finally, report the result to the base station.

The evaluation environment is a 5 X 5 grid with randomly deployed S-Motes. The sampling rate for detecting the acoustic source is 932Hz, which is the maximum sampling rate for Tmote Sky. To analyze the microscopic performance, we divide the function of the DSL application into several parts: MIC sampling, grid forming, leader election, and localization calculating, and set up trace points at each function, monitored with a oscilloscope.

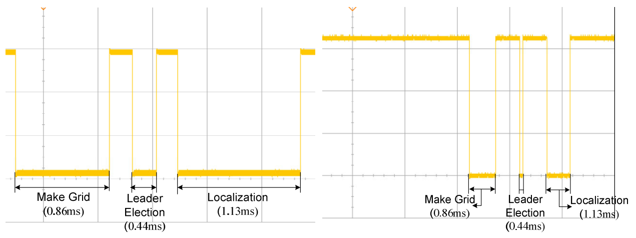
We first measured the MIC sampling overhead that is considered to be the largest overhead caused by repetitive sampling. We created a sampling thread separated from other functions, such as the localization algorithm, and measured the overhead when this thread is operational. Fig. 14 shows the sampling overhead. Fig. 14 (a) shows the execution time of the kernel function when the sampling thread is executed on Tmote Sky, which spends most of its time reading the MIC data and some time on thread and mode switching. Fig. 14 (b) shows the result for S-Mote. The execution time of the kernel is reduced when performing sensor reading, mode switching, and thread switching. The time for sensor reading is greatly decreased whereas mode switching and thread switching show minor enhancement. This is the result of a single execution; hence, the overall performance will increase significantly if the system is repeated for 932 times, for example.

To evaluate the location estimation part, we measured the execution time of the grid, leader election, and localization. We performed an evaluation using pre-logged data according to the worst-case scenario to minimize network delay. As a result, we acquired data, shown in Fig. 15, that show the performance enhancement of S-Mote. The localization performance is greatly improved compared to the other parts. We can



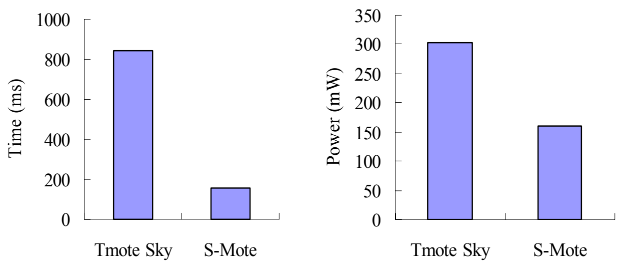
(a) Tmote Sky (b) S-Mote

Fig. 14. Comparison of sampling thread



(a) Tmote Sky (b) S-Mote

Fig. 15. Comparison of localization exaction time



(a) Execution time (b) Power

Fig. 16. Execution time and power consumption of the DSL application

guess from the previous analysis in Section 4.1 that the localization algorithm, which frequently uses a floating operation, showed significant enhancement.

Then, we measured the power consumption for one second to estimate the total system performance. We obtained the result of the average execution time of a total of 10 executions. Fig. 16 shows the performance and the energy consumption of each system. Fig. 16 (a) shows that Tmote Sky takes 843ms to perform a localization calculation whereas S-Mote needed only 112ms for the same action. This reflects the difference in energy consumption. In Fig. 12, S-Mote's power consumption is higher than Tmote Sky's, but Fig. 16 (b) shows that the total power consumption of DSL executed on S-Mote was lower. This is due to the shorter execution time of S-Mote, which allows for an extended idle time.

To observe the effect of the enhanced S-Mote performance, we measured the maximum sampling rate, which is shown in Fig. 17. The results show that S-Mote

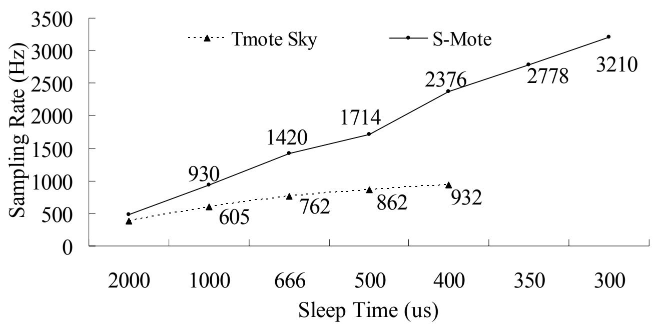


Fig. 17. Ideal sampling rate and real sampling rate on Tmote Sky

provides an almost linear MIC sampling rate and can perform over 2,500Hz sampling because of its high performance. This is the sampling rate that provides localization error under 3cm in DSL. Referring to the previous experiment, S-Mote provides a high MIC sampling ability due to the fast ADC reading, fast mode switching, fast thread switching, and fast network packet processing. These performance enhancements provide a reliable and energy-efficient development system for the DSL application.

## V. Conclusion

In this paper, we described the development of S-Mote sensor nodes based on the CC2430 microcontroller and implemented the RETOS operating system. We conducted various experiments to analyze the feasibility of S-Mote for use with sensor network applications. We implemented performance benchmarks and the DSL application and evaluated the feasibility of the system for real-world sensor network applications. Our experiments showed that S-Mote proved to be a useful platform for complex applications such as DSL, because of S-Mote's high performance and low power feature. CC2430's standard radio and instructions may act as merits for easy system development for sensor nodes.

For future work, we plan to study low power operation to enhance the energy efficiency of S-Mote. Our plan also includes the implementation of various kinds of sensor applications, from simple monitoring applications to complicated applications, to further evaluate the suitability of S-Mote.

## Acknowledgments

The authors would like to thank Mr. Hee Tae Jung, of Yonsei University, for the help he provided in the preparation of this paper. This work was supported by the Korea Science and Engineering Foundation (KOSEF) through the National Research Lab. The program was funded by the Ministry of Science and Technology (No. M10500000059-06J0000-05910), and the ITRC (Information Technology Research Center) programs of the IITA (Institute of Information Technology Advancement) (IITA-2006-C1090-0603-0015).

## References

[1] N. Xu, S. Rangwala, K. Chintalapudi, D. Ganesan, A. Broad, R. Govindan, and D. Estrin, "A wireless sensor network for structural monitoring," in *Proc. ACM Conf. Embedded Networked Sensor Systems (SenSys 2005)*, Baltimore, 2004, pp. 13–23.

[2] G. Werner-Allen, K. Lorincz, M.C. Ruiz, O. Marcillo, J. B. Johnson, J. M. Lees, and M. Welsh, "Deploying a wireless sensor network on an active volcano," *IEEE Internet Computing, Special Issue on Data-Driven Applications in Sensor Networks*, March/April 2006, pp. 18–25.

[3] I. Vasilescu, K. Kotay, D. Rus, P. Corke, and M. Dunbabin, "Data collection, storage and retrieval with an underwater optical and acoustical sensor network," in *Proc. ACM Conf. Embedded Networked Sensor Systems (SenSys 2005)*, San Diego, 2005, pp. 154–165.

[4] H. Kim and H. Cha, "Multithreading optimization techniques for sensor network operating systems," presented at the 4th European Conference on Wireless Sensor Networks (EWSN 2007), Delft, Netherlands, Delft, Netherlands, 2007, accepted.

[5] W. McCartney and N. Sridhar, "Abstractions for safe concurrent programming in networked embedded systems," in *Proc. ACM Conf. Embedded Networked Sensor Systems (SenSys 2006)*, Boulder, 2006, pp. 167–180.

[6] S. Bhatti, J. Carlson, H. Dai, J. Deng, J. Rose, A. Sheth, B. Shucker, C. Gruenwald, A. Torgerson, and R. Han, "MANTIS OS: An embedded multithreaded operating system for wireless micro sensor platforms," *ACM/Kluwer Mobile Networks & Applications (MONET), Special Issue on Wireless Sensor Networks*, vol. 10, no. 4, 2005, pp. 563–579.

[7] H. Kim and H. Cha, "Towards a Resilient Operating System for Wireless Sensor Networks," in *Proc. 2006 USENIX Annu. Tech. Conf.*, Boston, 2006, pp. 103–108.

[8] H. Shin and H. Cha, "Supporting application-oriented kernel functionality for resource constrained wireless sensor nodes," in *Proc. 2nd Int. Conf. Mobile Ad-hoc and Sensor Networks (MSN 2006)*, Hong Kong, 2006, Accepted.

[9] Crossbow Technology Inc., "MICA2 wireless measurement system ." Available: http://www.xbow.com/Products/productsdetails.aspx?sid=72

[10] Crossbow Technology Inc., "MICAz wireless measurement system." Available: http://www.xbow.com/Products/productsdetails.aspx?sid=101

[11] J. Polastre, R. Szewczyk, and D. Culler, "Telos: Enabling ultra-low power wireless research," in *Proc. Int. Conf. Information Processing in Sensor Networks: Special Track*

*on Platform Tools and Design Methods for Network Embedded Sensors (IPSN/SPOTS 2005)*, Los Angeles, 2005, pp. 364–369.

[12] C. Park, J. Liu, and P. H. Chou, "Eco: An ultra-compact low-power wireless sensor node for real-time motion monitoring," in *Proc. Int. Conf. Information Processing in Sensor Networks, Special track on Platform Tools and Design Methods for Network Embedded Sensors (IPSN/SPOTS 2005)*, Los Angeles, 2005.

[13] D. Lymberopoulos and A. Savvides, "XYZ: A motion-enabled, power-aware sensor node platform for distributed sensor network applications," in *Proc. Int. Conf. Information Processing in Sensor Networks, Special track on Platform Tools and Design Methods for Network Embedded Sensors (IPSN/SPOTS 2005)*, Los Angeles, 2005.

[14] Intel Coperation, "Intel mote." Available: http://www.intel.com/research/exploratory/mo tes.htm

[15] Nordic VLSI," 2.4GHz RF tranceiver with a 8051 microcontroller."Available: http://www.nvlsi.no/index.cfm?obj=product& act=display&pro=79

[16] Chipcon Cooperation, "1000 MHz CMOS CC1000 RF tranciever." Available: http://www.chipcon.com/index.cfm?kat_i d=2&subkat_id=12&dok_id=55

[17] Chipcon Cooperation, "CC2430/31A ZigBee(TM) low-cost, low-power System-on-Chip (SoC) solution" Available: http://www.chipcon.com/index.cfm?kat_id=2&subkat_id=12&dok_id=240

[18] Y. You and H. Cha, "Scalable and low-cost acoustic source localization for wireless sensor networks," in *Proc. Int. Conf. Ubiquitous Intelligence and Computing (UIC 2006)*, Wuhan and Three Gorges, 2006.

[19] H. Park, H. Cha, E. Jung, and S. Choi, "Strain-based structural health monitoring for high-rise building using wireless sensor network," in *Proc. Int. Conf. Damage Assessment of Structures(DAMAS)*, Torino, 2007, *in summation*.

[20] S. Choi and H. Cha, "Application-centric networking framework for wireless sensor nodes," in *Proc. Int. Conf. Mobile and Ubiquitous Systems: Networks and Services (MOBIQUITOUS)*, San Jose, 2006.

[21] Chipcon Cooperation, " CC2430DK Development Kit," Available: http://www.chipcon.com/index.cfm?kat_id=2&subkat_id=12&dok_id=244

[22] W. Jung, S. Shin, S. Choi, and H. Cha, "Reducing congestion in real-time multi-party tracking sensor network application," in *Proc. Int. Workshop RFID and Ubiquitous Sensor Networks*, Nagasaki, 2005.