Sensor node controller
Powering the future of connected sensors

Author(s)	Marios Iliopoulos, Fotios Kerasiotis, Nikolaos Moschopoulos
Revision	Rev. B
Release Date	February 2022

Dialog Semiconductor

L. Katsoni and Achileos 8,
17675 Kalithea Athens, Greece
Greece

Phone: +30 210 93 10 580
Fax: +30 210 93 10 581

Neue Straße 95
73230 Kirchheim/Teck
Germany

Phone: +49 7021 805-0
Fax: +49 7021 805-100

Introduction

Today's portable devices learn about their surrounding environment through continuous activity monitoring and contextual awareness. To achieve this, the devices integrate more and more sensors and peripherals, generating lots of data. This in turn makes the integration of more powerful CPUs necessary to perform the increasing amount of computations. At the same time, design size, cost and power consumption must be reduced without sacrificing the final product's evolving feature requirements.

The concept of a ***sensor hub*** is increasingly being adopted in today's SoC designs to meet the requirement for "always on" sensor / peripheral access and control (even at high rates), without increasing power consumption and design cost. Sensor hubs may be small CPU cores that can interface with sensors / peripherals and act as offload engines for the power-hungry main processor by performing background operations and "waking" the main processor only when needed.

Typical processor-based architectures for sensor acquisition

A typical microcontroller-based architecture applied in systems that integrate multiple sensors consists of the following components [1]:

a) A microcontroller unit (MCU) – also called the processing subsystem, an MCU controls the operation of all constituents within the system and processes data. It includes a processor, an internal or an external memory, and all peripherals and subsystems required for local data processing. In a typical MCU-based architecture the controller performs all sensor data collection, processing and storage.

b) Sensor elements (or sensing subsystem) – a set of sensors that can be any combination of passive or active, digital or analog. These convert input information from the external environment into electrical signals. In most applications, sensors are used to monitor motion, light, pressure, vibration, flow rates, temperature, ventilation, electricity, etc. Commonly, sensor elements generate voltage or current signals at their outputs. These signals are usually amplified and digitized through an analog-to-digital converter, before data is processed, stored and transmitted.

c) Radio –a short-range transceiver which provides wireless communication with the host.

d) Power subsystem – usually connected to a battery or an energy harvester. This subsystem acts as a controllable unit that switches the power supply of the system's building blocks on and off individually. It is usually a software block within the MCU software. The power subsystem is responsible for providing the right supply voltage to each individual hardware component.

In more sophisticated microcontroller-based architectures with multiple sensors, intelligence is integrated in hardware for controlling the various subsystems. For example, time- and power-consuming data transfers from sensors to the memory can be offloaded from the processor to a direct memory access (DMA) unit. A power management unit (PMU) can also be programmed to react to specific events and power down various subsystems, such as peripherals, sensors and radio.

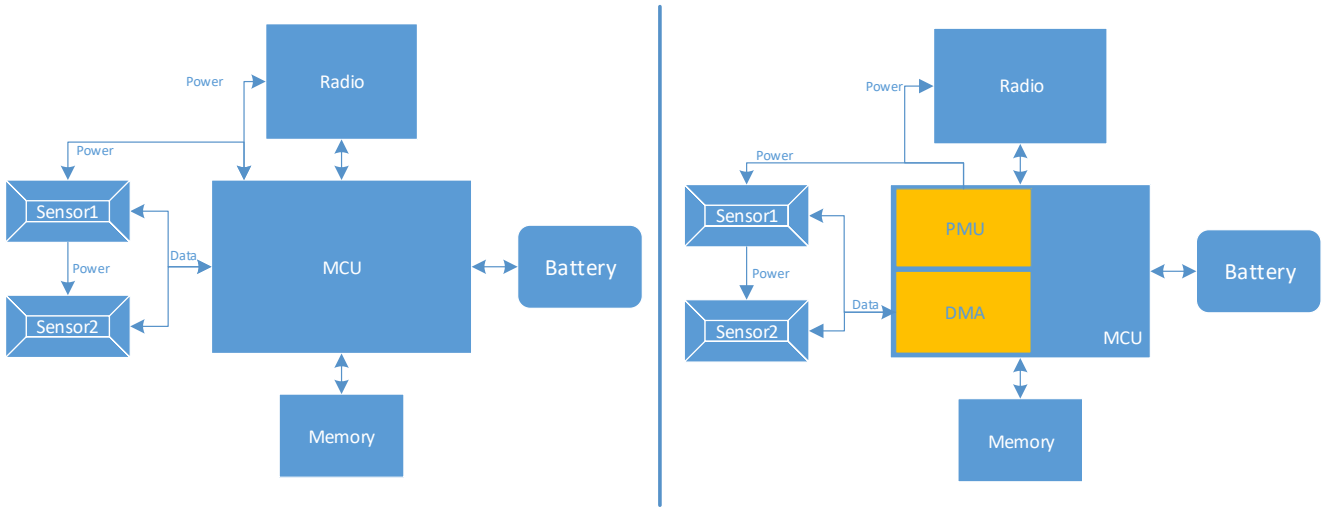

Figure 1 Typical microprocessor-based architectures

The aim of such advanced architectures is to reduce the time that the main MCU stays awake as much as possible. Tasks that would normally require the MCU intervention are now executed by the intelligent subsystems. However, the problem remains that the MCU should wake-up every time an event comes from the sensors, radio or various subsystems, as it is the only component which can implement some logic for handling those events.

Techniques for optimizing power – adding a sensor node controller

In order to extend the lifetime of systems with multiple sensors, a wide variety of techniques for minimizing energy consumption have been proposed [2], [3]. Some deal with saving energy at the *Media Access Control* level [4], [5], others with dissemination of data aggregations or fusion [6], [7], and others use chip design optimization techniques such as on-chip power gating [8] or dynamic voltage scaling [9].

This paper introduces a different technique to optimize power usage in systems with multiple sensors while minimizing wake-up time of the main processor for data acquisition. The technique is based on integrating a sophisticated hardware state machine that can take over repetitive tasks such as sensor polling and reading from the main processor, thus implementing the notion of an integrated low-power sensor hub. A dedicated hardware state machine can wake up much faster and uses fewer blocks to transfer data from sensors / peripherals to memory and vice versa, while the main processor remains in sleep. Additionally, the sensor hub can perform simple operations on data, so the main processor only needs to wake up when complex data manipulations are required.

A good example of this approach is the sensor node controller (SNC) hardware block integrated in Dialog Semiconductor's DA1469x Bluetooth Low Energy System-on-Chip solution, [10]. The SNC is a tiny hardware state machine capable of running microcode (µcode) consisting of a limited instruction set that enables developers to manipulate communication controllers (i.e. SPI, I^2C etc.), sensors and peripherals. It can operate autonomously without the rest of the system being awake by employing its minimum instruction set. This allows it to perform many operations – such as polling sensor status bits, comparing register to memory address contents (values), transferring data from communication interfaces to system RAM and branching on comparisons – while dissipating minimal current.

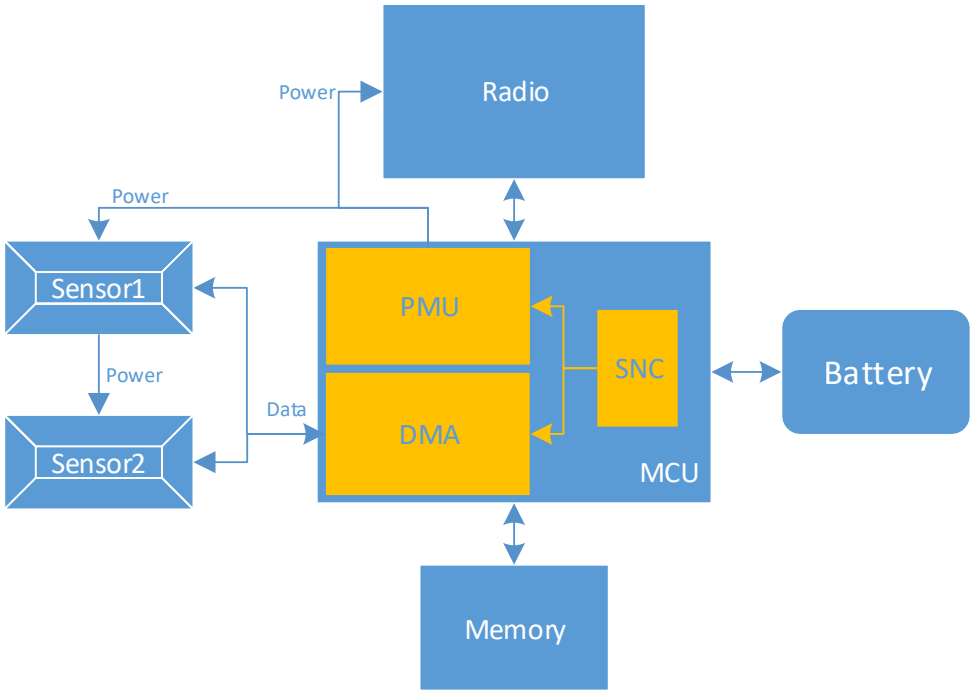

Figure 2 Application processor architecture with a sensor node controller peripheral

The main features of the SNC (figure 3) are:

- A sensor node instruction set (SeNIS) of 10 instructions for µcode generation, tailored for:
  - easy creation of pointers to memory buffers
  - polling of serial interface status bits
  - comparing thresholds
- System RAM used for both µcode storage and data
- DMA capabilities for transferring data directly from communication interfaces to the system RAM
- Direct access to all peripherals and registers
- Immediate execution after interrupt triggering and domain power up (e.g. timer, GPIO) via PMU
- SNC to main processor notification and vice versa

While residing in the same power domain as all the communication interfaces (SPI, I^2C and UART), the SNC can also control other power domains. It executes a µcode residing in the system RAM, where the SNC has a direct memory connection; operates at system clock speed; and can generate an interrupt to notify the PMU that all operations are complete so that the entire system can be powered off.

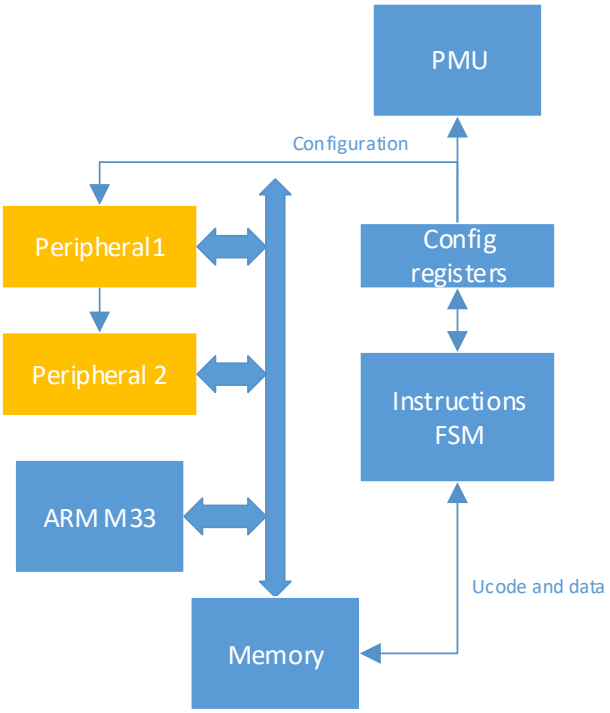

Figure 3: Sensor node controller block diagram

The major advantages of an architecture that uses such dedicated hardware to perform sensor and peripheral data manipulation are:

- Power consumption savings because the main CPU sleeps for a longer time
- MIPS savings as the CPU doesn't have to access slow peripherals or perform simple data manipulation

However, there are some disadvantages. The very simple instruction set of the SNC's programming model allows only basic operations. Also, programming the SNC requires the use of assembly language. Finally, because the SNC is a very simple block, debugging can become complex.

Power consumption and MIPS savings

We have performed a variety of measurements that demonstrate the power consumption and MIPS savings of using an SNC. The measurements were carried out on Dialog's ARM Cortex-M33 based DA1469x SoC and compared transactions that used just the main CPU with identical transactions carried out with support from the SNC. The first set of measurements focuses on power / MIPS savings when accessing slow peripherals (like I^2C and SPI) commonly used for sensor readings. The second set looks at real application use cases that employ the Bluetooth Low Energy (BLE) communication block included in the DA1469x.

Table 1 depicts the energy consumption when transferring small SPI or I^2C data transactions like writing / reading 16 / 128 bytes.

SPI@8 MHz 16B write / 128B read	Duration (ms)	Charge (µC)	Charge diff.
SNC	0.5	1.24	
CM33 + DMA	1.5	2.95	138 %
I²C@High speed 16B write / 128B read	Duration (ms)	Charge (µC)	Charge diff.
SNC	2.12	2.44	
CM33 + DMA	2.67	4.55	86 %

Table 1: Energy consumption when transferring small data transactions (numbers are for 1.8 V, 8-bit transfers)

Looking at the time required to perform a transaction shows more savings possible with the SNC compared to the CPU (CM33), as shown in table 2. These time savings translate to MIPS savings as the CPU may need to perform busy waits.

Master	Durations (µs)						
	CS - Write	Write	Write - Read	Read	Read - CS	Total	Time diff.
SNC (8-bit mode SPI)	13.98	26.1	16.68	209.64	5.5	271.9	
CM33 (DMA, with adapter)	46.96	18.06	139.18	145.04	87.3	436.5	60.5 %

Table 2: Time required by the SNC to write/read 8 bits of data and the respective time for the CPU

For real life use cases, we compare the energy consumed when advertising every 1500 ms (table 3), and 500 ms (table 4), while performing an accelerometer sensor reading every 100 ms using SPI.

Advertising (every 1500 ms) & sensor reading every 100 ms – for 15 sec	Charge (µC)	Saving
SNC	223.6	
CM33	374.8	40.3 %

Table 3: energy consumption when advertising every 1500 ms, sensor reading every 100 ms using SPI

Advertising (every 500 ms) & sensor reading every 100 ms – for 15 sec	Charge (µC)	Saving
SNC	255.6	
CM33	406.8	37.2 %

Table 4: energy consumption when advertising every 500 ms, sensor reading every 100 ms using SPI

In complex applications, with more than one sensor being accessed, the overhead using the MCU becomes even larger due to cache misses and task switching that must also be taken into account.

Reducing programming complexity

As mentioned earlier, one of the biggest challenges when using an integrated sensor node controller is to ensure easy programming, debugging and full exploitation of the underlying system capabilities in a similar manner as with a common MCU. The challenge's main aspects are:

- Providing a developer-friendly programming abstraction for efficiently controlling the underlying functionality that corresponds to driving the communication interfaces to connected sensors / peripherals and communication with the main CPU
- Leveraging the 10-instruction assembly-style programming and providing higher-level programming constructs to ease and speed-up software development
- Support complete and system-as-a-whole debugging – instead of debugging each CPU core separately which may fail to detect faulty system behavior when the cores operate in parallel

To address this a complete, easy-to-use programming framework is needed. It must include abstractions and procedures that extend the paradigm of concurrent operating system tasks to the corresponding (parallel) processing performed on the SNC. Such a programming framework has been developed for the Dialog DA1469x solution (Figure 4). It has the following characteristics:

- Simplified SNC µcode development
    - by defining a C-like programming language based on SeNIS, so that both assembly and C-like programming can be supported
- "Hybrid" coding model for programming
    - covering code development for both SNC and main processor context in the same source and header files – a C pre-processor is used for defining the SeNIS-based language constructs in a developer friendly way
- Abstractions of the underlying mechanisms and functionality related to driving communication interfaces, exchanging SNC notifications, and manipulating system resources with the main processor as a complete and easy to use set of API procedures / C-like functions
- Observability and an advanced debugging mechanism applied to both SNC and main processor execution contexts at the same time

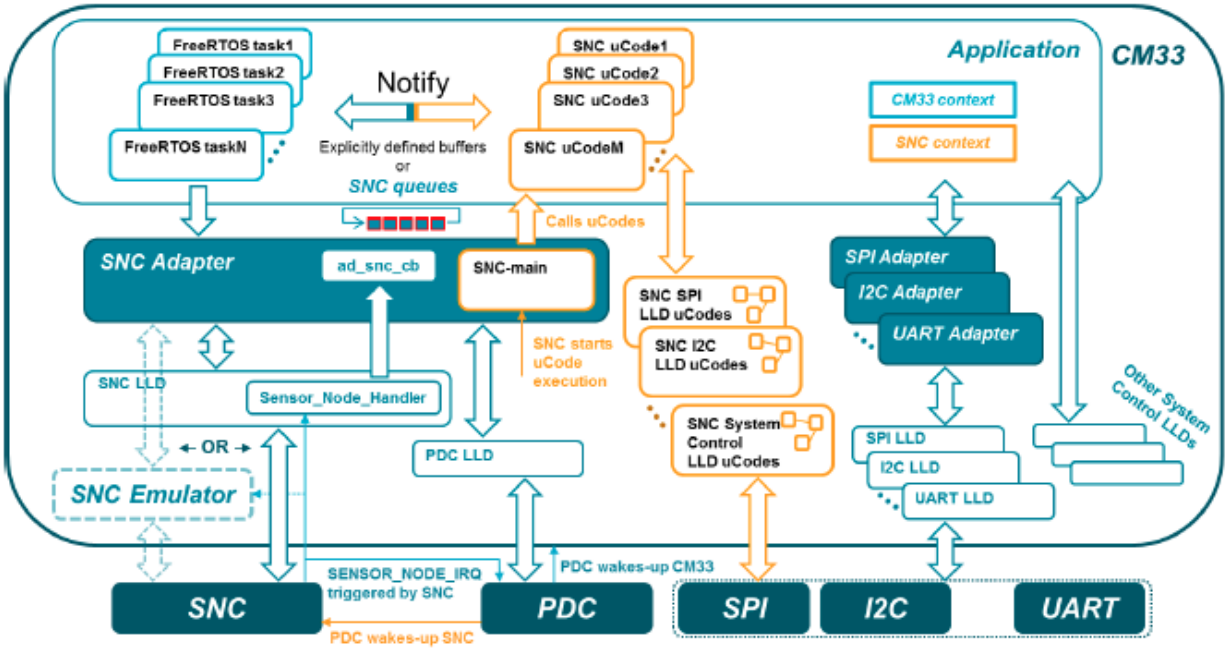

Figure 4: Block diagram of SNC programming framework

The SNC programming model characteristics can be summarized as follows:

- A DA1469x application includes processes executed by OS tasks and SNC ucodes in parallel.
- An SNC µcode is registered to – or unregistered from – the DA1469x system by the SNC adapter, which creates a list of SNC µcode each triggered by a specific PMU event
- The SNC adapter employs a special SNC µcode that both implements scheduling of the registered SNC µcode execution based on the respective µcode list and controls the SNC hardware module via its low-level driver
- A set of SeNIS-based construct pre-processor macros are defined, resulting in a set of assembly and C-like language constructs for SNC µcode development
- A set of low-level driver SNC µcode are provided which can be used to drive communication peripherals such as SPI, I^2C etc.
- A mechanism is provided for operating system tasks and SNC µcode comprising an application to exchange:
    - notifications
    - data (i.e. SNC queues)

On top of the above constructs that allow easy programming of SNC functions, the following are provided to support debugging:

- A mechanism for debugging SNC µcode, using SNC breakpoints and step-by-step debugging regions
- An SNC emulator, instead of the SNC hardware module, to improve and simplify the SNC µcode debugging process

Conclusion

This paper introduces a new architecture to minimize power consumption in portable systems that integrate several sensors and peripherals. The architecture uses a sophisticated hardware state machine to offload repetitive tasks such as sensor/peripheral polling and reading from the main processor. It provides great advantages compared to other architectures in terms of power consumption and MIPS optimization, but makes the programming model more complex. Hence, a developer friendly programming framework is also introduced to overcome this problem.

References

[1] Goran Nikolić, Mile Stojčev, Zoran Stamenković, Goran Panić, Branislav Petrović, “Wireless Sensor Node with low power sensing”, Electronics and Energetics Vol. 27, No 3, September 2014, pp. 435 - 453
[2] V. Raghunathan, S. Ganerival, and M. Srivastava, "Emerging Techniques for Long Lived Wireless Sensor Networks", IEEE Communication Magazine, 2006, Vol.41, No. 4, (pp. 130-141)
[3] G. Anastasi, M. Conti, M. Di Francesco, and A. Passarella, "Energy Conservation in Wireless Sensor Networks: A survey", Ad Hoc Networks, 2009, Vol. 7, (pp. 537–568)
[4] W. Ye, J. Heidemann, and D. Estrin, "An Energy-Efficient Mac Protocol for Wireless Sensor Networks," Proc. IEEE Infocom, New York (USA) 2002, (pp. 1567-1576).
[5] M. Al Ameen, S.M. Riazul Islam, and K. Kwak, "Energy Saving Mechanisms for MAC Protocols in Wireless Sensor Networks", Hindawi Publishing Corporation International Journal of Distributed Sensor Networks, Volume 2010 (2010), Article ID 163413, (pp 1-16)
[6] M. Hempstead, N. Tripathi, P. Mauro, G.-Y. Wei, and D. Brooks, "An Ultra Low Power System Architecture for Sensor Network Applications," Proc. 32nd Annual International Symposium on Computer Architecture, Madison (USA) 2005, (pp. 208-219).
[7] A. Boulis, S. Ganeriwal, and M. Srivastava, "Aggregation in Sensor Networks: An Energy Accuracy Trade-Off", Ad Hoc Networks, Vol. 1, 2003, (pp. 317–331)
[8] G. Panić, Z. Stamenković, and R. Kraemer, "Power Gating in Wireless Sensor Networks", Wireless Pervasive Computing, 2008. ISWPC2008. 3rd International Symposium on Santorini, Greece, May 2008, (pp. 499-503)
[9] T. Burd, and R. Brodersen, "Energy Efficient Microprocessor Design", Kluwer Academic Publishers, Norwell MA, USA, 2002
[10] Dialog Semiconductor, “DA1469x Datasheet”

