SNOW^5: a modular platform for sophisticated real-time wireless sensor networking

Marcel Baunach, Reiner Kolla, Clemens Mühlberger

Department of Computer Science V
University of Wuerzburg, Bavaria, Germany
{baunach, kolla, muehlberger}@informatik.uni-wuerzburg.de

Abstract This technical paper describes the architecture of SNOW^5, a low power sensor node for ad hoc wireless sensor networking (WSN). The node's versatility turns it into an ideal tool for WSN research, educational and commercial applications. After explaining the basic concept, the hardware design will be shown in detail. An extensive comparison to existing nodes will be given regarding technical aspects like application specific customization, power consumption, wireless communication and operating system design. Outstanding features and internals of the newly developed real-time operating system SMARTOS for sensor nodes will also be illustrated.

1 Introduction

The minimization of powerful information technology is essential for its integration into daily life. The deployability of small-sized autonomous systems into almost any object supports humans as well as machines and leads to increased convenience, performance, security, etc. However, one must always take into concern, that minimalistic computers have low performance due to very restrictive requirements like ultra low power consumption [1]. Thus, the demand for wireless networks of small autonomous devices increased heavily within the last few years. A so called wireless sensor network (WSN) uses the combined power of many small devices to solve even complex problems under exceptional circumstances for which a single device was too weak. Nevertheless, the successful coordination and interaction of these sensor nodes is a hard problem, comprising research areas like communication, self-organization, fault-tolerance, distributed algorithms and low-power design in both hardware and software.

As mentioned in [1–3], several sensor nodes already exist. Nevertheless, after careful comparison of some of them, we developed the new sensor node SNOW^5 to mainly achieve a higher versatility and more flexible communication at significantly improved power consumption (see section 2). This paper describes in section 3 which hardware components were selected for this purpose and which effects each individual decision implied on the node's realization. The circuit design follows in section 4 and examples for hardware extensions for SNOW^5 are shown in section 5. A detailed comparison to other nodes follows in section 6. The new real-time operating system SMARTOS for small devices like sensor nodes will be presented in section 7. Some application scenarios are shown in section 8, before a short conclusion and an outlook to future work (section 9) closes this technical paper.

2 Goals, requirements and features

Important requirements and fundamental factors for WSN applications like fault tolerance, scalability, deployment and power consumption can be found in [4, 5]. As mentioned in [6], we mainly focused on following aspects when designing SNOW^5 (see figure 1):

modularity and customizability Most important for our research is the support of additional analog and digital devices. Thus, an easy to handle design for stackable daughter boards is mandatory to propagate arbitrary signals and buses like SPI or I^2C to extensions when required.

energy efficiency Low power consumption allows a WSN to operate properly for a long time even with limited power supplies like batteries. Therefore not only economical hardware components are required but energy saving modes must also be supported by software.

flexible communication Since wireless communication is obligatory in WSN, we use a low power multi channel radio transceiver for research on communication protocols. For versatile communication and debugging, SNOW^5 is additionally equipped by some serial as well as JTAG interfaces. Further communication transceivers simply can be mounted by means of extension boards.

complete autonomous operation and robustness While cooperation between the sensor nodes improves the performance of the whole WSN, a single node should be able to operate autonomously for a specific amount of time, and if it is just for logging sensor data.

compact design To be small-sized, all electronic components are SMD size only. But for a comfortable and fast prototyping of diverse extensions on standard bread boards, a $2.54\ \mathrm{mm}$ grid was chosen for the node's headers. Thus, we found a reasonable trade-off between the overall size and convenience.

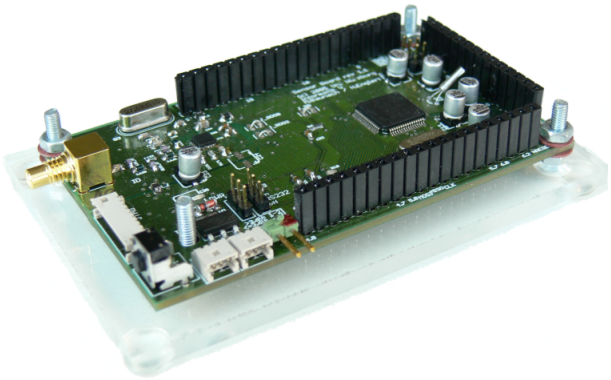

Figure 1. A SNOW^5 node, screwed onto acrylic glass

3 Hardware components and specifications

Our goals described above mainly outline the hardware components to use. This section will explain the specific components applied in detail and illuminate why each one was chosen (see figure 2). In addition to the requirements mentioned above one should always keep two things in mind:

1. A small but powerful operating system will be required to handle all the different hardware devices and to provide a simple to use interface for the application running. A detailed description of our operating system SMARTOS can be found in section 7.
2. All devices must provide low-power modes and accept a common operation voltage to avoid the use of multiple power supplies or DC/DC regulators. In our design we support operation voltages from about 3 V with all devices active, down to 1.8 V with only MCU and radio active.

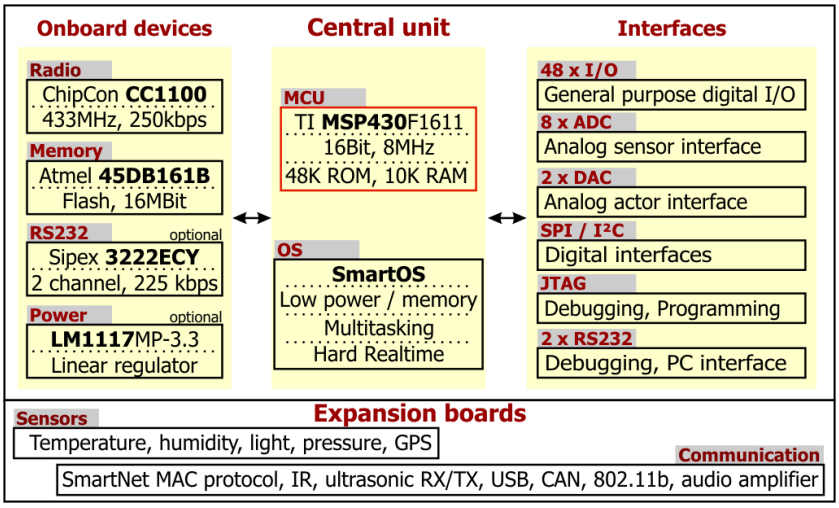

Figure 2. SNOW^5 – a rough overview

The central unit of a sensor node is its microcontroller. The design of SNOW^5 supports the 16 bit ultra low power MCU family MSP430 [7] from Texas Instruments (TI). The device we prefer is the MSP430F1611 [8] with 48 kB of flash memory (ROM) and 10 kB of RAM. It features five operation modes with different energy characteristics from 0.2 µA in standby mode up to 2.5 µA (9 µA) in active mode at 8 MHz and 2 V (3 V). Its supply voltage ranges from 1.8 to 3.6 V (see table 1), so battery-operation will be no problem. We are also ready for TI's pin-compatible successor MCU with 1 MB of on-chip memory, to be released in the near future.

Other reasons for choosing the MSP430 are various on chip peripherals which come in handy for research and education and which can be used by expansion boards through

the pin headers: 8×12 bit A/D converters, 2×12 bit D/A converters, 2×16 bit timer with capture/compare, brownout detector, hardware multiplier, 6×8 bit general purpose digital I/O ports and integrated bus protocols for serial communication like $2 \times$ SPI, $1 \times$ I^{2}C and $2 \times$ RS232. These features allow comfortable usage of external analog and digital components like sensors and actors. The clock frequency is software adjustable from 32 kHz up to 8 MHz using additional external crystals or an internal DCO. We don't recommend using the internal DCO because it increases power consumption and is subject to considerable fluctuations upon temperature changes.

As wireless communication unit we have chosen the ChipCon CC1100 radio transceiver [9] whose base frequency is adjustable between 300 to 348, 400 to 464 and 800 to 928 MHz. Its supply voltage ranges from 1.8 to 3.6 V, so battery operation is adequate. Beside one idle mode it offers one TX and two RX modes. A special feature for which we selected the CC1100 is the Wake-on-Radio RX mode which can be configured to require only 1.8 µA and upon detection of a preamble automatically enters full RX mode (14.5 mA). Beside this optimal DC characteristics it supports multi-channel operation for improved collision avoidance and offers an 8 bit digital *received signal strength indication* (RSSI) and *link quality indication* (LQI) output. This can be used to adjust TX power (-15 to $+10$ dBm) for dynamic cell sizes or even to roughly localize the node relative to its neighbours as proposed in MoteTrack [10] or RADAR [11]. Other features worth mentioning are the optional 8 bit hardware address check simplifying energy saving and MAC protocol implementation, the various selectable modulation formats (see table 1) supporting data rates from $1.2 - 500$ kbit/s and the two 64 byte RX/TX buffers allowing variable packet lengths as well as MCU-independent reception and transmission. Remarkable hardware specifications like its fully digital design allow the extremely small footprint, the few required additional electronic components and its comfortable interface via SPI protocol.

For data storage we use the non-volatile Atmel data flash AT45DB161B [12] with SPI interface and $2.7 - 3.6$ V supply voltage. Its power consumption is 2 µA in standby mode, 4 mA while reading and 15 mA while writing. The capacity of 16 Mbit (byte addressable) is sufficient even for long-term data logging. The reason for selecting this very device were the two 528 byte data buffers in combination with the ready/busy-indicator allowing a very smart implementation of an embedded file system that saves valuable RAM within the MCU [13].

Communication with a PC is optionally available via two RS232 interfaces attached to the MCU over a two channel signal driver. Since RS232 communication requires ± 12 V signal levels and implies wired nodes, it can be disabled by jumpers when low power operation is mandatory. Thus, except for the supply voltage of the signal driver, its DC characteristics are mostly irrelevant. We selected the Sipex SP3222E [14] as it is inexpensive, has a small footprint, offers data rates from 120 to 235 kbit/s, runs at $3 - 5.5$ V and thus avoids a secondary power source. Its supply current ranges from 1 µA to 0.3 mA. The reason for actually providing a RS232 interface was not only the additional debug or communication channel but also the possibility to easily attach ready-made devices like GPS modules to SNOW[5] which are not yet available as expansion boards.

As the MSP430 provides easy programming and in-system debugging via JTAG (IEEE1149.1), SNOW^5 forwards the JTAG signals to a six pin jack. This option can also be used as communication channel without any extra electronic components.

4 SNOW^5 circuit design

This chapter will describe the SNOW^5 board design in detail. Always keep in mind, that research and education are one of the main application areas of SNOW^5. Therefore, we value flexibility, expandability and convenient debugging higher than small dimensions and specialization. Of course, a smaller and application specific design can easily be derived from the existing one.

Initially it must be mentioned, that nearly each of the MSP430's pins has two exclusive operation modes: it can be used as general purpose (GP) digital I/O or as special function port like A/D converter, D/A converter or timer. Therefore one must carefully select which special function to sacrifice for digital I/O. Regarding this, the interconnection between the applied components was designed as follows.

The data flash and the radio transceiver share the same SPI bus as slaves and are connected to the MSP430 as master over its first USART0. As SPI is a three wire bus (in, out, clock) which selects the active component by an extra individual chip select (CS) signal, five MSP430 ports are required for flash and radio. Three extra pins are used for signaling ready/busy from the flash and two configurable events from radio to the MCU. For the sake of flexibility, these three pins can be individually (dis)connected via jumpers. SNOW^5 provides a short on-board antenna which will be sufficient for many applications as well as the possibility to connect an external antenna via SMC connector.

The RS232 driver is hardwired to both USARTs of the MCU. To release these pins for other purposes, the whole driver IC can be deactivated via jumper and thus it wastes no energy at all. Up to now, the SPI and I^{2}C function of USART1 is still unused on-board and thus available for further extensions.

Power can be supplied to the circuit in two exclusive likewise jumper selectable ways. The first option enables direct supply without any protection against DC polarity reversal or overvoltage. It is intended for autonomous battery operation granting the full voltage range from $1.8 - 3.6$ V. The second option is intended for wired power unit operation. It accepts any input voltage from $4.5 - 20.0$ V DC and transforms it to 3.3 V at max. 800 mA using the DC/DC linear regulator LM1117MP-3.3 [15]. No doubt this regulator consumes additional power but one gains more flexibility when battery operation is not desired and obtains both DC protections mentioned above.

The pin headers and the JTAG port are directly connected to the MCU's corresponding pins for easy accessibility and are left floating with two exceptions: the reference voltage pins VRef+ and VeRef+ can be grounded via jumpers when not in use. This takes into account the power saving guidelines as described in the MSP430 manual [7]. The two RS232 jacks are connected to the RS232 driver IC and provide the corresponding ± 12 V signals when enabled. A reset button is also available.

All remaining electronic components like capacitors, resistors and inductors were selected in their smallest package available. They are required for IC power supply

stabilization, reference voltages, pull up/down, the radio circuit and the RS232 charge pumps.

The versatility of this open design was confirmed by the application of SNOW^5 in several testbeds so far, e.g. the high-precise ultrasonic localization system SNOW BAT [16].

5 Hardware extensions

To achieve the goal of modularity, additional hardware can be stacked onto each sensor node as daughter boards (see figure 3). This allows highly customizable hardware for almost any application and research area. Furthermore, the board layout offers stacking of multiple daughter boards at the same time for dynamic configuration. This concept enables us to avoid placing any sensors directly on the SNOW^5 main board as this would restrict its versatility due to preassigned I/O signals that could be used wiser within some other applications. Besides, we prefer placing sensors where they are supposed to acquire information. Therefore we preserve the option to mount them onto the case where they are in direct contact with the environment they monitor and connect them to SNOW^5 via the expansion headers.

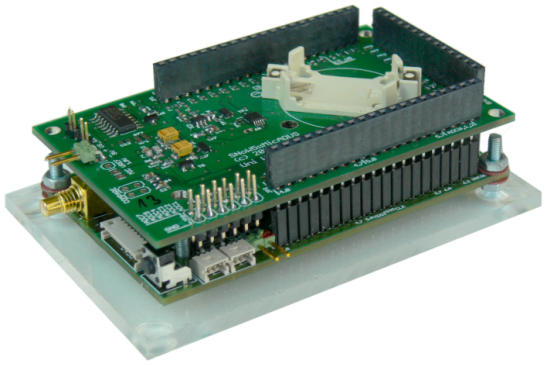

Figure 3. SNOW^5 with stacked SNOW BAT daughter board

Along with various digital and analog signals, these headers provide three power lines for increased flexibility in power supply. One power line propagates the primary 3.3 V power to the daughter boards, the remaining two are free for additional supplies as required by the specific configuration. In this way, several extension boards can share a single power supply unit easily. Beside we use standard bread boards with 2.54 mm grid to facilitate fast and cheap prototyping of new daughter boards.

Quite a number of expansion boards are already available or under development: the MICADUS (Microphone, A/D converters and ultrasonic) extension board for example provides the possibility to directly attach up to six analog sensors for light, humidity, pressure, temperature, etc. to SNOW^5's A/D converters. Due to an acoustic amplifier,

audio applications with a condenser microphone are also realizable with this board. It also features a circuitry for ultrasonic (US) applications like distance measurement. First tests showed that a node can perform self localization within the precision of few millimeters using this expansion (for more information, see [16]).

In the near future a special communication daughter board with *Universal Serial Bus* (USB) and *Controller Area Network* (CAN) bus connectivity can be expected as well as an extension for coarse outdoor localization, orientation and movement detection, with GPS, tilt and hall sensors. On the one hand this can increase accessibility via different communication channels and the accuracy of the node's localization on the other hand.

Sensor Node	**Mica2 MPR410CB**	**BTnode**	**ESB ScatterWeb**	**EYES**	**Telos**	**SNoW^5**
Developer	Crossbow	ETH Zurich	FU Berlin	Univ. of Twente	UC Berkeley	Univ. of Wuerzburg
Date	2002	2004	2005	2003	2004	2005
Microcontroller unit						
IC	ATMega128L	ATMega128L	MSP430F149	MSP430F149	MSP430F1611	MSP430F1611 / F16xx
Speed (MHz)	7.37	7.37	?	5	0.4 - 8	0.4 - 8
Architecture	8 bit RISC	8 bit RISC	16 bit RISC	16 bit RISC	16 bit RISC	16 bit RISC
Flash ROM (kB)	128	128	60	60	48	48
RAM (kB)	4	4	2	2	10	10
Power, active mode (mA)	8	8	3.2	3.2	4	4
Power, sleep mode (μA)	15	15	1.6	1.6	2	2
Wakeup time (μs)	180	180	6	6	6	6
Onboard memory						
IC	AT45DB041B	62S2048U	MC 24LC64	ST M25P40	ST M25P80	AT45DB161B
Type	Flash	SRAM	EEPROM	Flash	Flash	Flash
Non-volatile	yes	no	yes	yes	yes	yes
Interface	SPI	?	I^{2}C	SPI	SPI	SPI
Size (kB)	512	240	64	512	1024	2048
Power, idle (μW)	5	?	0.03	150	150	5
Power, read (mW)	10	?	0.15	12	12	10
Power, write (mW)	37.5	?	0.3	45	45	37.5
Primary wireless communication						
IC	CC1000	CC1000	TR1001	TR1001	CC2420	CC1100
Interface	SPI	SPI	non-SPI	non-SPI	SPI	SPI
Data rate (kbit/s)	38.4	38.4	19.2	57.6	250	500
Modulation	FSK	FSK	OOK,ASK	OOK,ASK	O-QPSK	2FSK,GFSK,ASK,OOK,MSK,QPSK
Frequency (MHz)	433	433-915	868	868	2400	315, 433, 868, 915
Hardware address check	no	no	no	no	yes	yes
Digital RSSI/LQI	no	no	no	no	yes	yes
Power, RX (mA)	7.4	7.4	3.8	3.8	18.8	14.5
Power, TX @ 0 dBm (mA)	10.4	10.4	12	12	17.4	16.1
Low power listen mode (μA)	74	74	1800	1800	–	15
Sleep mode (μA)	0.2	0.2	0.7	0.7	1	0.4
Interfaces / Sensors / Misc						
PC Communication	RS232	Bluetooth / JTAG	RS232 / JTAG	RS232 / JTAG	USB	RS232 / JTAG
Integrated sensors	no	no	yes	no	yes	no
Extension pins (incl. JTAG)	51	55	24	14	16	67
Accessible free Digital I/O	?	21	8	8	13	41
Accessible free ADC ports	?	2	0	8	6	8
Accessible free DAC ports	0	0	0	0	2	2
Accessible buses	SPI, I^{2}C	SPI, I^{2}C	SPI	–	SPI, I^{2}C	SPI, I^{2}C
Accessible DC ports	1	1	1	1	1	1 + 2 (free for expansion)
Recommended OS	TinyOS	– (TinyOS)	– (TinyOS)	PEEROS	TinyOS	SmartOS (FreeRTOS, TinyOS)
Overall DC / physical specifications						
Size (mm $\times$ mm)	32 $\times$ 58	32 $\times$ 58	$\approx$ 45 $\times$ 54	$\approx$ 32 $\times$ 92	32 $\times$ 65	50 $\times$ 85
Supported operation voltage (V)	2.7 - 3.3	3.3 or 3.8 - 5	3 - 26	3	1.8 - 3.6	1.8 - 20
Regulated supply	no	yes	yes	no	no	yes
Power, active mode (mA)	30	$\approx$ 33	12	?	14	8

Table 1. Node comparison table

6 Comparison to other nodes

As addressed before, some sensor boards already exist. Nevertheless, we decided to develop a new node from scratch for a better fulfillment of our demands from section 2. Table 1 compares SNOW^5 in detail to five other nodes available: Mica2 [17], BTnode [18], ESB [19], EYES [20] and Telos [21]. The named sensor nodes are somewhat similar: beside a microcontroller as central unit they all have a flash memory for data storage and a radio for wireless communication. For PC communication they all provide more or less different interfaces. But in detail, there are subtle distinctions.

The first remarkable feature is the MCU. Depending on the application, SNOW^5 can be customized not only with any 64 pin MSP430x16xx controller providing $48 - 60$ kB flash ROM and $2 - 10$ kB RAM but even with TI's pin-compatible successor MCU with 1 MB memory without any modifications on the PCB design (cf. section 3). Compared with the ATMega128L [22] used on Mica2 and BTnode, the MSP430 has a very short wake up time of 6 µs from idle mode and remarkable low power consumption in all operation modes. Furthermore SNOW^5 uses an external crystal instead of the internal DCO for a more precise clock generation. Moreover, the MSP430x16xx is capable of driving analog devices due to its D/A converters qualifying only SNOW^5 and Telos for controlling analog actors directly.

Concerning the onboard memory, ESB comes with a very energy efficient EEPROM. However we prefer flash technology despite of its higher energy consumption because it is available with faster access times and bigger sizes which enables efficient storage of a large amount of sensor data. A volatile SRAM memory as applied on BTnode was never considered, because in case of power failures all data will be lost. Thus, we decided to use a non-volatile flash like Mica2 does, but with 16 Mbit.

As wireless communication is very important but also very expensive within a WSN, an energy efficient radio transceiver is mandatory. The TR1001 [23] used by ESB and EYES is most efficient in pure RX mode but very inefficient in low power listen mode and provides no interface natively supported by the MCU. As successor to the CC1000 [24] (used by Mica2 and BTnode), the CC1100 has a similar energy profile with a much improved low power listen mode which will be used extensively in the SMARTNET MAC protocol (see section 7). It also offers a higher data rate, more modulation modes and digital RSSI/LQI. Its automatic hardware address check allows systematic waking up the MCU upon packet reception.

The dimension of SNOW^5 is rather large, therefore it provides the most complete extension ports making all pins of the MCU accessible for debugging and expansion. Thus, SNOW^5 offers the largest number of buses, digital I/O, ADC and DAC ports. Like BTnode and ESB, SNOW^5 can regulate its primary power supply on demand and additionally propagates up to three different DC voltages to the extension ports. This simplifies adaptation of SNOW^5 via daughter boards: if there are changes in the environmental setup it is sufficient to replace or upgrade the concerned daughter boards and to readjust the software for the given sensor node. These features make SNOW^5 more practical for all-purpose applications than the other nodes shown.

7 Operating system and software guidelines

Some sensor nodes like SNOW[5] support simultaneous usage of various sensors and actors. Thus, the coordination of all installed modules can be hard to manage. Especially on sensor nodes where attached devices often share special units of the MCU, this problem is hard to solve as different timing constraints, exclusive access, etc. must be handled. Manual and application specific coordination may be still possible within small and monolithic programs. Nevertheless, this is unfeasible in large applications where several software modules like drivers and tasks need to cooperate and still require sufficient resources and sometimes even hard real time operation. In this case, an operating system is mandatory to provide an adequate task management and a sophisticated hardware abstraction layer. But at the same time, an operating system for sensor nodes needs to be minimalistic due to the limited resources available. Critical factors like CPU load, ROM and RAM usage must be considered carefully when designing a kernel to run on a microcontroller. Obviously, the implementation of a large scale full featured operating systems for microcontrollers is impossible.

This section introduces the minimalistic operating system SMARTOS (**Sma**ll **r**eal-**t**ime **O**perating **S**ystem) which emanated from YAOS [25]. It offers outstanding features like preemptive multitasking under hard real time constraints, high energy efficiency, event handling, high resolution timestamping and advanced interrupt handling.

After this short motivation basic concepts of our operating system are explained in 7.1. Section 7.2 deals with the timing constraints of SMARTOS, whereas section 7.3 delivers insight to its network layer. Table 2 gives a short survey over already available operating systems like FreeRTOS [26], TinyOS [27] and SOS [28] for MSP430 based embedded systems which therefore would run on SNOW[5], too.

Features	freeRTOS	TinyOS	SOS	SMARTOS
Compiler	mspgcc	nesC@mspgcc	mspgcc	mspgcc
Uses assembly	inline	inline	inline	inline
License	GNU GPL	free	free	none, yet
Binary size	4.4 kB	unknown	$\geq$ 1.6 kB	$\leq$ 1.8 kB
Preemptive scheduler	yes	no	yes	yes
Cooperative scheduler	yes	yes	yes	yes
Real-time	yes	no	yes	yes
Semaphores	yes	yes	yes	yes
Events	yes	yes	yes	yes
Critical sections	yes	yes	yes	yes
Available drivers	no	few	no	few
Message queues	yes	unknown	unknown	yes
Trace visualization	yes	unknown	unknown	under construction

Table 2. Some existing operating systems for MSP430

7.1 SMARTOS concepts

Basic functions and requirements for (real-time) operating systems can be found in [29]. But because of the MCU's little memory, SMARTOS does not support some features well-known by full-grown operating systems, e.g. memory protection, dynamic memory allocation and dynamic task creation. Instead, it supports preemptive tasks with runtime adjustable priorities (up to 255 levels) and priority ceiling, static shared memory and event-based synchronization. It also provides dynamic resource management to coordinate the access of concurrent tasks to available buses and devices. That's why SMARTOS offers the following concepts (cf. figure 4):

Time management is realized by maintaining a 64 bit timeline. Due to the 8 MHz crystal on SNOW[5], a clock cycle takes 0.125 µs. SMARTOS provides a very precise time resolution of 1 µs by internally using a timer of the MSP430 as clock generator.

Events are used to manage and synchronize tasks. They can be triggered by other tasks, resources and interrupts. By waiting for an event, a task will be suspended until the event occurs or a timeout is reached. This avoids busy waiting loops and MCU time can be reassigned to another task. If all tasks are idle, SMARTOS switches to low power mode. The timeout can be specified absolute or relative to the timeline. To manage scheduling each event maintains an individual task queue, sorted by task priority. If an event takes place, two possible things can happen:

1. The first task in the queue will consume the event and resumes.
2. If no task is waiting for the event, it will be saved for future consumption. This means, that the next task waiting for this event immediately consumes it and will not be suspended.

Resources coordinate the mutual exclusion of tasks preventing the simultaneous usage of devices like timers or buses. Their allocation and deallocation is monitored by events. Both must be done explicitly by the same (owner) task. Furthermore priority ceiling among the tasks waiting for a resource avoids thwarting. If a task T_1 with priority P_1 wants to use a resource already allocated by another task T_2 with priority $P_2 < P_1$, the priority P_2 of task T_2 will be increased to P_1 temporarily until T_2 releases the resource.

Tasks are preemptive non-terminating program sections managed by the operating system. They are defined by an entry function, an initial priority and a dedicated stack area of fixed size. Before compilation, this stack size can be calculated by automatic analysis of the task's source code enriched with related annotations like the maximum number of loop iterations. Task scheduling relies on the event system as described above. A task can be suspended by other tasks with higher priority or suspend itself by waiting for an event, a resource or sleeping for a specified amount of time.

IRQ handling is greatly simplified by introducing software interrupts in addition to hardware interrupts. These interrupts automatically demultiplex hardware interrupts shared by several peripheral components. In general, developers can implement handlers for both types of interrupts. Furthermore, SMARTOS automatically saves the timestamp whenever an interrupt occurs (cf. section 7.2).

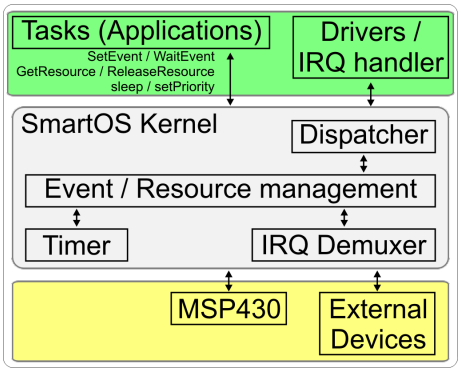

Figure 4. SMARTOS – overview

7.2 SMARTOS timings

Since our goal was to design a real-time system, SMARTOS offers fast task switching and low interrupt latency of 0.75 µs, required for saving program counter (PC) and state register (SR) of the MSP430. Everytime an interrupt occurs, the current timestamp is saved by the operating system within 7.5 µs. As this delay is constant it will be automatically compensated by SMARTOS.

Whenever a context switch happens, the dispatcher saves the context of the current task, chooses the next task from the waiting queue and restores its task context. All in all, this takes only about 20 µs. Thereby, SMARTOS is a minimalistic operating system which grants preemptive multitasking and is capable of real-time operation.

7.3 Usage of SMARTOS, taking SMARTNET as example

SMARTNET is the wireless MAC protocol of SMARTOS and links the operating system and the radio transceiver. It automatically manages the successful transmission of packets by listening for a free transmission channel. Application IDs work like TCP ports and allow SMARTNET to deliver packets directly to a task registered for this ID. The following listing demonstrates the simple usage of SMARTOS and SMARTNET. The example application consists of two tasks, one for periodically sending the broadcast message `"Hello World!"` and the other for receiving messages. The whole application is implemented as follows:

First, both tasks are declared with a priority of 200. Their stack size will be computed right before compilation (lines 1–2). Then, two SMARTNET message buffers of maximum size are allocated (lines 4–5). Two events (lines 7–8) are defined to indicate successful transmission and reception of messages respectively. Both tasks (lines 10–25 and 27–38) initially configure their message buffers before entering an infinite loop. The transmission task for example instructs SMARTNET to send the message (line 19) and then suspends by waiting for the successful transmission event (line 20). The receiving task in contrast is suspended until a reception event occurs, whereon the task displays some information about the received message on the serial console (lines 34–36). The

main function (lines 40–44) starts the whole application by initializing a serial debug console and the operating system environment along with the tasks (line 42).

```
1  SMARTOS_DECLARE_TASK(RadioTx,200,/*<computeStackSize()>*/);
2  SMARTOS_DECLARE_TASK(RadioRx,200,/*<computeStackSize()>*/);
3
4  SMARTNET_DECLARE_BUFFER(RadioTxBuffer, SMARTNET_MAX_DATA);
5  SMARTNET_DECLARE_BUFFER(RadioRxBuffer, SMARTNET_MAX_DATA);
6
7  SMARTOS_DECLARE_EVENT(RadioTxBufferDone);
8  SMARTOS_DECLARE_EVENT(RadioRxBufferDone);
9
10 SMARTOS_TASKENTRY(RadioTx) {
11  SMARTNET_INIT_BUFFER(RadioTxBuffer);
12  strcpy(RadioTxBuffer.data, "Hello␣World!");
13  RadioTxBuffer.event   = &RadioTxBufferDone;
14  RadioTxBuffer.datalen = strlen(RadioTxBuffer.data);
15  RadioTxBuffer.appID   = 1;
16  RadioTxBuffer.flags   = SMARTNET_BF_BROADCAST;
17
18  while(1) {
19   SmartNET_SendPacket(&RadioTxBuffer);
20   waitEvent(&RadioTxBufferDone);
21   if (RadioTxBuffer.state != SMARTNET_BS_TX_OK) {
22     /* handle TX error */
23   }
24  }
25 }
26
27 SMARTOS_TASKENTRY(RadioRx) {
28  SMARTNET_INIT_BUFFER(RadioRxBuffer);
29  RadioRxBuffer.event    = &RadioRxBufferDone;
30  RadioRxBuffer.appID    = 1;
31
32  while(1) {
33   SmartNET_ReceivePacket(&RadioRxBuffer);
34   waitEvent(&RadioRxBufferDone);
35   MSG("RX␣from␣0x%04X␣at␣time␣[%llu]\n",
36           RadioRxBuffer.src, RadioRxBuffer.timestamp);
37  }
38 }
39
40 int __attribute__((noreturn)) main(void){
41  init_uart(UART_CFG_115200);
42  smartos_init_environment();
43  run_smartos();
44 }
```

Transmission and reception of radio messages in SMARTOS using SMARTNET

8 Applications

On the one hand, SNOW^5 is ideal for commercial applications concerning its small, customizable and energy efficient design along with its real-time capabilities and its various communication channels. On the other hand, modularity as well as easy and cheap debugging and prototyping enables SNOW^5 for research and educational issues.

Recommended fields of application are role-based scenarios where differently equipped nodes cover distinct areas of a comprising complex task. But also the activation of actuators like stepping motors is possible, due to the available D/A converters. One imaginable scenario could be the supervision of territories and buildings for security, informational and controlling aspects. This is feasible even in dangerous and misanthropical environments where no communication infrastructure is available and ad hoc networking is necessary. Moreover, SNOW^5 is easy to adapt to the requirements given, which is especially interesting for task forces like search and rescue services or firefighters employing the node in a large variety of situations.

For research and education purposes, SNOW^5 provides serial interfaces and JTAG for easy accessibility, which allows a quick deployment and analysis of several embedded software and middleware concepts. Distributed algorithms as well as several communication protocols can be explored. Functionality of miscellaneous sensors and actors can be studied as well with SNOW^5 by means of its stackable design and rapid prototyping.

9 Conclusion and future work

In this paper we have initially defined our demands on nodes of a WSN. According to these requirements we developed the SNOW^5 node, whose hardware (sections 3, 4) and software (section 7) was described in detail. We have compared it in depth to some other sensor nodes available (section 6) and outlined its advantages, especially its ultra low power consumption, modularity and extensive connectivity (section 5). Some examples of particularly suitable applications for SNOW^5 in commercial and research areas close this paper.

The successful establishment of a wireless sensor network using SNOW^5 and the real-time operating system SMARTOS finally enables us to evaluate theoretical assumptions under hard real-world conditions.

We are currently researching on theoretical problems like self-organizing and fault-tolerant systems. Underlying aspects will be network protocols, routing, time synchronization, power saving concepts, localization and embedded systems software design regarding efficient local and distributed algorithms. Future work will also lead to several daughter boards for sensors and actors in addition to those mentioned above. We will also look for hardware improvements and miniaturization of the SNOW^5 main board.

Our long-term objective is the specification of hardware and software requirements leading to a mass market system-on-a-chip (SoC) design for generic WSN nodes.

References

[1] BULUSU, NIRUPAMA: *Introduction to Wireless Sensor Networks*. In BULUSU, NIRUPAMA and SANJAY JHA (editors): *Wireless Sensor Networks - A Systems Perspective*, chapter 1, pages 1–19. Artech House, 2005.

[2] FENG, JESSICA, FARINAZ KOUSHANFAR and MIODRAG POTKONJAK: *Sensor Network Architecture*. In ILYAS, MOHAMMAD and IMAD MAHGOUB [30], chapter 12, pages 1–19.

[3] YARVIS, MARK and WEI YE: *Tiered Architectures in Sensor Networks*. In ILYAS, MOHAMMAD and IMAD MAHGOUB [30], chapter 13, pages 1–22.

[4] RÖMER, KAY and FRIEDEMANN MATTERN: *The Design Space of Wireless Sensor Networks*. IEEE Wireless Communications, 11(6):54–61, December 2004.

[5] AKYILDIZ, I. F., W. SU, Y. SANKARASUBRAMANIAM and E. CAYIRCI: *Wireless sensor networks: a survey*. Comput. Networks, 38(4):393–422, March 2002.

[6] KOLLA, REINER, MARCEL BAUNACH and CLEMENS MÜHLBERGER: *SNoW^5: A versatile ultra low power modular node for wireless ad hoc sensor networking*. In MARRÓN, PEDRO JOSÉ (editor): *5. GI/ITG KuVS Fachgespräch "Drahtlose Sensornetze"*, pages 55–59, Stuttgart, July 2006. Institut für Parallele und Verteilte Systeme.

[7] TEXAS INSTRUMENTS INC., Dallas (USA): *MSP430x1xx Family User's Guide*, 2006.

[8] TEXAS INSTRUMENTS INC., Dallas (USA): *MSP430x161x Mixed Signal Microcontroller (Rev. D)*, 2005.

[9] CHIPCON AS, Oslo (Norway): *CC1100 Data Sheet*, 2005.

[10] LORINCZ, KONRAD and MATT WELSH: *A Robust, Decentralized Approach to RF-Based Location Tracking*. Technical Report TR-04-04, Harvard University, 2004.

[11] BAHL, PARAMVIR and VENKATA N. PADMANABHAN: *RADAR: An in-building RF-based user location and tracking system*. In *INFOCOM (2)*, pages 775–784, 2000.

[12] ATMEL CORP., San Jose (USA): *AT45DB161B Data Sheet*, 2004.

[13] LUTTER, GERALD: *An AT54DB161B file system for YaOS*. Department of Computer Engineering, University of Wuerzburg, Würzburg, Germany, 2006.

[14] SIPEX CORP., Milpitas (USA): *SP3222E/3232E Data Sheet*, 2005.

[15] NATIONAL SEMICONDUCTOR: *LM1117 800mA Low-Dropout Linear Regulator*, 2005.

[16] KOLLA, REINER, MARCEL BAUNACH, and CLEMENS MÜHLBERGER: *SNOW BAT: A high precise WSN based location system*. submitted at EWSN'07, 2006.

[17] CROSSBOW TECHNOLOGY INC., San Jose (USA): *MICA2 Wireless Measurement System*, 2005.

[18] BEUTEL, J., O. KASTEN, F. MATTERN, K. RÖMER, F. SIEGEMUND, and L. THIELE: *Prototyping wireless sensor network applications with BTnodes*. In *Proc. 1st European Workshop on Sensor Networks (EWSN 2004)*, volume 2920 of *Lecture Notes in Computer Science*, pages 323–338. Springer, Berlin, January 2004.

[19] FU BERLIN: *ScatterWeb - ESB*, 2006.

[20] HOESEL, L.F.W. VAN, S.O. DULMAN, P.J.M. HAVINGA, and H.J. KIP: *Design of a low-power testbed for wireless sensor networks and verification*, 2003.

[21] POLASTRE, JOSEPH, ROBERT SZEWCZYK, and DAVID CULLER: *Telos: Enabling ultra-low power wireless research*. In *Proceedings of the Fourth International Conference on Information Processing in Sensor Networks: Special track on Platform Tools and Design Methods for Network Embedded Sensors (IPSN/SPOTS)*, 25.-27. April 2005.

[22] ATMEL CORP., San Jose (USA): *ATMega128(L) Data Sheet*, November 2004.

[23] RF MONOLITHICS INC., Dallas (USA): *TR1001 Data Sheet*, 2005.

[24] CHIPCON AS, Oslo (Norway): *CC1000 Data Sheet*, 2005.

[25] KOLLA, REINER: *YaOS – Yet another Operating System*, 2005. presentation.

[26] BARRY, RICHARD: *FreeRTOS™ homepage*. http://www.freertos.org/, 12. December 2005.
[27] UC BERKELEY: *TinyOS*. http://www.tinyos.net/, 2004.
[28] SKYDAN, OLEG: *SOS - small operating system*. http://skydan.in.ua/SOS/, 25. February 2006.
[29] YANBING, LI, MIODRAG POTKONJAK, and WAYNE WOLF: *Real-time operating systems for embedded computing*. In *ICCD*, pages 388–392, 1997.
[30] ILYAS, MOHAMMAD and IMAD MAHGOUB (editors): *Handbook of Sensor Networks: Compact Wireless and Wired Sensing Systems*. CRC Press, 2005.