Smart Dust Mote Core Architecture

Brett Warneke, Sunil Bhave
CS252, Spring 2000: Project Report
Berkeley Sensor and Actuator Center, 497 Cory Hall, Berkeley CA 94720
{warneke, sunil}@eecs.berkeley.edu

Abstract

The Smart Dust project is exploring the limits of autonomous sensing and communication by packing an entire system into a cubic millimeter at a relatively low cost. These volumetric constraints correspond to energy constraints on the system. Therefore, the mote "intelligence" must operate on the absolute minimum energy while providing necessary features. The mote can be partitioned into four subsystems: sensors and analog signal conditioning, power system, transceiver front end, and the core. The core is essentially all the digital circuits in the system, including the receiver back end, sensor processing circuits, computation circuits, and memory. One requirement of the core is that it have a degree of on-the-fly reconfigurability determined by the changing needs of the mission. In this project we define an ultra-low energy architecture for the mote core that will meet the needs of the military base monitoring scenario.

Introduction

The goal of the Smart Dust project is to build cubic-millimeter scale sensing and communication platforms (figure 1) that form a distributed sensor network[1,2] and can monitor environmental conditions in both military and commercial applications. These networks will consist of hundreds to thousands of "dust motes" and a few interrogating transceivers. The dust motes are comprised of various subsystems from different fabrication technologies. Many sensors, including temperature, pressure, and acceleration sensors, from MEMS and CMOS processes can be attached to a mote. An ASIC handles measurement recording, data storage, and system control. A receiver circuit converts photocurrent from an incoming laser into a data stream to be used to interrogate or reconfigure the mote. Several transmission systems can also be utilized, such as a passive corner cube reflector (CCR)[3] for communication to a base station, or an integrated laser with beam steering MEMS structures[4] for inter-mote communication. Finally, all of the components are mounted onto a thick-film battery charged by a solar cell.

The most difficult constraints in the Smart Dust design are those regarding the minimum energy consumption necessary to drive the circuits and MEMS devices. When fitting the entire mote within a $1mm^3$ volume, the energy density of the power supply is the primary issue. Current technology yields batteries with ~$1J/mm^3$ of energy and a high series resistance. Modern capacitors can achieve as much as ~$10mJ/mm^3$ with a low series resistance. Series

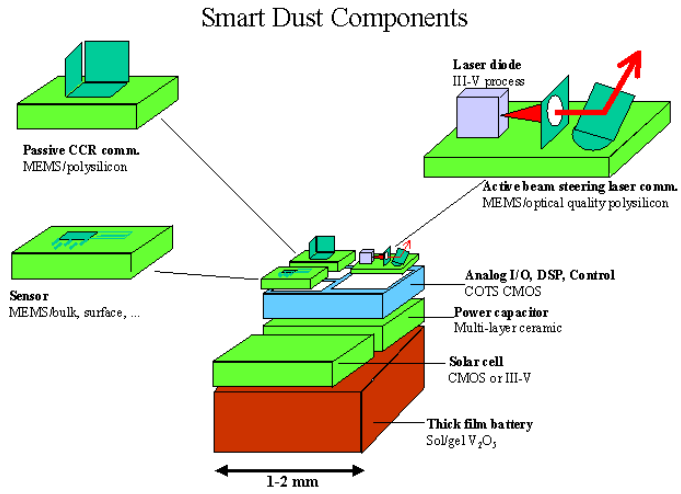

Figure 1: *Conceptual Diagram of the Smart Dust Mote for $1mm^3$ autonomous distributed sensing and communication*

resistance affects the peak power that can be pulled from the source.

In typical low power mixed-signal systems, most designers consider performance in terms of cycles, samples, or bits, maximizing performance first and minimizing power second. With the strict power constraints for Smart Dust, we are forced to consider performance in terms of Joules: given a cubic-millimeter battery, there is one Joule of energy to use. With the CCR, communication costs about 1nJ/bit, while sensing can be achieved at ~1nJ/sample. Modern processors, such as the StrongARM SA1100[5], can perform computations as low as ~1nJ/instruction. With these energy figures, one can make cost tradeoffs between the amount of computation, the amount of data transmitted and the sensor sampling frequency. However, by using a closer mapping of the application needs to the architecture and targeting ultra-low energy from the start, we believe we can achieve orders of magnitude reduction in the energy cost per instruction.

Keeping this in mind, the goals of the project were:

- Determine the exact functional needs for a particular scenario including necessary signal processing and computation functions. In addition, the exact amount of reconfigurability needed was to be determined.
- Map the necessary functionality of the core into various possible architectures.
- Evaluate the choices using Wattwatcher (Verilog power estimator) and/or Powermill (switch-level power estimator) to determine the lowest energy solution.

Core Functionality Specification

In order to allow us to make realistic tradeoffs, a particular application scenario was chosen to guide the design. We chose the case of military base monitoring wherein on the order of a thousand Smart Dust motes are deployed outside a base by a micro air vehicle to monitor vehicle movement. The motes can be used to determine when vehicles were moving, what type of vehicle it was, and possibly how fast it was travelling. The motes may contain sensors for vibration, sound, light, IR, temperature, and magnetization. CCRs will be used for transmission, so communication will only be between a base station and the motes, not between motes. A typical operation for this scenario would be to acquire data, store it for a day or two, then upload the data after being interrogated with a laser. However, to really see what functionality the architecture needed to provide and how much reconfigurability would be necessary, an exhaustive list of the potential activities in this scenario was made.

The operations that the mote must perform can be broken down into two categories: those that provoke an immediate action and those that reconfigure the mote to affect future behavior.

- Immediate operations
  - Transmit ID – provides a mote health report
  - Transmit current reading from sensor *x*
  - Transmit current readings from all sensors
  - Send logged data for sensor *x*
  - Are you logging data?
  - Do you have data logged?
  - Store the following value as the real time clock counter reference time
  - Turn on/off the transmission training sequence – the base station is going to move or has stopped moving
- Reconfiguration operations
  - Start logging data from sensor *x* with samples every *t* seconds
  - Stop logging data
  - Set the receiver wakeup interval to *t* seconds – may be from a second to a day
  - Set logging threshold
  - Set filter coefficients
  - Set FFT bin width and positions
  - Apply a FIR filter to the data stream from sensor *x*
  - Broadcast logged data every *t* seconds (Scattercast mode) – don't waste energy turning on the receiver
  - Take an FFT on sensor *x* and immediately transmit if the spectral content in band *y* is above *z*
  - Take an FFT on sensor *x* and store any bands that exceed the threshold
  - If sensor *x* exceeds *z*, then turn on sensor *y* and transmit/log its information

From this list of operations, a list of basic activities that the various logical portions would need to do (i.e. receiver – timing recovery, data recovery, decode packet). The list of activities was then broken down into a list of specific functions that the architecture would need to support, such as selecting a sensor, move the time stamp to memory, calculate the CRC, compare a value to a threshold, etc. The comprehensive list can be found at http://www-bsac.eecs.berkeley.edu/~warneke/cs252/Military_Base_Scenario.html.

Proposed Architecture

Trimming the application space of a general purpose microprocessor can achieve only so much in terms of energy savings. Instead we propose to implement an ultra-low power ASIC design with on-the-fly reconfigurability of the computational blocks.

Looking through the functional specifications for the core, we realized that each operation is regulated by a timed event; hence a bank of timers forms the basis of the architecture. For minimum energy, a direct mapping of a particular function into hardware is generally best, but from the list of specifications it was clear that a certain amount of reconfigurability would be necessary. Thus, the timers enable setup memories that configure functional blocks into data paths that provide only the capabilities necessary for that event. These paths are data-driven so that functional blocks are only powered up when their inputs are ready, minimizing standby power and glitching. A block diagram of this new architecture is shown in figure 2.

Figure 3 details a section of the timer bank and setup memory. The timer is loaded from the timer value memory, setting its period. When the timer expires, it enables *setup memory 1*, which configures the data path to perform the desired function. When the data path has finished its operation, *setup memory 1* will release its configuration and

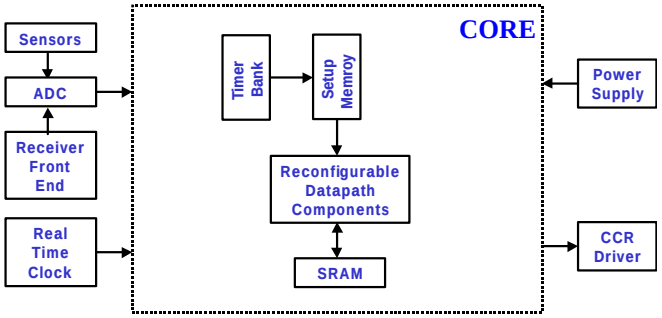

Figure 2: *Top-level diagram of the mote and the proposed core architecture architecture*

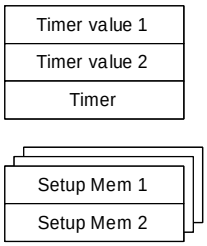

Figure 3: *Timer and associated setup memories: The timer allows multiple timer periods. The different setup memories allow multiple data path configurations per event. Energy-driven operation isfacilitatedby multiple banks of setup memory facilitated*

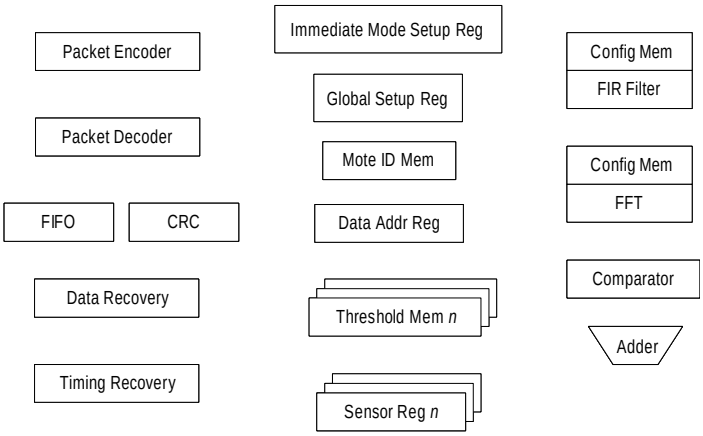

Figure 4: *Functional blocks whose connectivity is configured by the setup memory when a timer expires. The global setup memory is always enabled.*

either the timer value can be loaded into the timer and the countdown restarted or *setup memory 2* can be enabled. *Setup memory 2* will then configure the data path for another operation, thus facilitating multiple operations per timer event. Additional setup memory can be added for more involved sequences.

Multiple timer periods are desirable for several situations. For example, one might want to sample a sensor at a slow rate until an interesting signal is detected. At that point, the sampling rate should increase. In addition, the motes might be deployed without anyone coming back to talk to them for a day, so it would be desirable to be able to set the receiver wake-up timer to not wake-up for 24 hours, but then it should decrease the period dramatically to 10's of seconds in case one doesn't make it back to talk to the mote at exactly the right time. The proposed architecture facilitates this by providing multiple timer values that can be loaded into the timer depending on the results of the data path computation.

Another feature of this architecture is energy-driven operation modes. An energy-monitoring unit selects between multiple banks of setup memory and timer values depending on the current level of the energy stores. Each bank can have different timer periods and algorithms to control energy expenditure.

Two types of packets can be sent to the mote, corresponding to the two types of operations. Immediate mode operations use the packet body to configure the data path right away. Reconfiguration operations load the packet body into the setup memory for future configuration.

Figure 4 shows the functional blocks included in the reconfigurable data path. For the communications back end, there is a data recovery block, timing recovery block, FIR filter, packet encoder that does bits stuffing and adds the flag byte, packet decoder that does bit unstuffing, CRC block, and a FIFO. Incoming packets are stored in the FIFO until the CRC can be verified, at which point the packet body will be used as described above. The global setup memory holds certain timer-independent configuration bits, such as timer enables. The sensor registers are used to store previous sensor readings to use in computing data changes. Various computation blocks can be included in the data path, such as an adder, comparator, and FFT unit.

All of the functional units in the data path are data driven. The setup memory only powers up and enables the first set of units that are needed, such as the sensor and ADC. Once these units have done their job, they assert a *done* signal that is routed, based on the configuration memory, to the next unit, such as the adder, and powers it up and enables it. Likewise, when this unit has finished its job, it will power up and enable the next device in the chain. The last unit in the path will cause the timer to reload its value and cause the setup memory to stop configuring the data path. The advantages of this data driven technique include minimizing the standby power by keeping components powered down until exactly when they are needed, and ensuring that the inputs are stable before the next device is powered up, which minimizes glitches.

It is significant to note that since this architecture does not use shared busses as in traditional microcontrollers, the functional components can be configured for certain parallel operations. For example, a sensor reading could be both stored in SRAM and transmitted with the CCR, although this is not necessarily a desirable capability.

A large number of wires will be necessary to implement this architecture in order to allow configurable connectivity between so many units and to distribute all the control signals. Two potential choices for implementing the wires include point-to-point routing of control and data wires between each block and a mesh of wires with routing switches, similar to an FPGA. We focused on the former approach in this work since at this point the design seemed small enough that the point-to-point wiring would not be too onerous and the mesh would not show significant advantages.

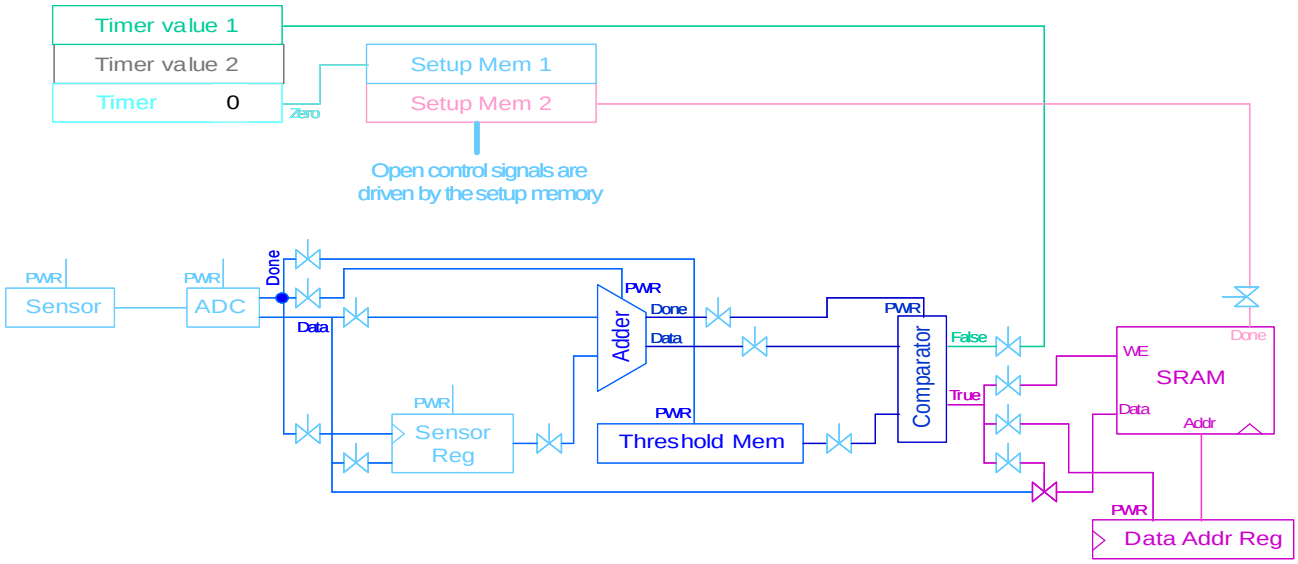

Figure 5: *Example configuration to sample data, find the change in value, and store it in the SRAM is the change exceeds a certain threshold. The colors indicate the order in which the components and signals become active, starting with the timer, proceeding to the SRAM and back to the timer.*

Figure 5 delineates the operation of the architecture by show the configuration for one of the most common tasks, acquiring sensor data, checking if it has changed more than a threshold value, then storing the result to memory. The colors indicate the order that components and signals become active, clearly showing the data driven nature.

One potential hazard of this architecture is that the *done* signals can glitch as the blocks are powered up, which would provide a false trigger to the next stage. A second issue is that despite the fact that the blocks are powered down, the internal nodes do not discharge immediately. For example, an 8-bit comparator whose 1V Vdd line is supplied by a PFET, will only discharge 13mV in 100µs with the PFET turned off. If the inputs change while powered off in this manner, a new calculation can be performed and only drop Vdd by 170mV. An advantage of this is that less charge will be needed when the block is powered up again. However, this stored charge will also allow the block to continue to drive its outputs despite being powered down, so the outputs will generally need tri-state buffers. These hazards will require some extra work at the circuit level to make this architecture work.

Hspice simulations were used to determine the power and energy consumption of some of the blocks to test the feasibility of the proposed architecture. We used a standard cell library for the National Semiconductor 0.25µm process as the basis of the design. A 1V supply was used with a V_T of 0.55V. Initially we simulated the timer since it runs continuously and thus is a significant portion of the power consumption. A 12-bit, loadable countdown timer running at 10kHz consumes 5.4nW, or 540fJ per cycle. The same simulation in PowerMill gave 5.2nW, demonstrating comparable results to Hspice while running more than 100 times faster. Next, we simulated the 8-bit comparator with a power-up/down PFET. We adjusted the rise and fall time of the power control signal between 1ns and 10µs and the W/L of the PFET from 1/0.24 to 100/0.24. The circuit was power cycled from the initial operating point to charge up the internal nodes, the comparator inputs were changed to effect a new comparison, and a second power cycle was run. The energy was computed for the second power cycle, including 1µs of on-time. The energy consumption only varied by about 10% and was approximately 95fJ. The average power consumption during the 1µs of on-time was measured to be about 2.9nW with about 30% variation. The average power consumption for the off-time with the internal nodes charged, was about 6.4pW with 3% variation. From these numbers we can determine when it is worthwhile to perform power cycling. By dividing the energy consumed in the power cycling by the static power consumed when the module is powered up, we find the maximum amount of time that the circuit can be on before it is beneficial to turn it off. In this case, we find that the comparator should be turned off if it is idle more than 33µs, or running slower than 31kHz. Since the maximum speed that anything in the core would run at is 10kHz, and most operations would occur at speeds on the order of 100Hz down to 0.01Hz (100sec), this power cycling scheme is very advantageous.

We are currently in the process of estimating the energy consumption for the configuration shown in figure 5 using PowerMill. A corresponding subroutine on an ARM8 would consume 1.44nJ at 1V and 10kHz[6]. The preliminary results from the Hspice simulations above indicate that we should be able to achieve at least two orders of magnitude lower energy consumption with the proposed architecture. In addition, a microcontroller being designed for ultra-low energy operation and targeted at the same scenario as the proposed architecture will provide a more realistic comparison between a conventional microprocessor architecture and the proposed architecture.

Conclusion

Since Smart Dust has very limited energy resources, our goal for this project was to minimize energy consumption through architecture, while providing the necessary functionality and reconfigurability for a particular scenario. The proposed architecture uses a reconfigurable data-driven data path, which facilitates power cycling of the functional units, thereby reducing the standby energy consumption to a minimum. HSPICE simulations were performed on two key elements of the architecture, namely the timer and the comparator, to determine if power cycling was advantageous.

References

1. J. M. Kahn, R. H. Katz, and K. S. J. Pister, "Mobile Networking for Smart Dust", *ACM/IEEE Intl. Conf. on Mobile Computing and Networking*, Seattle, WA, August 17-19, 1999.
2. B. Atwood, B. Warneke, K.S.J. Pister, "Preliminary Circuits for Smart Dust," *Proceedings of the 2000 Southwest Symposium on Mixed-Signal Design*, San Diego, California, February 27-29, 2000.
3. P. B. Chu, N. R. Lo, E. Berg, and K. S. J. Pister, "Optical Communication Using Micro Corner Cube Reflectors", *Tenth IEEE International Micro Electro Mechanical Systems Conference (MEMS 97)*, Nagoya, Japan, Jan. 26-30, 1997, pp. 350-5.
4. Last, M., Pister, KSJ, "2-DOF Actuated Micromirror Designed for Large DC Deflection", *MOEMS '99*, Mainz, Germany, Aug29-Sept 1.
5. StrongARM SA1100 microprocessor datasheet.
6. Peggy Laramie, "Instruction Level Power Analysis and Low Power Design Methodology of a Core Processor", M.S. Thesis , UC Berkeley, 1998.