MICROSYSTEM CONTROLLER FOR SENSOR NETWORK CONTROL AND DATA CORRECTION

Prasanna Balasundaram, Kartik Vaidyanathan, and Andrew Mason
Michigan State University, East Lansing, MI 48824.

ABSTRACT

A microcontroller specifically tailored to the processing needs and energy budget of battery-powered sensor-based microsystems is described. The microsystem controller contains both a microprocessor core and a data computation engine to support the operation control and data processing needs common to sensor systems. The RISC microprocessor is capable of general-purpose control and data handling but also includes hardware blocks and instructions to facilitate sensor network communications. The configurable computation engine provides energy-efficient hardware for digital calibration and compensation of sensor data and supports general signal processing algorithms. The methodology, features and functionality, of the microsystem controller, are presented.

1. INTRODUCTION

Semiconductor sensors play an important role in measuring non-electrical parameters such as temperature and pressure. Advances in Micro-Electro-Mechanical-Systems (MEMS) technologies enable sensors only micrometers in size to be employed in range of military, medical, and industrial applications. These advances, along with modern microelectronics and packaging technologies, facilitate the development of integrated microsystems, which combine sensors with readout, signal processing, and communication circuitry to form highly capable, intelligent, interfaces to the physical world.

The system architecture for a highly integrated multi-parameter microsystem is shown in Figure 1. Here, various sensors and actuators are connected to a microsystem controller through interface circuits. The sensor interface circuits provide analog signal conditioning and support communication of sensor data over an intra-module sensor network [1]. The microcontroller plays an important role in the microsystem, controlling system operation and managing communications between the sensor network and the user or host system. It also plays a major role in front-end signal processing, performing calibration and compensation of the sensory signals and computationally intensive operations such as filtering and data compression.

This paper presents a microcontroller designed specifically for the features and performance demands of sensor-based

microsystem. Because the controller power consumption is typically a limiting factor in battery powered microsystems [2], the reported microsystem controller maps several key functions into energy optimized hardware to improve microsystem operation lifetime. The controller contains a microprocessor core and a data computation engine which share a memory block to hand data back and forth. These two primary blocks are described.

Figure 1: Components of a sensor microsystem.

2. MICROPROCESSOR CORE

The processor was designed to work in conjunction with a network of sensors whose requirements have been used to define the features of the processor. These sensors measure analog data from the environment that are then quantified into measurements of pressure, temperature, humidity etc. The role of the processor is to monitor these sensor nodes and arbitrate the processing of the data received from these nodes. To minimize the complexity (and therefore size and power) of the controller, a Reduced Instruction Set Computer (RISC) load store processor was chosen over a Complex Instruction Set Computer (CISC). Low power design requirements of the microsystem application called for implementation of architectural and circuit-level techniques to reduce the processor power consumption [3]. A power saving standby mode was incorporated to cater to the sporadic operation common in sensor-based systems. Other features such as gated clocks and interrupts were

used to further reduce the power consumption of the processor. Harvard architecture, with independent program and data memory, was chosen over the Von-Neumann architecture. This allowed simultaneous memory access from both the instruction and data memory thereby increasing the memory bandwidth. It also simplified memory access by keeping the instruction and data memory separate. This improved the speed of the processor and removed the need for dual ported memory.

2.1. Architecture

The processor architecture is shown in Figure 2. The CPU has a 16-bit arithmetic logic unit and a register file that consists of 8 registers for data. The CPU has additional registers such as the program status register and condition code register that are used to set flags. To control the flow of execution, the CPU uses a 10-bit program counter and 10-bit memory address register. The CPU is pipelined into three stages to improve the throughput of the processor and control the sequence of operations in the processor. The instruction memory is synthesized to a 2Kbyte (1024 x 16bit) static RAM and the data memory a 512 byte (256 x 16bit) RAM block. In addition to the instruction and data memory, a separate 1Kbyte (256 x 32bit) sensor data memory has been designed to store sensor data information and interface with the data correction unit.

Figure 2: Functional components of the microsystem controller microprocessor core.

The I/O ports are divided into general purpose I/O, used to interface with blocks such as Direct Memory Access (DMA) controller, and a specific sensor bus I/O port, designed to implement the Intra Module Multielement (IM²) bus. The IM² bus is based on the IEEE 1451.2 standard modified to support networks of sensor nodes within a microsystem module [1]. The sensor bus port maps to hardware what would otherwise be a complex set of instructions needed to facilitate communication between the processor and the sensor nodes.

The sleep unit in the processor is used to shut down the processor and manage a sleep mode to save power. During normal operation, the processor will periodically initiate a

request for data to sensor node and then processes data received from the sensor. In the sleep mode the processor clock is turned off and the processor monitors the sensor ports for an interrupt to turn the processor back on. After receiving an interrupt the processor turns back on, shifting to normal operation mode, and starts executing instructions.

2.2. General Purpose & Sensor Bus I/O Ports

The general-purpose I/O port consists of a 16-bit data line and a 10-bit address line. The processor supports direct memory access (DMA) operations and has signals such as dma_req, dma_ack and dma_rw on the I/O port. These ports are used to perform DMA operations and can read and write data to the memory block. The DMA operation is also flagged in the program status register to indicate that a DMA operation is in progress.

A special port was implemented in the processor to arbitrate communication between the controller and the front-end sensor nodes using the IM² sensor bus protocol [1]. The signals, functions, and driver for IM² bus are listed in Table 1. The required functions of all communication signals (i.e., not power signals) are implemented in the sensor bus port of the processor. In addition to managing serial data transfer and handshaking signals, the sensor bus port monitors the NINT line for interrupts and implements software controlled asynchronous triggering on NTRIG.

Table 1: IM² sensor bus signals [1].

Signal	Description	Driver
DIN	Serial data line from the microcontroller to the interface module.	Processor
DOUT	Data transmitted from the interface module to the microcontroller.	Sensor Node
DCLK	Positive-going edge latches data on DIN and DOUT.	Processor
NIOE	Signals that data transport is active and delimits data framing.	Processor
NTRIG	Performs triggering function.	Processor
NACK	Trigger/data transport acknowledge.	Sensor Node
NINT	Used by the interface module to request service from microcontroller.	Sensor Node
NSDET	Used by microcontroller to detect the presence of new interface module.	Sensor Node
Ground	Signal/power reference.	
Power	Normal 3-V power supply.	
Power2	Controllable power supply. Low when system is in sleep mode.	

In addition to typical Arithmetic, Load/Store, Logical, Control/Branch, and Rotate/Shift instructions, the processor implements several special instructions defined in Table 2. The Snd and Rcx instructions take two clock cycles to complete, so there is a stall in the pipeline that will float through after any of these instructions are executed. The Str

instruction has to be executed immediately after the Rcx to store the received data in the sensor data memory. The Pull instruction can select between NIOE and NTRIG with bits [10:8] of the instruction and chooses between pulling high or low by setting the bit [0] of the instruction.

Table 2: Sensor Bus Instructions

Instruction	Description
Snd #data	Sends the data over the DIN; 1 byte of data is sent over 8 clock cycles on the DCLK line.
Rcx #data	Receives data over the DOUT; 1 byte of data is received over 8 clock cycles on the DCLK line.
Str #addr	Received data is stored in #address of the sensor data memory.
Pull #sig	Pull the NIOE/NTRIG lines high/low.

3. DATA COMPUTATION ENGINE

The other primary block of the controller is designed to process sensor data. It can be configured for general computation but includes special hardware to perform sensor data correction in an energy efficient manner.

3.1. Data Correction Unit

Sensor signals contain various types of error inherent in the sensor manufacturing process. Correcting offset (non-zero output for a zero input condition) and nonlinearity (nonlinear output to a linear input) is known as calibration, while correcting cross-sensitivity (output depends on parameters other than it measures) is known as compensation. These error correction processes can consume up to 40% of the energy spent by a microsystem, and correcting these errors can add up to 50% to the cost of the microsystem. Error correction is typically performed using an off-the-shelf microcontroller [4, 5]. However, the significant impact calibration and compensation can have on the cost and power budget of a microsystem warrant effective and efficient error correction techniques be built in to the microsystem controller.

The data correction unit, shown in Figure 3, performs calibration and compensation operation prescribed by the IEEE P1451.2 standard by evaluating a single multinomial [5]. The entire operating region of the sensor is divided into many segments defined by the calibration Transducer Electronic Data Sheet (TEDS), and in each segment, the signal transfer curve is represented by a multinomial given by:

$$Y = \sum_{i=0}^{D(1)} \sum_{j=0}^{D(2)} \dots \sum_{p=0}^{D(n)} C_{i,j,\dots,p} [X_1 - H_1]^i [X_2 - H_2]^j \dots [X_n - H_n]^p$$

where, X_k variables are the independent signals that determine the corrected output (Y), H_k are the offsets associated with each input signal, $C_{i,j,p}$ are the floating-point coefficients, and $D(k)$ are the maximum order a particular

input signal can be raised. The multinomial can have different coefficients and offsets for each segment. The data correction engine performs this error correction operation using dedicated floating-point arithmetic hardware in an energy efficient manner.

Figure 3: Block diagram of the data correction unit.

3.1.1. Floating Point Unit

The floating-point arithmetic units are deeply pipelined to perform error correction in a short period of time. The input signals are pre-normalized before the integer multiplication and addition stages. The result produced by both the multiplier and the adder is post-normalized to maintain the accuracy of the floating-point computations. Before performing the actual floating-point multiplication and summation, the significance of each sum term in the multinomial is predicted by calculating the tentative exponent of the sum term in the multinomial. To avoid loss of accuracy in the floating-point summation, the summation is performed starting from the term of least significance. Alternatively, if the summation is started from the term of largest significance, a coarse result can be obtained with fewer computations. This method can save power by halting the computation when successive terms will not add significant value to the result. The computation engine can be configured to operate in either of these modes, allowing the user to make the accuracy versus power tradeoff.

3.1.2. Hardware Sorter

A hardware sorter capable of efficiently sorting sixteen 8-bit wide unsigned integers has been designed to sort the tentative exponents of the sum terms (Figure 4). When a new integer arrives to the sorter, it is compared with all the existing integers. The comparator produces a '1' if the existing integer is smaller than the new integer. Hence the number of 1's in the comparator results represents the rank of the new integer. Once the rank of the new integer is determined, the rank of all the integers whose rank is greater than or equal to the rank of the new integer is incremented. Thus, at any point of time, the ranks of the integers are unique. The new integer is stored in the next vacant register along with its rank.

The parallelism of the required operations does not require more than one clock cycle to find the rank of a new integer and to update the rank of all other integers. Hence it efficiently map in to a pipelined hardware implementation for the data correction unit. Moreover, the scheme does not require any time (and energy) consuming register-swapping operations as in software sorting schemes. The integers are stored in content addressable memory (CAM) locations with the rank as the 'key' field. If an integer of a particular rank is needed, the rank is given as input for the CAM and the integer is obtained by matching the input against the rank of all existing integers. Using this scheme, the integers can be recalled from the sorter in either ascending or descending order by giving the output of an up or down counter as the rank input.

Figure 4: Functional diagram of the hardware sorter.

3.2. Re-configurable Computation

The data correction unit is capable of adapting itself to a variety of operations in addition the error correction operation. It can be programmed to work as a dedicated integer/floating point multiply and accumulate unit shutting down the hardware sorter unit. It can adapt to a new rounding scheme where the rounding bit is forced to zero bypassing the entire post normalization blocks and complete with computations in fewer clock cycles. The microcontroller can take control of all hardware resources to operate as a general-purpose floating-point co-processor to perform filtering operations and data fusion algorithms.

4. SUMMARY OF ENERGY EFFICIENCY

The microsystem controller was implemented in a top-down design process using a custom library of energy-optimized cells. For example, since flip flops are used extensively in the pipelined microprocessor and data correction unit, the energy demands of five different flip-flop structures (in-house flip-flop with and without reset, push-pull isolation flip-flop [6], transmission gate flip-flop and a regular master-slave flip-flop) were thoroughly analyzed (Figure 5) and the two most efficient structures were included in the cell library. Significant energy

savings are provided by the configurable sorting options, rounding schemes, and perturbation-only calculations within the data correction unit. Clock gating schemes and sleep mode operations, which shut down the unused blocks, further contribute to the low energy objective.

Figure 5: Comparison of energy consumption in flip-flops.

5. CONCLUSION

A microcontroller developed for sensor network control and data computation applications has been described. The controller implements several microsystem-specific instructions and utilizes a configurable computation engine to process sensor data. A novel data correction engine, which implements several techniques to maximize energy efficiency, was presented. The controller simplifies microsystem implementation while facilitating the use of sensors that require complex calibration and compensation.

6. REFERENCES

- [1] J. Zhang, J. Zhou, P. Balasundaram, and A. Mason, "A Highly Programmable Sensor Network Interface with Multiple Sensor Readout Circuits," *Proc. of IEEE Sensors*, October 2003.
- [2] A. Mason, N. Yazdi, A. V. Chavan, K. Najafi, and K. D. Wise, "A Generic Multielement Microsystem for Portable Wireless Applications," *Proc. IEEE*, vol. 86 (8), pp. 1733-1745, August 1998.
- [3] K.M. Lim, S.W. Jeong, Y.C. Kim, S.J. Jeong, H.K. Kim, Y.H. Kim, B.Y. Chung, H.L. Roh, and H.S. Yang "CalmRISCTM: a low power microcontroller with efficient coprocessor interface," *IEEE Int. Conference on Computer Design*, pp. 299-302, 1999.
- [4] K.F. Lyahou, G. van der Horn, and J.H. Huijsing "A noniterative polynomial 2-D calibration method implemented in a microcontroller," *IEEE Trans. on Instru. & Meas.*, vol. 46 (4), pp. 752-757, August 1997.
- [5] M.Mozek, D.Vrtacnik, D.Resnik, U.Aljancic, M.cvar, and S.Amon, "Calibration and Error Correction Algorithms for Smart Pressure Sensors," *IEEE MELECON*, May 2002.
- [6] D. Markovic, B. Nikolic, and R.W. Brodersen, "Analysis and design of low-energy flip-flops," *IEEE ISLPED*, August 2001.