

Sunflower: Full-System, Embedded Microarchitecture Evaluation

Phillip Stanley-Marbell and Diana Marculescu

Department of Electrical and Computer Engineering
Carnegie Mellon University
Pittsburgh, PA 15217, USA

Abstract. This paper describes *Sunflower*, a full-system microarchitectural evaluation environment for embedded computing systems. The environment enables detailed microarchitectural simulation of multiple instances of complete embedded systems, their peripherals, and medium access control / physical layer communication between systems. The environment models the microarchitecture, computation and communication upset events under a variety of stochastic distributions, compute and communication power consumption, electrochemical battery systems, and power regulation circuitry, as well as analog signals external to the processing elements.

The simulation environment provides facilities for speeding up simulation performance, which tradeoff accuracy of simulated properties for simulation speed. Through the detailed simulation of benchmarks in which the effect of simulation speedup on correctness can be accurately quantified, it is demonstrated that traditional techniques proposed for simulation speedup can introduce significant error when simulating a combination of computation and analog physical phenomena external to a processor.

1 Introduction

The focus of computing systems has shifted over the years, from mainframes in the 1960's and minicomputers in the '70s, to workstations and personal computers (PCs) in the '80s. A majority of recent microarchitecture research has focused on architectures destined for PC, workstation and server-class systems. The last decade has however seen increasing use of computing systems that are *embedded* unobtrusively in our environments. Examples at the low end of the spectrum include low-power wired and wireless sensor networks, to home automation systems, and so-called "white goods" (e.g., microwaves, dishwashers). At the middle- to high-end, personal digital assistants (PDAs) and mobile telephones employ embedded processors for application execution and signal processing. Processors in embedded systems typically interact directly with analog signals in their environment, through analog-to-digital (A/D) and digital-to-analog (D/A) converters. For example, a processor in a home automation system might be connected through an on-chip or external A/D converter, to temperature and humidity sensors. These systems not only have power consumption constraints, but are also often battery powered, necessitating the consideration

of how the variation of power consumed over time affects the efficiency of energy sources such as electrochemical cells.

In order to accurately evaluate microarchitectures for embedded applications, it is desirable to model the *entire system* — from the detailed microarchitecture, to the typical integrated peripherals as well as the time-varying analog signals in the environment which often drive computation. It is desirable to be able to perform power and thermal estimation in these systems, and also to have a means of determining how the power consumption profile (variation in power dissipated over time) affects battery lifetime; this arises from the fact that two power consumption profiles with the same average, can result in different effective battery lifetimes, due to non-linearities in electrochemical cell characteristics. Since integrated circuits are usually connected to batteries through voltage regulators, which have variable efficiencies across different current loads, an ideal simulation environment would also provide capabilities for modeling the behavior of such regulators. Perhaps even more so than in PC-, workstation- and server-class systems, embedded computing systems are subject to failures. If investigating microarchitectural techniques for building reliable systems, it is desirable to be able to model intermittent processor and network failures, drawn from a variety of stochastic failure models.

This paper describes *Sunflower*, a full-system microarchitectural evaluation environment for embedded computing systems. The environment enables simulation of multiple instances of complete embedded systems, their peripherals (e.g., A/D converters, timers), as well as medium access control- (MAC) and physical-layer (PHY) communication between systems. For each system, the environment enables detailed microarchitectural simulation of instruction execution (pipelined instruction execution, caches, memory management hardware, on-chip buses), modeling of computation and communication faults, compute and communication power consumption, electrochemical battery systems and power regulation circuitry. The simulation environment enables the modeling of analog signals external to the processing elements, permitting realistic evaluation of a combination of computation, communication and their interaction with analog signals. The facilities provided by the environment include:

- Full-system simulation of multiple, networked embedded systems.
- Microarchitectural simulation of a choice of a 16-bit or 32-bit architecture.
- Integrated trace gathering capabilities harnessing debugging information in compiled binaries.
- Switching activity estimation and instruction-level power models.
- Voltage and frequency scaling across different values of process technology parameters (V_T , K , α).
- Electrochemical battery and voltage regulator models for several batteries and voltage regulators.
- Modeling propagation and spatial attenuation of analog signals.
- Medium Access Control (MAC) and physical-layer modeling of arbitrary communication topologies.
- Modeling of logic upsets in the microarchitecture, and whole-system failures in processor cores and network interfaces.

- Failure modeling supported by random variables generated from over 20 common probability distributions.

2 Related Research

Simulation forms the backbone of exploratory microarchitecture research. Through the use of appropriate tools, architects can gain insight into how design decisions affect metrics of interest, be they performance (throughput, latency), power consumption, thermal efficiency, reliability, or other domain-specific properties.

Tools that have been developed for use in microarchitecture simulation include the Wisconsin Wind tunnel simulators [1], SimpleScalar [2] and its derivatives, Trimaran’s HPL-PD simulator [3], Microlib [4] and Liberty [5]. Each of these simulation environments have addressed specific needs of their developers, or have addressed trends in architecture research, such as multiprocessor systems and their coherence mechanisms, wide-issue, deeply pipelined superscalar out-of-order machines, EPIC/VLIW and embedded architectures, or modular simulator construction. It is sometimes the objective not to design a new microarchitecture, but rather, to design a *system architecture*, including the processor, complete memory hierarchy (caches, main memory, backing store), networking and other I/O. In such situations, architects typically use *full-system simulators*. These include tools like SimOS [6], M5 [7] and Simics [8].

Over the last decade, power consumption and thermal efficiency have emerged as leading design evaluation metrics, and this has led to the development of new tools and extensions of existing microarchitectural simulators, to enable power and thermal modeling. Examples include the HotSpot [9] thermal analysis framework, and the Wattch [10] extensions to the SimpleScalar toolset. New tools and techniques, such as the use of instruction-level power analysis in microarchitectural simulators [11–13] have also enabled faster power estimation, with adequate accuracy on embedded processor architectures.

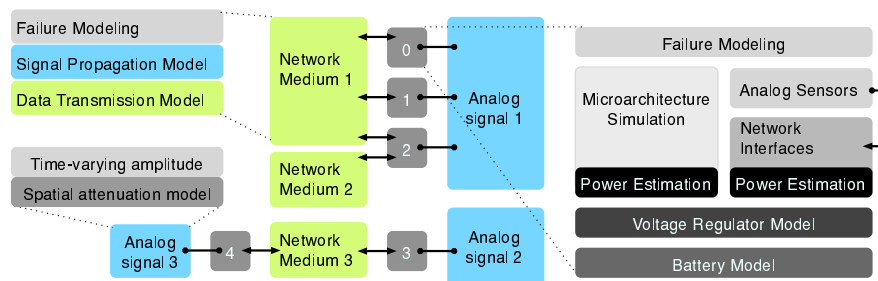


Fig. 1. Illustrative example of simulator organization. For any collection of nodes (processing element + memory + energy source), the separate instantiation of *network interfaces* and *network media* allows the creation of rich communication topologies. A collection of definitions of signal sources can similarly be used to create rich topologies of physical phenomena.

3 Simulation Environment Architecture

The Sunflower simulator comprises components for modeling five primary entities: (1) processor microarchitecture, (2) networking between embedded systems and the attendant power dissipation, (3) battery systems (electrochemical cells and voltage regulators necessary to provide stable voltages to electronics as the battery terminal voltage declines with battery discharge), (4) failures in computation and communication, and (5) modeling the physics of signal propagation in the environment in which computation occurs. Each instantiated node has an associated location in three-dimensional space, which is used in the modeling of signal propagation and attenuation with distance, described in Section 3.6. Figure 1 illustrates these facilities with an example. In the Figure, five complete embedded systems, composed of processors and their peripherals, such as network interfaces, along with their energy sources, are shown. Each modeled embedded system (henceforth, *node*), is instantiated with one *network interface* (e.g., nodes 0, 1, 3 and 4) or more (e.g., two network interfaces for node 2). These network interfaces are attached to simulated *communication-* or *network media*. Three external *physical phenomena* (analog signals 1, 2 and 3) are instantiated, and these could be configured to model, e.g., an acoustic signal or a light source, by specifying signal propagation and attenuation models, as described in Section 3.6.

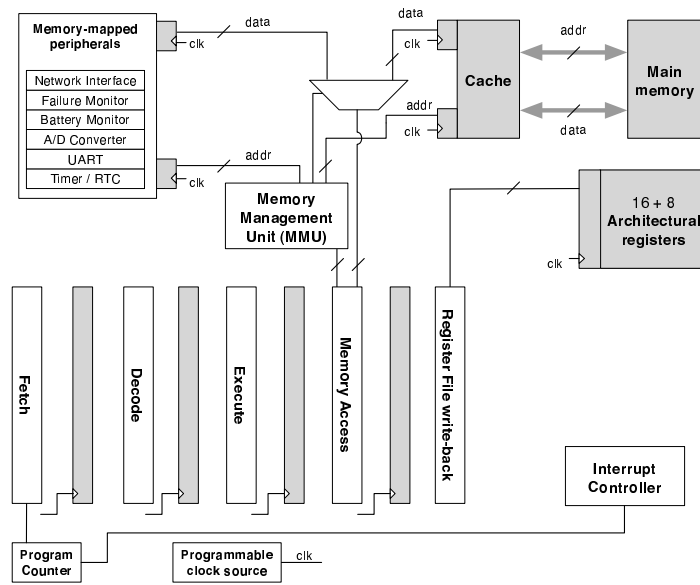
3.1 Instruction Execution

The simulator models two complementary instruction set architectures (ISAs): a 16-bit RISC ISA, the Texas Instruments MSP430, and a 32-bit RISC ISA, the Renesas SH; the Renesas SH ISA simulation core is adapted from our earlier work [13]. The default microarchitecture configurations for the two modeled ISAs are shown in Figure 2.

For each microarchitecture, the simulator models the movement of instructions through the pipeline latches, as well as signal transitions on the address/data buses and register file ports. The modeling of these structures facilitates two things. First, it enables the estimation of signal switching activity in these structures (Section 3.3), which account for a large fraction of the dynamic power consumption in embedded processors. Second, it enables the simulation of logic upsets, both single-event upsets (SEUs) and multi-bit upsets, as described in Section 3.5.

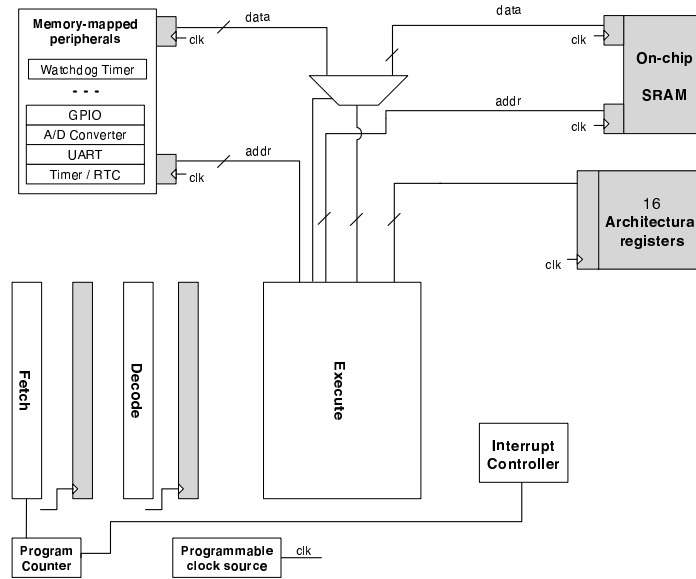
The default microarchitecture configuration for the 32-bit ISA model includes a shared instruction / data cache. The cache size, block size and associativity are simulation parameters, and an implementation of an LRU replacement algorithm is provided. The default 32-bit ISA model also includes a Translation Lookaside Buffer (TLB) implementation based on that in the Renesas SH3. Like the cache, it is configurable by number of entries, set size and associativity.

Asynchronous hardware *interrupts* and synchronous software *exceptions* constitute an important part of program execution in many embedded software applications. In these systems, hardware interrupts are usually generated by peripherals within or external to the processor, such as communication interfaces,



■ : Structures modeled at bit-level, enabling monitoring of signal transition activity and logic upset modeling

(a) Default configuration of the modeled, 32-bit architecture, employing the Renesas SH ISA.



■ : Structures modeled at bit-level, enabling signal transition activity and logic upset modeling

(b) Default configuration of the modeled 16-bit architecture, employing the TI MSP430 ISA.

Fig. 2. Default configuration of the modeled microarchitectures.

timers, and the like. Software exceptions on the other hand are also common as they are used to implement the boundary crossing between applications and an operating system kernel. Interrupts and exceptions are modeled for both architectures. In the case of the 32-bit architecture, the interrupt sources include timer interrupts (generated by the simulator at configurable intervals), low battery interrupts (described in Section 3.4) and several communication interface specific interrupts (described in Section 3.2). For the 16-bit ISA, the simulation environment models the signals at the I/O pins. The processor's I/O pins may be configured to generate interrupts on *level* or *edge* triggers, thus any analog signal configured outside the modeled processor (described in Section 3.6) may be used to trigger interrupts. Applications executing over the simulator which desire to interact with peripherals that generate interrupts, must therefore link in an assembly language low-level interrupt handler, to save machine state upon triggering the interrupt, and restore it after completion.

To enable the simulation of common benchmark suites, the applications within which often depend on executing above an operating system, the Sunflower simulation environment can intercept software exceptions corresponding to system calls. The exception value along with the machine state (according to the manufacturer and compiler defined application binary interface (ABI)) are then used to recreate the system call and pass it down to the simulation host's operating system; the results of the system call are similarly inserted into the machine state.

3.2 Communication Interconnect

There are a variety of communication protocols employed in embedded systems, ranging from serial protocols like RS-232, RS-485, SPI and I2C, to protocols like Ethernet (IEEE 802.3), 802.11a/b/g and the recent 802.15.3 short range low power MAC protocol. The approach taken for modeling networks in Sunflower attempts to accommodate the implementation, by the user, of any variety of protocols.

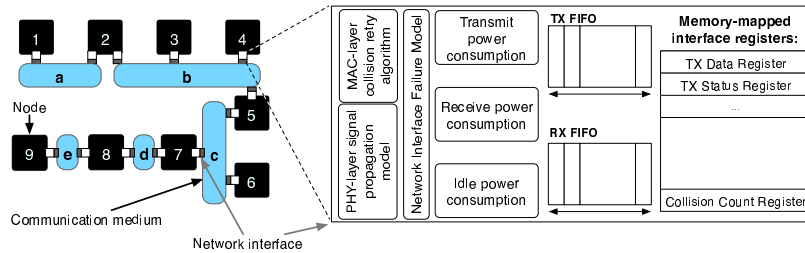


Fig. 3. Modeling of communication networks is separated into the modeling of *network interfaces*, connected to *communication media* to form communication topologies.

Communication between devices in the Sunflower framework may either be through a modeled MAC-layer communication interface, or by direct modula-

tion of an analog signal as described in 3.6. This modeling of communication comprises two parts. First, the interface to communication, as driven by computation, is a *network interface* peripheral. Each processor may be configured with multiple instances of such network interfaces. Each interface is parameterized by transmit and receive FIFO queue sizes, transmit, receive and idle power consumption, as well as a choice of collision back-off algorithms. Second, network interfaces must be explicitly connected to (at most one) *communication network medium*, at which point they inherit their remaining properties—communication speed, frame size and physical layer characteristics—from the medium to which they are attached.

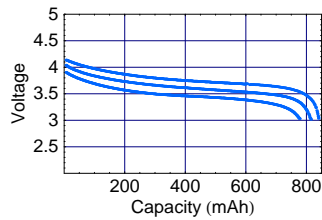
A link failure probability can be selected for each communication medium, as well as the maximum number of simultaneous transmissions permitted on a medium. The latter permits the modeling of either a single access medium accessed via MAC layer protocols that perform, e.g., *Carrier-Sense Multiple Access with Collision Detection (CSMA/CD)*, or a multiplex medium. Network topologies are created by instantiating network media, and connecting to them network interfaces of nodes that must communicate over the same channel; this separation of network interfaces and network media enables the modeling of arbitrary sophisticated interconnection topologies. Figure 3 illustrates the organization of an example network. Applications being modeled within the microarchitecture simulation interact with the network interface through a set of memory-mapped registers and generated interrupts. The figure depicts an example network comprising nine nodes, connected in a topology consisting of 5 disjoint communication media. Some of the nodes, e.g., 2, 5, 7, and 8 have multiple network interfaces, and are attached to multiple media through these interfaces. Some media, e.g., **b** and **c** are “multi-drop” or shared links, while others (**a**, **d** and **e**), are point-to-point.

In addition to network traffic driven directly by computation being simulated within the framework, it is also possible to use real-world MAC-layer network traces in simulation studies (e.g., as gathered on networks using the open source Ethernet tool). Traces of the network communications obtained during simulation, resulting from the dynamic behavior of simulated benchmark execution, can also be dumped to disk.

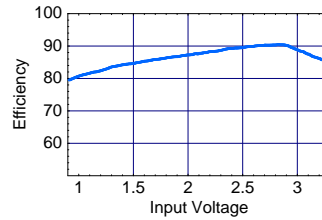
3.3 Modeling Power Consumption

The simulator includes a set of complementary methods for power estimation. By modeling the movement of instructions through the microarchitecture, it is possible to obtain estimates of signal switching activity in many microarchitectural structures (Figure 2). These reports can be used in comparative analysis of dynamic power dominated systems; if the capacitances of the structures can be determined, they can be used to estimate actual dynamic power consumption.

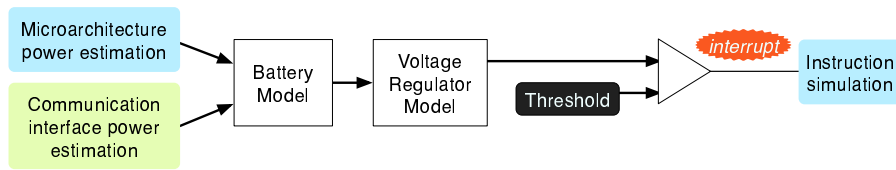
The second, coarser-grain facility for power estimation is an instruction-level power model developed as part of our earlier research [13]. This model involved characterizing the average power for each instruction in the ISA through actual



(a) Battery voltage versus charge, for the Panasonic CGR17500 (drain currents, top to bottom: 156mA, 780mA and 1560ma).



(b) Voltage regulator efficiency versus battery terminal voltage curve, for the TI TPS61070.



(c) Interaction between battery subsystem model and microarchitecture simulation.

Fig. 4. Examples of manufacturer supplied characterization data provided with the Sunflower simulator ((a) and (b)), and illustration of battery subsystem modeling organization (c).

hardware measurements, and incorporating these measurements into the simulator as a lookup table. During instruction simulation, the lookup table is used to provide power estimates based on the cycle-by-cycle mix of instructions. It has been shown [11, 12] that for embedded processors, which usually have simple in-order pipelines, such instruction-level power estimation provides acceptable accuracy in power estimation. The instruction-level power analysis characterization for the Renesas SH employed in Sunflower, has previously been verified against hardware measurements [13]; we are currently in the process of obtaining similar instruction-level power characterization for the TI MSP430.

In the third and coarsest grain power estimation facility, the average active and sleep power consumption for the CPU and network interface are provided by a user, obtained either from empirical measurements or from a manufacturer’s data sheet. During simulation, as the simulated processor transitions between active and sleep modes, or engages in communication, the associated power consumption is fed into the power estimation and battery modeling framework. Unlike workstation and server class systems, many embedded systems spend most of their time idle. As a result, their overall energy consumption is largely dictated by the fraction of time they are active versus idle (in sleep mode). This coarse-grained power estimation, in conjunction with battery models, can therefore still provide a good indicator of system lifetime.

The net current drawn by the CPU and all its peripherals, based on the above power estimation methods, is fed to a model for a battery subsystem, made up of a voltage regulator (DC-DC converter) model, and a battery model.

Table 1. A list of some of the commercial electrochemical batteries and voltage regulators for which models are supplied with the Sunflower simulation framework. Parameters for additional models are easily supplied at simulation time.

<u>Modeled Electrochemical Batteries</u>	<u>Modeled Voltage regulators</u>
Panasonic CGP345010, CGP345010g CGR17500, CGR18650HM.	Dallas Semiconductor MAX1653, TI TPS6107x, TPS6110x, TPS6113x.

3.4 Battery Subsystem Modeling

Embedded systems are often powered by electrochemical batteries such as Lithium-ion (Li-ion), Li/MnO₂ or NiMh cells. The terminal voltage across these cells varies with their state of charge and discharge current (illustrated in Figure 4(a) for a Panasonic CGR17500 battery, at 20 °C), and they are thus usually connected to the circuits they drive through *voltage regulators*. Voltage regulators incur energy conversion losses, and their efficiency depends on their load current, as well as the battery terminal voltage; this is illustrated for a TI TPS61070 voltage regulator in Figure 4(b), for a load current of 5mA and output voltage of 3.3V. As a result, of these phenomena, two systems with identical average power consumption, but with different time-varying power consumption profiles, may witness different battery lifetimes.

In order to capture the effect of simulated system power consumption, and its variation over time, on battery lifetime, the Sunflower simulation environment enables the modeling of both batteries and the voltage regulators that are used to interface them to the systems they drive (Figure 4(c)). The simulator incorporates actual measured characteristics supplied by the manufacturers of batteries and regulators, into a discrete-time model adapted from [14]. The default set of such battery and regulator models supplied with the simulator are listed in Table 1. The curves shown in Figure 4 are plotted from this characterization data that is used by the simulation environment.

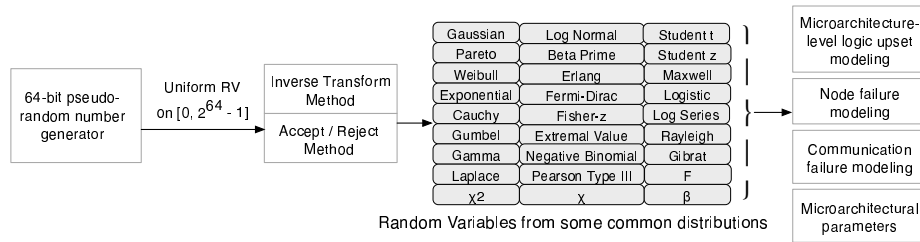


Fig. 5. A high periodicity 64-bit pseudo random number generator is used as the basis for generating random variables from a large set of distributions. In addition to the distributions listed here, a distribution having a “bathtub” shaped hazard function is also provided.

3.5 Modeling Processor and Interconnect Failures

Runtime faults in processing elements might be due to causes ranging from design errors, defects introduced during the manufacturing process and device aging, to external phenomena such as high energy particles (e.g., alpha particles). These faults, if not masked by fault-tolerance or fault-avoidance techniques, may manifest themselves as failures in the system.

The modeling of such faults in hardware may be performed using a variety of approaches. *Bit-level modeling of logic upsets* enables investigation of microarchitectural techniques for reliability-aware architectures. In most embedded systems, it is undesirable for a processor to get “wedged” or “stuck”, as such systems often control critical infrastructure (e.g., the anti-lock braking system of a car). Many embedded processors therefore include a facility called a *watchdog timer* — a hardware peripheral that must be accessed by software at regular intervals, failure to do so causing a system reset. The modeling of *whole system-failures* is thus also of interest, due to the possible occurrence of such watchdog timer resets.

Underlying both the modeling of bit-level and whole-system failures is the generation of random events. For simulations which have long duration, it is desirable for any pseudo-random sequences they employ to not repeat. While useful for some simple applications, the standard C library pseudo-random number generation routines provided with most operating systems do not provide sufficiently high periodicity when simulating billions of random events. Fortunately, the research literature contains better solutions for generating pseudo-random numbers, and we have incorporated one of these [15] in the simulator.

Pseudo-random number generators typically generate values uniformly distributed on some support set. However, during systems evaluation, it is often desirable to use random numbers drawn from some other distribution, such as, e.g., a Gaussian, χ^2 , or a heavy-tailed distribution like the Pareto distribution. Standard textbook methods [16] facilitate transforming random variables taken from one distribution (e.g., uniform) to obtain random variables drawn on a different distribution (e.g., exponential). The Sunflower simulation environment implements the generation of random variables from over twenty different distributions, shown in Figure 5. These distributions (with appropriate parameters,) can be used by a system architect to define the distribution of time between failures, duration of failures, or locations of logic upsets. When modeling bit-level logic upsets, the structures shaded grey in Figure 2 are treated as a single large bit vector. A bit in this vector is then forced to a logic 1 or logic zero, with probability across the bits determined by the chosen distribution.

Failures in communication can be induced either by directly configuring a communication link with a failure probability and duration distribution, or by defining time-varying external noise signals which interfere with the physical layer communication model (Section 3.6) associated with a communication link. During failures of the former kind, nodes that attempt to access the medium via a connected network interface will incur a *carrier-sense* error at the network interface. For the latter kind of failures, random bit errors are introduced into the transmitted data if the signal to noise ratio falls below the user-specified

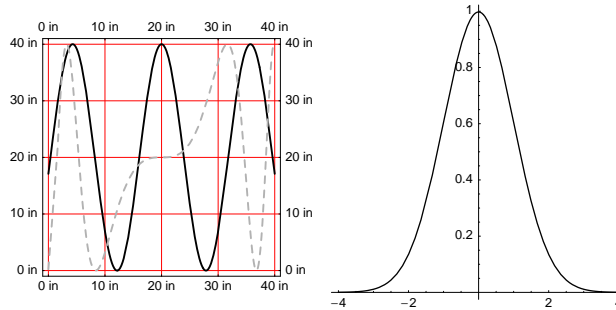


Fig. 6. Trajectories of two signal sources (solid black line and dashed gray line) across a 40"×40" area (left), and their attenuation with radial distance (right). The simulated target velocity was 10" per second.

Table 2. Relevant simulation configuration parameters.

Network Parameter	Value	Other Parameters	Value
Bit rate	38.4 Kb/s	Voltage regulator	Maxim MAX1653
MAC layer frame size	128 bytes	Electrochemical cell	Panasonic CGR-18
Propagation model	Ricean	Battery capacity	0.1 mAh
TX power	89 mW	CPU active/sleep power	26.4 mW / 49μW
RX power	33 mW	Clock frequency	4 MHz

tolerable minimum. Such corrupted frames will lead to the generation of frame checksum errors (and the associated interrupt) at the MAC layer.

3.6 Modeling Analog Signals External to Processing Elements

The Sunflower framework enables the simulation of time-varying analog signals external to an embedded system, alongside microarchitecture, power and communication modeling. These real-valued analog signals are accessible to the modeled computation through the A/D converter peripherals. The modeling of analog signals consists of four components: (1) the modeling of the time-variation of signals; (2) signal location in three dimensional space; (3) signal attenuation with radial distance; (4) constructive and destructive interference between signals.

A signal propagation model may be associated with a communication medium, enabling the modeling of some aspects of a network's physical (PHY) layer. This can be used, for example, to model the signal propagation properties of a wireless channel. While it is widely understood that there is no single best model for capturing wireless network behavior in simulation, this approach lets the user of the simulation environment decide what signal propagation model to use.

4 Example

To illustrate the use of Sunflower in studying the behavior of an embedded application, this section presents an evaluation of a cluster formation and object

tracking algorithm, *Distributed Aggregate Management (DAM)* and its extension, *Energy Based Activity Monitoring (EBAM)* [17]. Errors in the results of these algorithms can be exactly quantified, and we utilize this fact to demonstrate how traditional simulation speedup techniques, if applied naively, can introduce errors in the behavior of the simulated application. These errors result from the need, when simulating both instruction execution and analog signals external to the processor, to keep the passage of time synchronous across the two domains.

Figure 6 shows the two target (light source) trajectories, and the light intensity attenuation profile employed in subsequent simulations¹. Since these trajectories may overlap, when a single location is reported by simulation, it may be due to two overlapping actual targets. The simulated network consisted of 25 nodes, located at the grid points in the figure. The nodes and their network interfaces were configured at simulation time to model the computation performance, network speed and power consumption properties listed in Table 2. The DAM and EBAM algorithms were implemented in C and Renesas SH assembly language (for parts of the interrupt handler), and compiled with the GCC tool-suite. The microarchitectural simulation of this code, on each simulated node, interacts with the modeled sensor and networking peripherals. The values read off sensors are driven by the simulator’s analog signal modeling, and interaction with the network peripherals drives the simulator’s communication network models. In the investigations that follow, each configuration of the algorithm was simulated until the modeled battery (sized to constrain simulation time) was fully depleted. Each data point in the following graphs thus corresponds to the simulation of over 150 million instructions.

4.1 Measuring Benchmark Error

The benchmark output from each architectural simulation includes a list of reported target locations with timestamps. Since the actual trajectory of the targets provided as input to simulation is known, the error in reported target locations can be computed. Three different error metrics were computed and are reported in what follows.

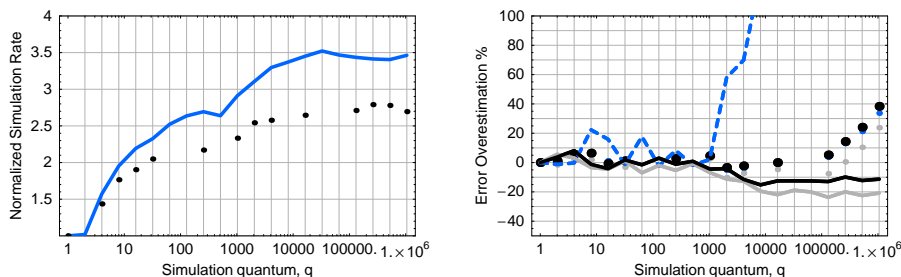
Optimistic error (oerr) Whenever the simulation output, resulting from architectural simulation, indicates a reported target location, the true target locations *at that time instant* are determined. Then, for each *true* target location, the closest *reported* target location is determined. The Euclidean distance between this pair is computed, removing the reported target from further consideration. If there are more reported target locations than true target locations, such excess reports are discarded.

Optimistic error with uniform sampling (o-uerr) When the error computation is performed as above, the error is only computed when the results of simulation report a target location. This does not capture the fact that during certain periods of the application, no target location is reported when one or

¹ While the trajectories shown here are *regular*, the modeling of arbitrarily irregular trajectories is supported.

more should be. In calculating the optimistic error with uniform sampling, a list of true target locations is generated for each sampling period of the application. For each such period, all reported locations falling within the period are grouped together. The error is then calculated as described above. If no reported target locations exist for a period, the optimistic error is set to zero.

Pessimistic error with uniform sampling (p-uerr) In the pessimistic error computation, when the list of true target locations has been exhausted, the process is restarted for the remaining reported target locations. The pessimistic error will therefore be large when the application reports an excessive number of target locations compared to true locations (the optimistic error on the other hand ignores “over-reporting”).



(a) Simulation speedup with increasing simulation quantum: DAM (solid), EBAM (points).

(b) Location error overestimation versus simulation quantum, for DAM (solid), EBAM (points) and EBAM p-uerr (dashed).

Fig. 7. Effect of simulation quantum size on simulation speed and accuracy of measured system parameters. The error plot in (b) shows $oerr$ (light grey), $o-uerr$ (black) and $p-uerr$ (blue in color document, dark grey in greyscale). In (b), the data points for EBAM $p-uerr$ are coincident with those for EBAM $o-uerr$ (black points) and are thus not visible.

4.2 Effectiveness and Error of Simulation Speedup Techniques

A primary source of overhead in simulating systems comprising multiple processing elements, under the requirement of keeping their evolution in time synchronous, is the overhead of advancing each in time, in lock-step. In a simulator implementation, this implies that each processor is simulated for one clock cycle, and any relevant simulated state due for updating (e.g., external signal sources) are updated, before each processor is once more advanced for one clock cycle. An applicable simulation speedup method is to increase the number of cycles by which each modeled processor is advanced before returning to the main simulation loop. This number of cycles is referred to henceforth as the simulation *quantum*. Modeling analog signals external to the microarchitecture requires synchronization of the time-line of computation with these signals. Employing increasingly larger simulation quanta will therefore violate this synchronization, and possibly introduce errors in comparison to fine-grained interleaving. There

is therefore a tradeoff between simulation speed and the degree of fine-grained interleaving (and hence, simulation accuracy).

Effect of simulation quantum size on simulation error Increasing the quantum from 1 to n causes n instructions to be simulated on one CPU before the next, round-robin. This leads to a simulation speedup since the time spent in the outer simulation loop is reduced. The trend in simulation speed as n is increased from 1 to 2^{20} , is shown in Figure 7(a), for the DAM and EBAM benchmarks; large quantum sizes can provide simulation speedups of up to 250%. The difference in speedup obtained for the two benchmarks is a result of their differing communication patterns.

Relaxing the synchronization of simulated time across simulated embedded systems in a network may however introduce errors in the simulation of an application. This phenomenon is illustrated in Figure 7(b). As can be seen in Figure 7(b), the error is also dependent on the application. The first source of error in using large fixed quantum sizes, is the disruption of the correct fine-grained interleaving of communication. For communication intensive benchmarks, employing large fixed quantum sizes is therefore undesirable. The second source of error arises from de-synchronization of the models for components external to the CPU (e.g., external modeled analog signals) during each quantum.

5 Summary and Future Work

This paper presented Sunflower, a simulation environment for networks of failure-prone, battery powered embedded systems. It models computation at the microarchitecture level, communication at the MAC and physical layers, failures in devices and communication media, and external analog signals, their location, and motion, in 3-dimensional space. Three complementary means of power estimation are provided, and instantaneous power estimates during simulation are fed into a model for a battery system consisting of a voltage regulator and an electrochemical cell. The simulation environment is implemented in ANSI C, and is available in source form, with pre-compiled binaries for several platforms including Linux, MacOS X, and Windows.

Based on the TI MSP430 architecture modeled in the simulation framework, we have developed a research hardware prototype, that we intend to make publicly available. We are currently in the process of performing detailed characterization of this prototype, and integrating our measurements into the simulation framework. It is our hope that this will enable future detailed investigations of microarchitectures for existent and emerging embedded platforms, that can be verified against cheaply available research hardware.

References

1. Reinhardt, S.K., Hill, M.D., Larus, J.R., Lebeck, A.R., Lewis, J.C., Wood, D.A.: The wisconsin wind tunnel: virtual prototyping of parallel computers. In: SIGMETRICS '93: Proceedings of the 1993 ACM SIGMETRICS conference on Measurement and modeling of computer systems, New York, NY, USA, ACM Press (1993) 48–60

2. Burger, D., Austin, T.M.: The simplescalar tool set, version 2.0. *SIGARCH Comput. Archit. News* **25**(3) (1997) 13–25
3. Hewlett-Packard Research Compiler and Architecture Research group: Trimaran HPL-PD simulator. (In: <http://www.trimaran.org/> (accessed June 2006))
4. Perez, D.G., Mouchard, G., Temam, O.: Microlib: A case for the quantitative comparison of micro-architecture mechanisms. In: *MICRO 37: Proceedings of the 37th annual IEEE/ACM International Symposium on Microarchitecture*, Washington, DC, USA, IEEE Computer Society (2004) 43–54
5. Vachharajani, M., Vachharajani, N., Penry, D.A., Blome, J.A., August, D.I.: Microarchitectural exploration with liberty. In: *MICRO 35: Proceedings of the 35th annual ACM/IEEE international symposium on Microarchitecture*, Los Alamitos, CA, USA, IEEE Computer Society Press (2002) 271–282
6. Rosenblum, M., Herrod, S.A., Witchel, E., Gupta, A.: Complete computer system simulation: The simos approach. *IEEE Parallel Distrib. Technol.* **3**(4) (1995) 34–43
7. Reinhardt, S.K., Dreslinski, R.G., Hsu, L.R., Lim, K.T., Saidi, A.G.: Tutorial: Using the m5 simulator. In: *ISCA '06: Proceedings of the 33rd annual international symposium on Computer architecture*, New York, NY, USA, ACM Press (2006)
8. Magnusson, P.S., Christensson, M., Eskilson, J., Forsgren, D., Hällberg, G., Högberg, J., Larsson, F., Moestedt, A., Werner, B.: Simics: A full system simulation platform. *Computer* **35**(2) (2002) 50–58
9. Skadron, K., Stan, M.R., Sankaranarayanan, K., Huang, W., Velusamy, S., Tarjan, D.: Temperature-aware microarchitecture: Modeling and implementation. *ACM Trans. Archit. Code Optim.* **1**(1) (2004) 94–125
10. Brooks, D., Tiwari, V., Martonosi, M.: Wattch: a framework for architectural-level power analysis and optimizations. In: *ISCA '00: Proceedings of the 27th annual international symposium on Computer architecture*, New York, NY, USA, ACM Press (2000) 83–94
11. Tiwari, V., Malik, S., Wolfe, A.: Power analysis of embedded software: a first step towards software power minimization. *IEEE Trans. Very Large Scale Integr. Syst.* **2**(4) (1994) 437–445
12. Sinha, A., Chandrakasan, A.P.: Jouletrack: a web based tool for software energy profiling. In: *DAC '01: Proceedings of the 38th conference on Design automation*, New York, NY, USA, ACM Press (2001) 220–225
13. Stanley-Marbell, P., Hsiao, M.: Fast, flexible, cycle-accurate energy estimation. In: *Proceedings of the 2001 international symposium on Low power electronics and design*, ACM Press (2001) 141–146
14. Benini, L., Castelli, G., Macii, A., Macii, E., Poncino, M., Scarsi, R.: A discrete-time battery model for high-level power estimation. In: *Proceedings of the conference on Design, automation and test in Europe, DATE'00.* (2000) 35–39
15. Nishimura, T.: Tables of 64-bit mersenne twisters. *ACM Trans. Model. Comput. Simul.* **10**(4) (2000) 348–357
16. Ross, S.M.: *Simulation*. Academic Press, San Diego, CA (2001)
17. Fang, Q., Zhao, F., Guibas, L.: Lightweight sensing and communication protocols for target enumeration and aggregation. In: *MobiHoc '03: Proceedings of the 4th ACM international symposium on Mobile ad hoc networking & computing*, New York, NY, USA, ACM Press (2003) 165–176