# Dynamic Voltage and Frequency Scaling For On-Demand Performance and Availability of Biomedical Embedded Systems

Dejan Raskovic, *Member, IEEE*, and David Giessel, *Member, IEEE*

***Abstract*—The goal of the study presented in this paper is to develop an embedded biomedical system capable of delivering maximum performance on demand, while maintaining the optimal energy efficiency whenever possible. Several hardware and software solutions are presented allowing the system to intelligently change the power supply voltage and frequency in runtime. The resulting system allows use of more energy-efficient components, operates most of the time in its most battery-efficient mode, and provides means to quickly change the operation mode while maintaining reliable performance. While all of these techniques extend battery life, the main benefit is on-demand availability of computational performance using a system that is not excessive. Biomedical applications, perhaps more than any other application, require battery operation, favor infrequent battery replacements, and can benefit from increased performance under certain conditions (e.g., when anomaly is detected) that makes them ideal candidates for this approach. In addition, if the system is a part of a body area network, it needs to be light, inexpensive, and adaptable enough to satisfy changing requirements of the other nodes in the network.**

***Index Terms*—Biomedical computing, electrocardiography, embedded systems, microcontrollers, on-demand performance.**

## I. Introduction

Designers of high-availability embedded systems, such as those used in biomedical applications, have several fundamental constraints that guide their design process. These systems must provide deterministic results throughout their deployment life and must be able to react to rapidly changing environmental variables. For example, detection of a critical event might generate the need for increased sampling and processing. If they are part of the body area or larger network, these systems often spend a lot of energy communicating with other nodes in the network, or responding to service requests from other nodes in the network. On the other hand, they need to be capable of prolonged remote deployment, especially systems intended to be implanted in a patient, or to be light enough to be worn without discomfort (for example, a Holter monitor). To meet these contrasting goals, such systems are required to have highly accurate timing, minimal energy consumption, and adaptive processing capabilities.

Manuscript received March 8, 2009; revised June 18, 2009. First published September 22, 2009; current version published November 4, 2009.

D. Raskovic is with the Department of Electrical and Computer Engineering, University of Alaska Fairbanks, Fairbanks, AK 99775 USA (e-mail: raskovic@computer.org).

D. Giessel is with the Poker Flat Research Range, Fairbanks, AK 99775 USA (e-mail: d.giessel@ieee.org).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TITB.2009.2030181

To achieve these ends, several approaches can be taken. Of all the possibilities, we will briefly discuss the two limiting cases and leave to the reader the consideration of possibilities lying between these two. One option for such a system would be to use a relatively high-performance processor with reasonably low power consumption. One example of such processor is the Intel Atom [1]. This solution easily meets the requirements for accurate timing and processing capability, but results in a very expensive power budget requiring extensive use of low-power modes to achieve an acceptable battery life. However, this solution might have a significant impact on the ability of the system to meet timing requirements, as we will show later. The other limiting case would be to use a significantly lower performance microcontroller with extremely low power consumption. Such processors include the Texas Instruments MSP430 family [2], certain members of the ARM family [3], etc. This solution can provide very accurate timing due, in part, to architectural simplicity, as well as long battery life even with a very small battery. However, meeting peak processing requirements can be a challenge in this case.

In order to determine which end of this spectrum would be best suited to our application, we must quantify our fundamental design constraints, such as available energy (or battery size) and the threshold for timing skew. Let us consider a 3-V battery with a capacity of 2 Ah to be typical for our application. Under the best discharge conditions, such a battery will have a capacity of 6 Wh. Therefore, to meet a one-year service life requirement, our average power consumption must not exceed 685 $\mu$W. For a solution based on a high-performance processor, typical high-frequency power consumption is in the order of 1 W, with low-frequency consumption being in the order of 0.1 W [1]. Thus, without extensive use of low-power modes or processor shutdown, our battery life will fall hundreds of times short of its requirement. To get a rough estimate of what our timing precision must be, let us consider a device that samples ECG signals and analyzes ST-segment levels to predict ischemic events. Studies have shown [4] that a sampling rate of 125 Hz significantly reduces peak amplitude values of ECG signals, but it could be satisfactory under conditions when only the measurement of certain intervals is required. Statistical analysis conducted as a part of the same study showed no significant difference between signals obtained at sampling rates of 500 and 250 samples/s. Therefore, we have either 8 ms (1/125 Hz) or 4 ms (1/250 Hz) of time between two samples. Consequently, our acceptable timing skew, in order not to miss samples, should be in the order of several milliseconds, and preferably much less than that. As we have mentioned above, our higher performance processor option will require extensive use of sleep modes to meet its battery life requirements. Unfortunately, these devices will have a relatively significant time overhead associated with entering low-power modes. In addition to the time required to enter and leave the sleep mode, there will be a significant delay while the contents of RAM are copied to flash. For a typical microprocessor in this category, this time approaches or exceeds our total allowable time skew [1]. Thus, these higher complexity systems either require more energy than is available for our application, or might fail to meet the tight timing requirements because of the frequent use of sleep modes with a high latency penalty.

A system based on a low-power microcontroller can be typically designed to operate at a point that would make the battery life acceptable. This is possible partly because a typical microcontroller in this category takes far less than 10 $\mu$s to wake up from sleep mode [2]. Unfortunately, unlike the high-performance microprocessor, a low-power microcontroller might have difficulty in sustaining higher processing requirements associated with dynamic changes in sampling rate or implementation of advanced signal processing algorithms. There are at least two approaches to solving this problem for the aforementioned example. One is to assure deterministic processing time of less than 4 or 8 ms per sample. The other one is to design the algorithm in such a way that all processing tasks (typically done inside interrupt routines) can be preempted by sampling and timing tasks, and provide sufficient buffer storage to store the samples until they can be processed. The first approach typically leads to a more expensive, heavier system, while the second approach could lead to buffer overruns, creation of backlog in processing, and could potentially delay an important medical decision.

Our approach to solve this problem is to base the system on a low-power microcontroller that can satisfy the deterministic processing time requirements on demand, and to provide hardware and system software solutions, some of which are available on high-performance processors, that would allow it to operate most of the time at the optimal power level. Some of the techniques that we incorporate are adaptive voltage and frequency scaling, adjusting the operating voltage to the current battery voltage to reduce losses in the voltage regulator, and selective use of low-power modes.

## II. Power Optimization Techniques

### A. Node Optimization Using D.I.A.F. Method

In this section, we discuss a hardware–software codesign method of performance optimization for a system based on a low-power microcontroller. This system utilizes a diagnostic loop to determine the optimal operating point for each individual microcontroller at every possible load condition. This loop creates a 3-D map by plotting the performance measured in millions of clock periods per milliwatts for every desired voltage/frequency operating point. Because we use a pipelined microcontroller, and to follow the standard performance reporting practices, we will, somewhat incorrectly, refer to this metric as millions of instructions per second per milliwatt (MIPS/mW).

Our process of optimization is *dynamic* (it happens on the fly), *intelligent* (it automatically selects the ideal operating point under all conditions), and the *frequency* of the microcontroller is *adaptable* for any given load condition. Thus, we will refer to it as D.I.A.F. Its advantages include:

1) individual performance optimization regardless of differences in hardware (manufacturing, layout, etc.);
2) optimal performance and energy consumption at every operating point;
3) diagnostic loop that not only maps out optimal points, but also determines which operating point should be selected for any given network condition (such as varying node workload, percentage of remaining battery life, etc.)
4) coarsely populated initial operating map that gains resolution as a processor experiences different load conditions and operating points;
5) default operating points based on the datasheet ensuring reliable operation during initial power-ON conditions.

When determining the most efficient operating point for each sensor node in the network, our optimization algorithm takes into account some usual variables-–network traffic, required processing power, the current battery voltage, etc. The reason to include battery voltage into account is that, over the life of a battery, the battery voltage can be very far from the optimal voltage determined by our algorithm. Typically, this happens close to the beginning and the end of battery life. In these cases, the penalty in the voltage regulator, where the power dissipation typically depends on the difference between input and output voltage, can outweigh the benefit brought by the fact that the microcontroller is running at the optimal power supply voltage. In addition, our algorithm also accounts for variance between microcontrollers. This includes both small variations that always exist even among microcontrollers produced in the same batch, and possibly large variations caused, for example, by changes in the manufacturing process. In order to continue this discussion, we must first clarify how we define the "most efficient operating point."

At a zero frequency (e.g., pushbutton clocked), every microprocessor or microcontroller has both a voltage floor below which it will not return repeatable results (some gates will fail to switch) and a voltage ceiling above which excess gate current also leads to erroneous results. A chip's envelope of operational frequencies is contained within this voltage range. Low frequencies require only the floor voltage, while the highest frequencies require higher voltages either to the point where calculations are no longer correct or the chip's thermal maximum is reached. Both frequencies between zero and the chip's maximum frequency require a different minimum voltage to ensure reliable performance. For any given frequency, a chip will consume the least amount of power when operating at that local voltage floor. Therefore, to find the most efficient operating point for a given frequency, one only needs to find the minimum voltage required to operate at this frequency.

For any given chip, there also exists another most efficient operating point, which is the point where the chip achieves the highest MIPS/W ratio. This point can be found on the 3-D plot that will be shown later (see Fig. 3) and represents the operational point at which the chip can perform the maximum number of calculations per second per watt. A "ridge" can also be found on this plot, which is the peak performance (MIPS/W) at any given frequency. If very little or very much computational power is necessary, it may be beneficial to operate at a point along this ridge, rather than the peak of the ridge. The reason for this is that points on the low-frequency side of the peak consume less power, albeit less efficiently per calculation, while points on the high-frequency side of the peak enable the sensor node to process data in a time-critical fashion if the peak efficiency point is too low in frequency to accomplish this.

Low-power modes (specifically those on MSP430) provide an additional enhancement, as in these modes, power consumption can drop to very low levels [2]. This opens up a question of finding the optimal ratio between the time spent in the active mode and the sleep mode. This ratio, ofcourse, can be changed by changing the clock frequency, and therefore, by changing the amount of time spent in active mode. However, the amount of energy expended for each possible frequency must be taken into account. Although countless combinations exist, there is only one global optimum for any given case. The purpose of D.I.A.F. is not only to map out the range of ideal operating points for all frequencies, but also to select complimentary sleep mode and active mode duty cycles to maximize battery life while still delivering results in a time-sensitive and deterministic fashion throughout the network.

Several key components are required to implement the D.I.A.F. algorithm. These components include: an adjustable voltage source, an adjustable clock source, a diagnostic mechanism to ensure that the microcontroller is operating correctly, an extremely low-power mode at our disposal where power consumption drops to essentially zero relative to active mode operation (e.g., one of the low power modes in the case of MSP430), a way of measuring execution time, and additional nonvolatile memory space to store the processor performance map data. On a typical microcontroller, measuring the execution time is inexpensive in terms of processing time and resources. MSP430 typically provides about ten capture–compare registers associated with timers, and only one of them would be required for our algorithm, at a cost of several clock cycles per execution of a monitored code segment. This time overhead is negligible because, typically, only the portions of code that are on a longest, critical path should be monitored. In our example application, the execution time of these portions of code is measured in tens of thousands of clock cycles.

An adjustable voltage source provides one of the key control parameters for altering node power consumption. The adjustable voltage source in our self-optimizing sensor node (see Fig. 1) is created by combining the TPS63000 adjustable regulator [5] with the MCP41100 digital potentiometer [6]. As the digital potentiometer is varied, the output of the TPS63000 increases or decreases accordingly. The MSP430 interfaces with the MCP41100 via the serial peripheral interface (SPI) bus, allowing it to vary the power supply ($V$cc) in runtime, with a minimal delay (several clock cycles).

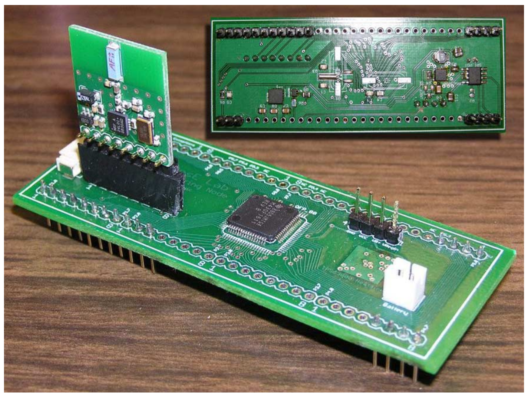


Fig. 1. Self-optimizing sensor node platform.

Adjustable frequencies are essential to optimizing node power consumption for a given processing requirement. There are several ways to provide variable-frequency clock sources, for example, by using single or multiple external crystals. In this case, we decided to use the internal digitally controlled oscillator (DCO) present on MSP430. DCO provides a broad range and high resolution of frequencies (from near dc to 21 MHz in the case of MSP430F2618). It has a few drawbacks relative to an external crystal-–higher frequency variation with temperature and voltage, as well as a slight increase in power consumption. However, the increase in available frequencies and linearity of these frequencies (external oscillator clocks can only be sourced in powers of two of their fundamental) outweighs drawbacks in most cases. Changing the frequency in runtime on MSP430 takes only 4–8 clock cycles. If precise synchronization among nodes is important, one of the standard solutions [7] can be used. We developed a specialized synchronization technique to keep the sensor nodes synchronized in the presence of extreme temperature variations [8]. The same solution can be used to solve problems with DCO instability, even though the clock frequency varies for different reasons.

To achieve the lowest possible power consumption in active mode, it is necessary to determine the lowest voltage required to operate at a given frequency, or conversely, the highest frequency that can be attained at a given voltage. In applications where reliability and optimal performance are not critical, these values can be taken from datasheet specifications. However, for this algorithm we want maximal optimization and minimal hard coding to achieve the dynamic nature of the system. In the diagnostic phase, and for some applications even in the deployment phase, we might not be concerned with staying inside the manufacturer-recommended operating window for a particular family of devices. More importantly, for biomedical applications, this approach allows us to increase the safe margins of operation and set them even more stringently than the manufacturer recommends. We want to stay inside the actual, empirical safe operating window for each specific device. Thus, a diagnostic utility that continually checks to see that the microcontroller is operating correctly as voltages and frequencies are varied is needed. The self-diagnostic performed by the microcontroller consists of operations that use some of the most critical hardware modules and a set of complex calculations whose results are compared to the "known" results. When voltage is gradually reduced below a safe level, arithmetic calculation error usually appears first, allowing the microcontroller to increase its voltage back to a safe level and save its current safe voltage/frequency setting to memory for future reference. This process can be performed prior to deployment of devices and would not affect the performance of the actual device in the runtime.

A code timing mechanism allows for overall active mode time to be monitored and recorded as a percent of available CPU time. This ensures that the node will increase its frequency to meet increased network demands only if necessary, thus preventing the delay of time sensitive data, or that it will decrease its frequency to a point of maximum efficiency, if network traffic or data processing requirements decrease. A typical microcontroller, including the MSP430, has a myriad of available timers that function in both active- and low-power modes, and can be used to very efficiently obtain this metric. Node resource usage is the fundamental environmental input parameter for the D.I.A.F. algorithm.

Having in mind limited resources in low-power microcontrollers, it is important for the power optimization algorithm to have a small memory footprint. The D.I.A.F. algorithm maintains a table of optimized operational points. Each entry in this table will need to have a one-bit field to specify whether this operational point has been optimized or not, a one-byte field for voltage settings (for a 256-step programmable potentiometer), and a field for frequency settings (7 bits for a DCO used in MSP430). In our case, this results in 2 bytes per entry. A good choice for a number of entries for our 16 MHz microcontroller is 8-–providing frequency steps of 2 MHz. Increasing the number of steps beyond this would be hard to justify because voltage differences between steps become too low. This small table, populated prior to runtime and kept in nonvolatile memory, contains frequency/voltage operational points taken from the datasheet. These values serve only as the starting point for the optimization process that can be done either before the deployment (using recorded input signals) or in the runtime (optimizing each frequency as they are encountered). For each portion of code that is being monitored, the node dynamically adjusts its operation mode whenever the execution time is above the threshold. Because we use 16-bit timers, the thresholds require only 2 bytes each. Rather than monitoring each subroutine in the program, which would quickly become too expensive (even at 2 Bytes per subroutine), it is more beneficial to monitor the entire critical path (or several paths) by measuring the total execution. This should be done only for portions of code that exhibit large variability in execution times. A detailed flowchart of the D.I.A.F. algorithm, as implemented on our sensor nodes, is given in Fig. 2.

To determine the potential performance of this algorithm, we carried out an extensive series of measurements across the range of processor frequencies and voltages listed in the datasheet [2]. Voltages and frequencies outside this range were also explored, as one of the touted benefits of this algorithm is the ability to operate even beyond the listed specifications, provided that small, but possible, risk of erroneous behavior is acceptable. Specifically, current was measured at a particular voltage and frequency with the processor in active mode. To precisely measure the current in the static mode, we used Agilent 34420A $\eta$V/$\mu\Omega$ meter, and to obtain the long-term current drain profiles, we used Agilent 66319D mobile communications dc-source instrument with the accompanying software. An abbreviated tabulation of the data, omitting frequencies outside the recommended values and those obtained using external oscillators, is shown in Table I. Blank cells indicate points where the processor was inoperable. Not surprisingly, as supply voltage increases, current draw increases. Similarly, as operating frequency increases, current draw increases. With just raw data at our disposal, it is difficult to select an ideal operating point. However, when we calculate instantaneous power for each of the points, and then plot (see Fig. 3) the number of clocks available per milliwatts (that we refer to as MIPS/mW for reasons mentioned before), some trends become obvious.

It is clear that the most efficient operating frequency from the standpoint of the number of calculations per unit energy for the MSP430F2618 microcontroller is at 8 MHz. At frequencies above and below this point, performance, as measured using this metric, diminishes. Most notably, it diminishes rapidly as frequencies approach dc. The primary reason for this is that below 1.5 V, the microcontroller will not operate even at frequencies approaching dc. Once this voltage floor is reached, power savings only come as a result of reduced clock frequency (a linear

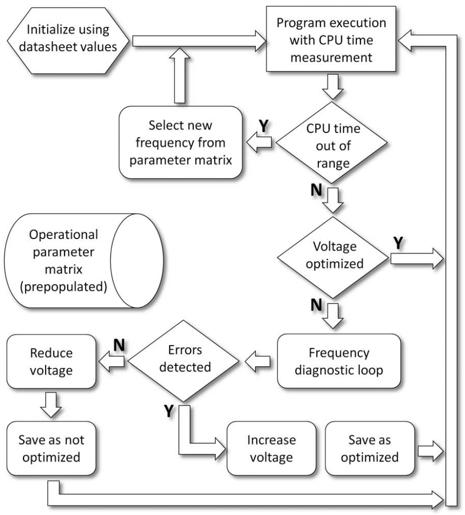

Initialize using datasheet values
Program execution with CPU time measurement
Select new frequency from parameter matrix
Y
CPU time out of range
N
Operational parameter matrix (prepopulated)
Voltage optimized
Y
N
Reduce voltage
N
Errors detected
Frequency diagnostic loop
Y
Save as not optimized
Increase voltage
Save as optimized

Fig. 2. Adjustable regulator flow chart.

TABLE I
CURRENT (IN MILLIAMPERES) AS FUNCTION OF VOLTAGE AND FREQUENCY

| Freq. (MHz) | VOLTAGE (V) | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | 1.5 | 1.6 | 1.7 | 1.9 | 2.1 | 2.3 | 2.5 | 2.7 | 2.9 | 3.1 | 3.3 |
| 16.35 | | | | | 8.0 | 8.5 | 9.2 | 9.9 | 10.6 | 11.3 | 12.0 |
| 12.00 | | | | 6.0 | 6.4 | 6.9 | 7.4 | 8.0 | 8.5 | 9.1 | 9.7 |
| 8.00 | | | 4.4 | 4.9 | 5.2 | 5.6 | 6.0 | 6.4 | 6.8 | 7.2 | 7.6 |
| 4.17 | | 2.9 | 3.4 | 3.8 | 4.0 | 4.2 | 4.4 | 4.7 | 4.9 | 5.2 | 5.5 |
| 2.08 | 2.0 | 2.4 | 2.8 | 3.1 | 3.2 | 3.4 | 3.6 | 3.7 | 3.9 | 4.0 | 4.2 |
| 1.03 | 1.7 | 2.1 | 2.4 | 2.7 | 2.8 | 2.9 | 3.0 | 3.1 | 3.2 | 3.3 | 3.5 |
| 0.51 | 1.6 | 1.9 | 2.3 | 2.5 | 2.6 | 2.7 | 2.8 | 2.9 | 3.0 | 3.1 | 3.2 |

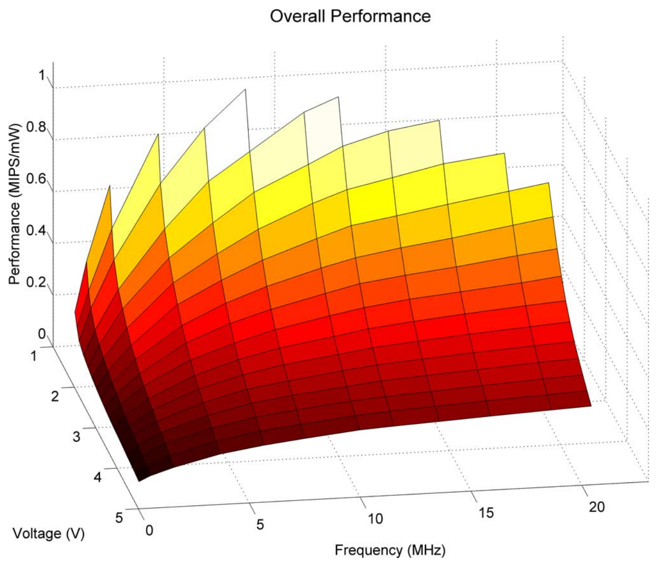


Fig. 3. Voltage versus frequency versus performance for the MSP430F2618 microcontroller.

gain in efficiency) and the larger $V^2/R$ gain cannot be achieved no matter how low the frequency is.

It is worth noting that this particular microcontroller is rated to 16 MHz maximum operating frequency, whereas previous versions of the MSP430 were rated to 8 MHz operation. Measurements performed by Dudacek and Vavricka [9] have shown that these older variants of the MSP430 family had a peak efficiency frequency of operation at 4 MHz or 50% of their rated maximum, and thus, it should not be too surprising that the newer 16 MHz parts are also most efficient at 50% of their rated maximum frequency.

## B. Battery Run-Down Model

When it comes to the actual practical system implementation, battery run-down considerations must also be taken into account. In order to provide a relatively realistic framework for evaluating node performance, we used an empirical battery run-down model. Our work was based on the measurements reported in [10], done to determine the total energy a typical CR2450 coin cell battery is capable of delivering to loads of varying average current draw. One of the key findings of this research was that for battery chemistries such as the one used in CR2450 (560 mAh rated at 2.6 V average), total useable energy is highly dependent on current draw. The datasheet discharge characteristic for this battery indicates a usable battery life of approximately 1600 h across a load impedance of 7.5 kΩ [11] resulting in a current draw of just 0.35 mA. Total energy delivered under these conditions is nearly two orders of magnitude higher than the amount of energy delivered when the current drawn is 12.2 mA on the regulated side of a 3.3 V step-up regulator (approximately 17.5 mA average on the input side of the regulator). In a similar, but somewhat less sophisticated setup, we extended these experiments to confirm the model behavior under conditions of interest in our system. We performed battery run-down tests using loads resulting in lower current drains than those

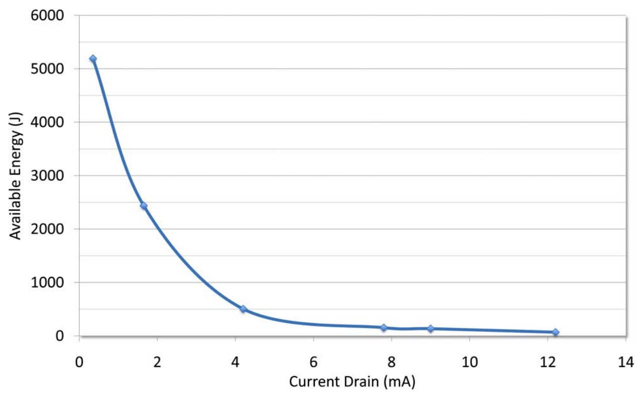


Fig. 4. Available battery energy versus average discharge current.

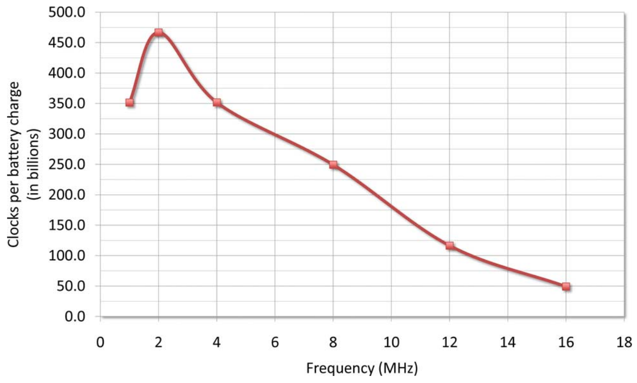


Fig. 5. Clock cycles per battery charge versus clock frequency.

used in [10], and recorded the results using an Agilent 66319D Mobile Communications dc-source instrument. A combination of these measurements produced an analytical model that we could then use to estimate available energy for any current drain in the tested range.

The expected total available energy from the battery for our test application running on an MSP430F2618-based system is shown in Fig. 4. The battery used for the experiment was a Duracell CR2450, but the results would be very similar for all batteries using lithium–manganese dioxide chemistry. The values on the $x$-axis correspond to average current draws when the system runs at various frequencies.

With a comprehensive model of our microcontroller power requirements and a battery run down profile, we can now determine the steady-state operational point that will provide the maximum number of microcontroller clock cycles (or calculations) over the life of the battery. We can estimate the usable amount of battery energy at our disposal for any given mode of steady-state operation. In this case, frequencies of operation between 1 and 16 MHz have been selected using current draw measurements for voltages that lie within the datasheet specifications for each frequency, respectively. Some of the measurements plotted in the previous section lie outside of the datasheet specifications, but these points do not alter the results fundamentally. The results of these calculations are plotted in Fig. 5.

The most notable result from this part of our research is that the optimal operating point with the additional insight of a battery run-down profile is no longer at 8 MHz, but instead, very clearly at 2 MHz. This is because the increased battery capacity due to lower current consumption outweighs the reduced efficiency of operating the microcontroller at this low frequency. This raises the question as to why 1 MHz does not yield even better system efficiency even though it still yields better battery life. Looking at the MIPS/mW efficiency metric, it is clear that there is a sharp decline in processing element efficiency when reducing frequency from 2 to 1 MHz. This is due to the processing element reaching its voltage floor at 2 MHz. Therefore, the decrease in current when lowering the frequency from 2 to 1 MHz is comparably smaller than when going from 4 to 2 MHz, and therefore, there is a sharp decrease in processor efficiency and only a comparably small increase in battery capacity. This result vividly illustrates the importance of considering the system as a whole when attempting to select the ideal point for maximum active-mode battery life.

## C. Related Work

Several similar techniques for higher performance processors have been developed. Probably the best known example is Intel's Enhanced SpeedStep technology [12]. It uses six preset voltages and frequencies to optimize power consumption for a given load condition. These steps are preset in a static matrix, and switching between them is determined by kernel-level software based on user-defined parameters. SpeedStep is, thus, adaptive, but not self-optimizing. A six-output voltage regulator (providing each of the voltage steps for the corresponding frequencies) is a fundamental hardware requirement, which may be prohibitively large and expensive for low-cost embedded systems. Similar in overall function, but quite different in actual implementation, real-time dynamic voltage-loop scheduling has been developed by Shao *et al.* as an approach for processor power optimization under varying load conditions [13]. This approach is of lower level than the approach used in D.I.A.F., and works by unrolling loops and comparing required and available execution times at a given frequency. CPU speed can then be preemptively increased or decreased (along with corresponding voltage) as necessary to meet load demands within specified timing constraints. This technique was demonstrated to be highly effective on higher power processors; however, the hardware resource overhead (while negligible for a desktop computing system) was determined to be prohibitive for an embedded environment such as the one considered in this paper.

## III. Discussion

Processing, storage, and timing requirements of mobile health systems [14] and wearable health-monitoring devices [15] are very application-dependant. To better illustrate a type of applications that could benefit from our D.I.A.F. approach, we give an example of an actual MSP430-based system, developed by us to process ECG signals in real time. The purpose of the system is to detect depressions in the ST-segment level of the ECG signal, because these depressions can be an indication of a possible ischemic event [16]. The system was tested using an annotated collection of ECG signals, the European ST-T database [17].

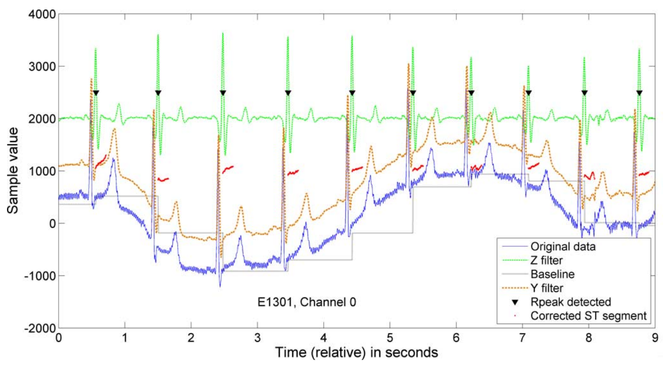


Fig. 6. Results of processing an original ECG signal—original data, filtered data, extracted baseline, corrected ST segment, and detected heartbeats are plotted.

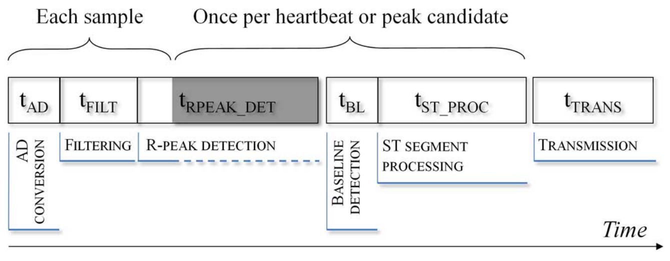


Fig. 7. Components of the ECG processing algorithm; execution times not shown in proportion.

Fig. 6 shows one 10-heartbeat section of a typical signal from that database. Obviously, the original signal is highly distorted from a signal processing point of view, and requires a lot of careful filtering and processing to preserve and extract the information about certain parts of the ECG signal. When implemented on a PC, the algorithm has no problem in achieving high levels of accuracy. However, when implemented on a microcontroller, certain sacrifices might have to be made in order to satisfy stringent timing requirements.

Without going into details, we show components of the ECG processing algorithm (Fig. 7). The total execution time depends on the particular path that processing of each sample takes through the algorithm, which, in turn, depends on the number and types of the events detected in the signal.

Fig. 8 illustrates variations in the processing time per sample for a set of procedures that constitute our critical execution path. The data shown in Fig. 8 represents the execution time as measured on a microcontroller running at 4 MHz. The peaks that are going outside of the graph in Fig. 8 represent the samples where the algorithm went through the longest path. The execution times for these cases are plotted on the same graph using a different scale, and show that the timing requirements could not be satisfied even if we run at 8 MHz (maximum frequency for the older series of MSP430 microcontrollers that would result in the execution times approximately half of what Fig. 8 shows) because our optimal sampling rate of 250 Hz leaves us with less than 4 ms for processing. Therefore, to achieve processing times of less than 4 ms, and avoid buffering and possible delays in sampling or making critical decisions, we would have to run at a highly inefficient frequency of 16 MHz (the execution time

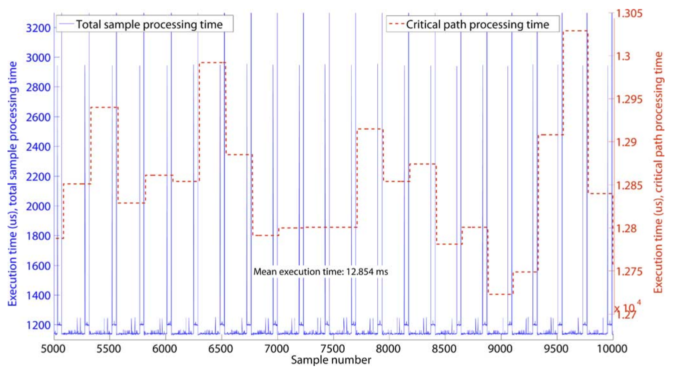


Fig. 8. Total processing time per sample and critical path execution time of the ECG processing algorithm.

in this case drops to about 3 ms). This would result in increased power consumption at all times, and would significantly reduce the battery life (see Fig. 5). Fig. 8 represents a "normal" case, where the periods of high processing demands are evenly distributed in time (once per heartbeat). Noisy signals can result in false detections resulting in high processing demands coming in burst, which could, without our approach, lead to buffer overruns and processing backlog. Most of the time, however, the execution time per sample is one or two orders of magnitude shorter-–typically, in the order of several hundred microseconds (less than 500 $\mu$s at 4 MHz, or less than 2000 clock cycles). This kind of processing time requirement can be easily satisfied while operating at our most energy-efficient operating point (see Fig. 5) with the microcontroller operating at 2 MHz without sacrificing the performance (2000 clocks would take 1 ms). The actual battery savings depend on the percentage of time spent at each operating point, which, in turn, depends on the path each signal takes through the algorithm. Our preliminary results show that for all the processed signals, at least 95% of samples can be processed while operating at 2 MHz. Because of the steep slope of the curve given in Fig. 5, it is obvious that savings could be significant, resulting in prolonged monitoring times, better processing algorithms, or both. Another example where this approach could lead to better health outcomes is that the ability to increase the processing power on demand, and therefore, reduce the processing times, allows us to store raw data on flash memory or transmit it wirelessly to other nodes in the personal area network even in critical health situations, when significant processing requirements would otherwise hinder our ability to do so.

Without using the solution presented in this paper, some sacrifices would have to be made. To satisfy strict timing requirements, we would either have to lower the sampling rate while risking the loss of certain features of the signal, use less-advanced signal processing algorithms, reduce the amount of data sent to the rest of the personal area network, or to switch to the more powerful system and bigger battery when designing a system. Our approach for dynamic voltage and frequency scaling gives us the ability to increase the processing power only when the application requires it, while keeping the overall system simple, cheap, and energy-efficient.

## REFERENCES

[1] *Intel Atom Processor Z5xx Series Datasheet,* Intel Corporation, Santa Clara, CA, 2009.

[2] *MSP430×2xx Family User's Guide,* Texas Instruments Incorporated, Washington, DC, 2008.

[3] ARM7TDMI-S, ARM [Online]. Available: http://www.arm.com/products/CPUs/ARM7TDMIS.html

[4] G. P. Pizzuti, S. Cifaldi, and G. Nolfe, "Digital sampling rate and ECG analysis," *J. Biomed. Eng.*, vol. 7, no. 3, pp. 247–250, Jul. 1985.

[5] *TPS6300x High Efficiency Single Inductor Buck-Boost Converter With 1.8-A Switches,* Texas Instruments Incorporated, Washington, DC, 2008.

[6] *MCP41XXX/42XXX Single/Dual Digital Potentiometer With SPI Interface*: Microchip, Inc., Bedford, MA, 2003.

[7] M. Maróti, B. Kusy, G. Simon, and Á. Lédeczi, "The flooding time synchronization protocol," in *Proc. ACM Conf. Embedded Netw. Sens. Syst.*, 2004, pp. 39–49.

[8] D. Raskovic, O. Lewis, and D. Giessel, "Time synchronization for wireless sensor networks operating in extreme temperature conditions," presented at the 41st IEEE Southeastern Symp. Syst. Theory, Tullahoma, TN, 2009.

[9] K. Dudacek and V. Vavricka, "Experimental evaluation of the MSP430 microcontroller power requirements," in *Proc. EUROCON 2007*, Warsaw, Poland, pp. 400–404.

[10] S. Park, A. Savvides, and M. B. Srivastava, "Battery capacity measurement and analysis using lithium coin cell battery," in *Proc. Int. Symp. Low Power Electron. Design*, Huntington Beach, CA, 2001, pp. 382–387.

[11] Duracell, Inc. (2009, Jan.). *CR2450 Lithium-Manganese Dioxide Batteries.* Duracell, Inc., Bethel, CT [Online]. Available: http://www.duracell.com/oem/primary/Lithium

[12] *Enhanced Intel SpeedStep Technology for the Intel Pentium M Processor*: Intel Corporation, Santa Clara, CA, 2004.

[13] Z. Shao *et al.*, "Real-time dynamic voltage loop scheduling for multi-core embedded systems," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 54, no. 5, pp. 445–449, May 2007.

[14] D. Raskovic, A. Milenkovic, P. deGroen, and E. Jovanov, "From telemedicine to ubiquitous M-health: The evolution of e-health systems," in *Biomedical Information Theory*, D. D. Feng, Ed. Burlington, MA: Academic, 2008, pp. 479–496.

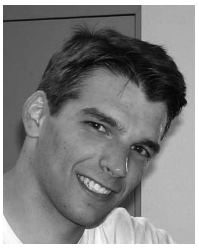
[15] D. Raskovic, T. Martin, and E. Jovanov, "Medical monitoring applications for wearable computing," *Comput. J.*, vol. 47, no. 4, pp. 495–504, Jul. 2004.

[16] N. Maglaveras, T. Stamkopoulos, C. Pappas, and M. G. Strintzis, "An adaptive backpropagation neural network for real-time ischemia episodes detection: Development and performance analysis using the European ST-T database," *IEEE Trans. Biomed. Eng.*, vol. 45, no. 7, pp. 805–813, Jul. 1998.

[17] European Society of Cardiology. European ST-T Database. European Society of Cardiology. Sophia Antipolis, France [Online]. Available: http://www.physionet.org/physiobank/database/edb/

**Dejan Raskovic** (S'95–M'04) received the B.S. (Dipl.Ing.) and the M.S. degrees in computer engineering from the University of Belgrade, Belgrade, Serbia, in 1993 and 1996, respectively, and the Ph.D. degree in computer engineering from the University of Alabama, Huntsville, in 2003.

He is currently an Assistant Professor of electrical and computer engineering in the University of Alaska Fairbanks, Fairbanks. His current research interests include embedded systems, energy efficient processing, and body area sensor networks.



**David Giessel** (S'03–M'07) received the B.S. and the M.S. degrees in electrical engineering from the University of Alaska Fairbanks, Fairbanks, in 2005 and 2009, respectively.

He is currently with the Poker Flat Research Range, Fairbanks Geophysical Institute, University of Alaska Fairbanks. His current research interests include building automotive control systems, cycling, and Austrian economics.