Going Beyond Worst-Case Specs with TEAtime

The timing-error-avoidance method continuously modulates a computer-system clock's operating frequency to avoid timing errors even when presented with worst-case scenarios.

Augustus K. Uht
University of Rhode Island

Virtually all engineers use worst-case component specifications for new system designs, thereby ensuring that the resulting product will operate under the worst conditions they can envision. However, given that most systems operate under *typical* operating conditions that rarely approach the demands of worst-case conditions, building such robust systems incurs a significant performance cost. Further, classic worst-case designs do not adapt to previous manufacturing conditions or current environmental conditions, such as increased temperature.

The *timing-error-avoidance* prototype provides a circuit and system solution to these problems for synchronous digital systems. TEAtime has demonstrated much better performance than classically designed systems and also adapts well to varying temperature and supply-voltage conditions. TEAtime works by increasing the operating frequency of the system clock until just before a timing error would occur, then slightly decreasing the clock frequency. These changes in clock frequency happen continuously. Low cost, TEAtime involves no software actions.

DIGITAL SYSTEM DESIGN

Most current digital systems are synchronous in that their state changes only in response to transitions of a systemwide clock signal, typically changing on the clock's rising edges, that is, from a logical 0 to a logical 1. Such systems work correctly only when the delay from a clocked part's output—the flip-flop—to the input of the same or other flip-flops is less than the *clock period*, the time between the clock's adjacent rising edges. All synchronous systems designed today use delays that assume worst-case environmental, operating, and manufacturing conditions.

Student engineers are often surprised when their first synchronous systems fail to operate as expected. This usually happens because they used typical part specifications during the system design. Thus, on a hot summer day, for example, before the due date at the end of the semester, the gates will have a greater than typical delay, the system clock period originally specified will be too short, and the system will fail.

The solution, as all practicing engineers know, requires using worst-case part specifications for system design. For example, the delay used for a gate will be its delay at the highest specified temperature and lowest supply voltage. Further, engineers build in even more latitude to allow for manufacturing variations. The resulting worst-case design provides a system that will operate under a wide range of operating and manufacturing conditions.

Unfortunately, given that the system usually operates under typical conditions, the worst-case approach incurs a severe performance cost: The system usually runs at a clock frequency much lower than necessary.

PC game enthusiasts frequently compensate for this decrease in performance by increasing the operating frequency of their systems far above what the manufacturers specify. This practice is dangerous, however, because the only way overclockers can know they have pushed their CPUs too far is when

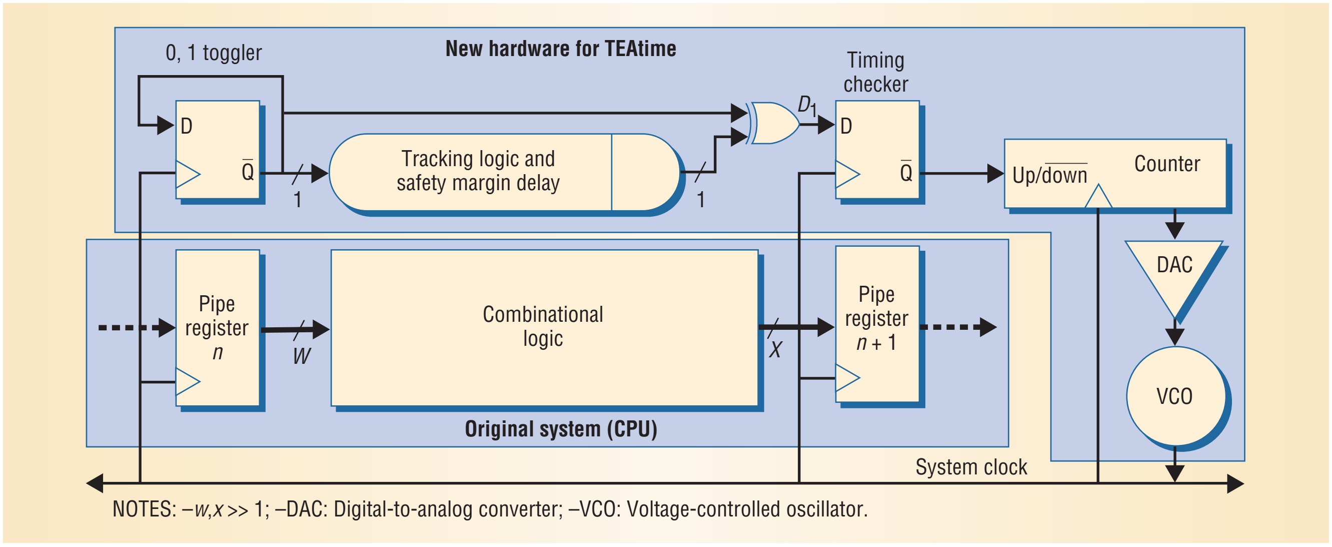


Figure 1. TEAtime CPU or digital system modifications. TEAtime adds a toggler, tracking logic and safety margin delay, timing checker, up-down counter, digital-to-analog converter, and voltage-controlled oscillator.

their systems experience a potentially catastrophic failure. For the casual user, constantly tuning the clock frequency while destroying data and even hardware offers an unsatisfactory tradeoff to enhancing performance beyond stock specifications. The situation becomes much worse when applications more critical to society than the latest shoot-em-up computer game are involved.

ALTERNATIVE APPROACHES

Researchers have sought ways to dynamically increase the clock frequency either without causing timing errors or, if errors do occur, by providing built-in methods for recovering from them. If the resulting solution is always active and always seeks the highest clock frequency possible, the system will also adapt to varying environmental and manufacturing conditions.

In one alternative to TEAtime, a system uses a microcontroller to periodically check the operation of the controller's adder[1] because this component causes the greatest delay in the system. The microcontroller performs the checking by propagating a signal through the adder's entire carry chain by, for example, adding a 1 to all 1s. If this operation provides an incorrect result, the system decreases the clock frequency. Unfortunately, the worst-case path through a system may not be through the adder and thus not be as easy to check. Another disadvantage of this approach is the need to modify the system software for the scheme to work.

A previous technique checked systems by letting errors occur.[2] The system then backed up the state to a known good state, decreased the clock frequency, and continued. This technique more than doubled the hardware needed in the system and was much harder to add to existing designs than anticipated. The recent Razor system[3] also allows errors to occur, then recovers from them. While the additional quantity of hardware needed is small, it is not simple. Also, extra pipeline bubbles and flushes occur, unlike in TEAtime. Large power savings are predicted; improved performance is not a goal.

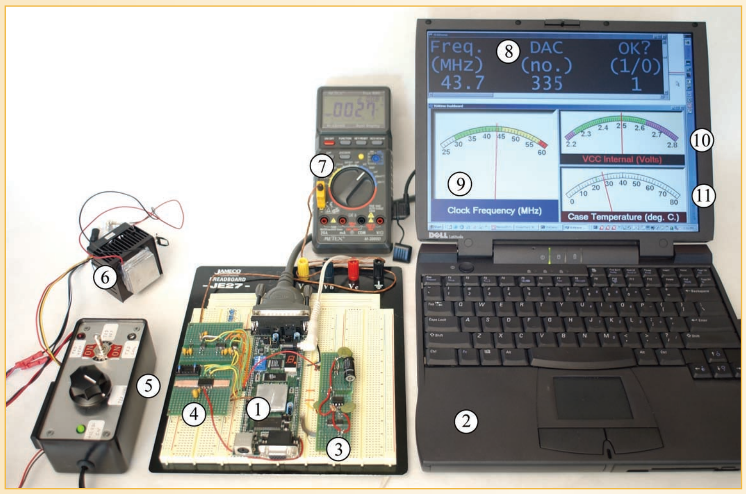
Some techniques modify the clock frequency based on varying constraints,[4] but these systems are *open-loop*: They either do not have a feedback system or do not include the operating frequency or clock period as part of the regulating system. Some systems throttle the clock based on power or temperature,[5,6] but the clock frequency does not increase above the usual worst-case design value.

Self-timed[7] and some other systems[8] use a ring oscillator to mimic the worst-case path delay through the system. Although this method resembles TEAtime, it does not use true tracking logic to mimic the actual circuitry along the worst-case path, thus it does not truly adapt to existing system conditions. This can either produce system failure in the worst scenario or lead to an overly conservative design that does not achieve the high performance that might be possible otherwise.

Asynchronous systems[9] sidestep the problem completely because they do not use a clock. Such systems operate as fast as the gate delays allow. Although this might seem to be an ideal approach, designing robust and error-free asynchronous systems is difficult, and few if any current computer-aided-design tools can generate them.

TEATIME

Figure 1 shows how TEAtime uses tracking logic to mimic the worst-case delay in a synchronous system. Normally, the tracking logic is a one-bit-wide replica of the worst-case path in the system, with a slight delay added to it that provides a safety margin for the system. The flip-flop at the input to the tracking logic is wired as a toggle flip-flop and clocked by the system clock, changing from 0 to 1 and 1 to 0 on alternate cycles. This provides a test signal for the tracking logic during every cycle of operation and ensures that the signal tests both types of transitions. The latter is necessary since



LEGEND
1. FPGA and XESS XSA-100 prototype card
2. PC host
3. Custom oscillator card with DAC and VCO
4. Custom card containing the PC host-controlled FPGA internal supply voltage
5. Custom thermoelectric device controller
6. Thermoelectric device assembly
7. FPGA case temperature meter
8. Controller program command-line window and data display
9. System clock frequency meter
10. VCCint meter
11. FPGA case temperature meter

Figure 2. TEAtime prototype with experimental and demonstration setup. The test computer's system clock has stabilized at about 44 MHz. The normal worst-case specified frequency is about 30 MHz. The system clock frequency meter (9) consists of four regions: White is part of the classical non-TEAtime operating region, green shows better-than-worst-case operating frequency, yellow shows the safety margin, and red shows system failure.

delays for the two transitions can differ.

The tracking logic output then goes through the safety margin delay. Next, the exclusive-OR gate normalizes the test signal for the timing checker flip-flip at the end of the chain—the final version of the test signal, D1, will always change from a 1 to a 0 at the end of the cycle. The timing checker flip-flop also operates with the system clock.

If the timing checker flip-flop latches a 1, this signifies that the system clock period is close to being less than the worst-case path delay, and the system decreases the clock frequency. Conversely, if the flip-flop latches a 0, the clock period is greater than the delay through the worst-case path of the real logic, and the system increases the clock frequency, improving performance.

The timing checker flip-flop output therefore provides the command signal for the system clock generator to increase or decrease the clock frequency. This signal controls the counting direction of the up-down counter. The digital-to-analog converter (DAC) converts the counter output to an analog voltage signal. This signal sets the clock frequency by controlling the voltage-controlled oscillator, and the VCO output becomes the system clock, completing the feedback loop.

Thus, the clock period will never be less than the delay through the tracking logic plus the safety margin delay. Since the system's real logic is as slow or slower than the tracking logic, no timing error will occur in the real logic, which ensures correct system operation.

THE PROTOTYPE

Figure 2 shows the Xilinx field-programmable gate array-based prototype of the TEAtime system. The FPGA contains the TEAtime logic and a test computer. This computer contains a simple 32-bit, five-stage pipelined CPU with forwarding, and the equivalent of a small single-cycle-access cache memory.

The test computer executed a small program continuously during the experiments. This included all typical program constructs such as assignment statements, forward- and backward-conditional branches, and subroutine calls and returns. The program also exercised the pipeline's forwarding paths. The test processor stores the program's results in the cache memory. After every program execution, the PC host controller checks the results for correctness, then resets the results to bogus values before the next execution.

The test computer system clock's nominal worst-case specified operating frequency is about 30 MHz. The unit could be expected to operate at this frequency under even worst-case conditions. This baseline clock frequency was determined by using CAD tools performing worst-case-condition simulations of the test computer executing the test program.

Basic operation and stabilization

The first experiment established TEAtime's basic operational soundness and stabilization properties. With the FPGA's supply voltage held constant, and the test computer operating at room temperature, power was applied to the system. The system clock frequency rose from 25 MHz, the VCO's lowest frequency, then stabilized at about 45 MHz, as Figure 3 shows.

The horizontal line at 30 MHz indicates the approximate baseline worst-case operating frequency. The top subplot shows the value of the control line to the counter driving the DAC. The control is set for constant increases until stabilization, when the control value oscillates between increasing and decreasing frequency. The clock period varies only slightly above and below the delay through the tracking and safety margin logic. From a throughput perspective, TEAtime increases performance by about 50 percent compared with the baseline system under typical operating conditions—exactly the desired results.

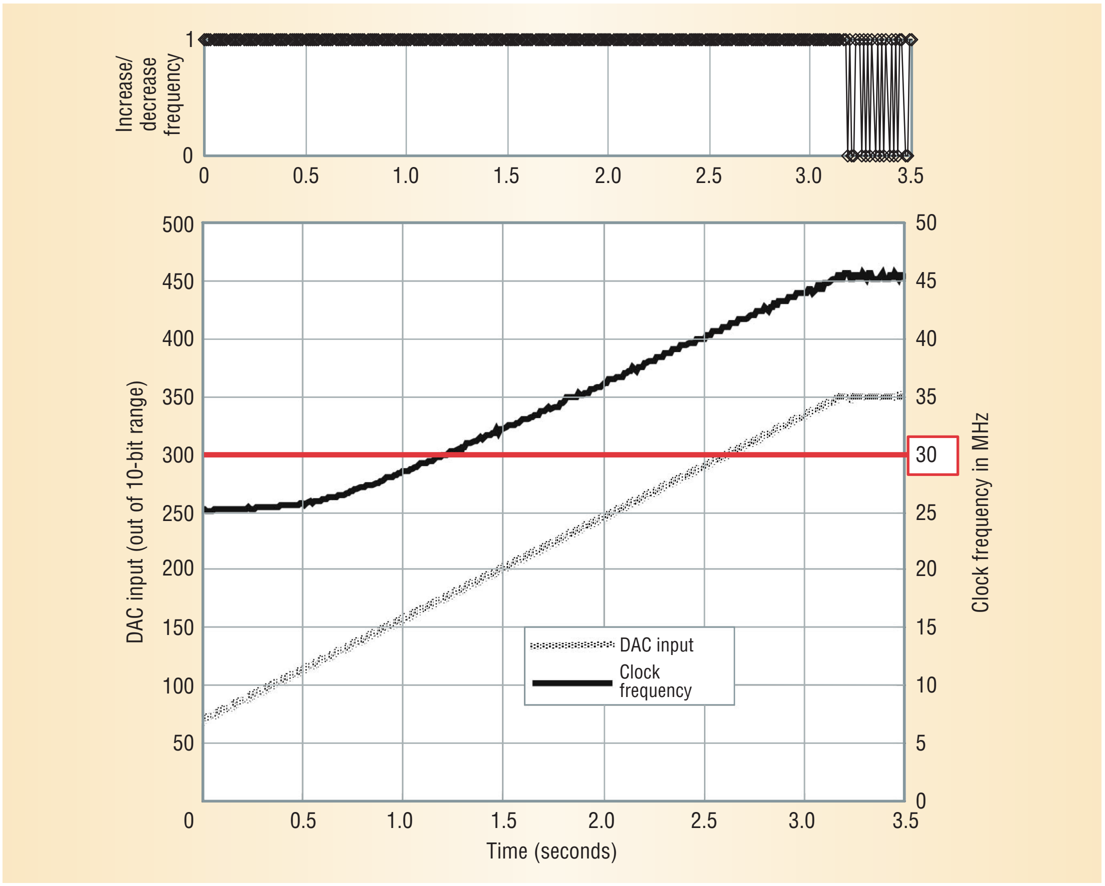


Figure 3. TEAtime's basic operation and stabilization. The clock frequency automatically increases to a high level, then stabilizes. The horizontal line at 30 MHz indicates the approximate baseline worst-case operating frequency. The control is set for constant increases until stabilization, when the value oscillates between increasing and decreasing frequency. The clock period varies only slightly above and below the delay through the tracking and safety margin logic.

TEAtime's adaptability

To examine TEAtime's adaptability to both FPGA temperature and supply-voltage (VCCint) changes, a thermocouple embedded in the center of the aluminum block, which is in turn thermally bonded to the top of the FPGA, measured the FPGA's case temperature—and hence the test computer's.

The FPGA has two supply voltages: one at 3.3 V for its I/O circuitry and one for the FPGA's internal logic, and hence for the test computer's logic as well. For correct operation, Xilinx specifies 2.5 V for VCCint, with an allowed variance from plus 5 percent to minus 10 percent. The green-colored region on the PC host's VCCint meter indicates this range. Only VCCint varied, with the test computer having no direct I/O connections to or from the FPGA chip. The FPGA never operated at above 3 V, which would have been a physically damaging VCCint value.

Figure 4 shows the detailed data for the combined temperature and supply-voltage variations. The supply voltage varied from 2.2 to 2.8 V, while the test computer's case temperature remained constant for each set of voltage data. Even though the operating frequencies varied widely—from 38 MHz to 49 MHz—TEAtime adapted to the existing operating and environmental conditions and always maximized the system clock frequency, within the safety margin delay. The test program executed correctly in all cases. In a perhaps extreme example of this adaptation to environmental conditions, a plastic-wrapped ice cube was placed directly on the FPGA's aluminum block, reducing the case temperature to 3° C. The system still adapted to the existing environmental conditions, functioning correctly and at a high clock frequency.

Tuning the system

The prior experiments used a large safety margin delay, but in this experiment, decreasing the delay and running the system to stabilization with typical conditions dramatically improved the operating frequency to 53 MHz. Time-related performance increased to 43 percent over the baseline, while throughput almost doubled.

Power reduction

To indirectly measure FPGA power use in the original untuned system, a current meter was added to the VCCint supply line, first testing VCCint set at the nominal 2.5 V, then with VCCint reduced to 2.2 V. Not unexpectedly, the power usage decreased by about 30 percent, but with only a 7.7 percent drop in time-related performance. This suggests a future application in which the operating system or application program could dynamically set a power budget. The hardware would then adjust VCC to use that power, and the TEAtime hardware would adapt the system to obtain the best performance under those conditions.

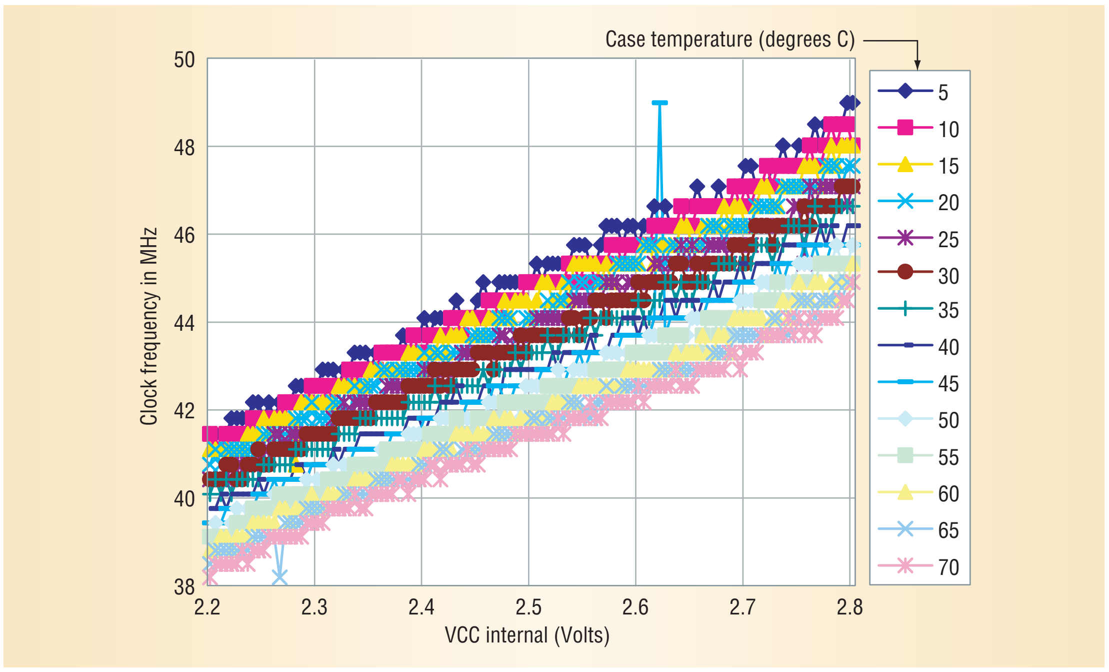


Figure 4. System clock frequency as a function of both supply voltage (VCCint) and case temperature. VCCint and temperature are both independent variables, while the clock frequency is a dependent variable. TEAtime successfully adapted to varying environmental and operating conditions—the two deviant points stem from frequency-measuring-circuit anomalies.

DESIGN CONSIDERATIONS

When implementing a TEAtime system, developers must consider several common digital system factors having unique TEAtime solutions. These factors include multiple worst-case paths and metastability—the unlikely state in which flip-flop output is unpredictable. There are also other factors that could be areas of concern but do not turn out to be, including the inductance-caused power-supply voltage-droop problem, created by large and rapid changes in power supply current—recall from physics: $V = L \times di/dt$.

Multiple worst-case paths

To design a target system for TEAtime use, and particularly to construct the tracking logic, the designer must determine the worst-case path in the system, then construct a one-bit-wide version of this logic and its wiring to mimic the worst-case delay. Next, the design must place the tracking logic as close to its corresponding real logic as possible—if not right in the middle of it—so that both components experience the same manufacturing, environmental, and operating conditions the real logic encounters.

This approach worked fine for the prototype, a very simple computer. However, complex microprocessor chips can have hundreds of worst-case paths, or worst-case paths within some small delta delay. These can all be different and exist in different operating environments because hot spots with varying temperatures and placements can occur on large chips.

To solve this problem, designers must construct the tracking logic for each possible worst-case path. Then, if any path indicates the clock frequency should be decreased, the DAC decrements. The DAC increments only if all tracking logic circuits indicate an increase.

This scenario raises yet another issue. The worst-case paths will likely be distributed throughout the chip. Given that it takes multiple clock cycles for signals to cross a chip, and more cycles are likely in the future with higher-speed clocks, the overall TEAtime control must be insensitive to such long delays.

The prototype only changed the operating frequency after every complete program execution, that is, after hundreds of cycles. The ideal TEAtime logic shown in Figure 1 actually had another flip-flop between the timing checker and the counter. This flip-flop was initially set, then cleared whenever the timing checker indicated a down signal. Over hundreds of cycles, if only one of the timing-checker samples indicated the clock frequency should be decreased, it was. This modification actually solves both the cross-chip delay problem—in which signals from the individual tracking logic circuits can use very low-speed paths—and the metastability issue.

Metastability

Flip-flop unpredictability occurs when both the data and clock inputs to a flip-flop change at or very close to the same time. In this case, the flip-flop can go into a metastable state in which its output is neither 0 nor 1: The output voltage level lies between the 0 and 1 thresholds. This means that circuits with inputs connected to the flip-flop will not only see a possibly incorrect value, but two different circuits could interpret this bogus value two different ways—one as 0, the other as 1—causing the system to malfunction.

Barring other stimulation, a metastable condition can last indefinitely. However, this is unlikely. Further, synchronous systems cannot avoid metastability completely—the best that developers can do is minimize the likelihood it will occur and, if it does, minimize its duration.

In TEAtime, the timing checker and the feedback loop's basic construction increase the likelihood of a metastable condition. When the system is stable, the input to the timing checker always changes right about the time the system clock's rising edge occurs. This is not a problem with the modified circuit, which examines the timing checker's output over hundreds of cycles, because of the very low probability that conditions will exist long enough to create a lasting metastable condition. The TEAtime control loop's integral changes in clock frequency and the slight changes in clock timing—known as clock jitter, which is inherent in all synchronous systems—further reduce the likelihood of metastability.

di/dt and other adverse conditions

On large microprocessors, such as Intel's Itanium, millions of transistors can switch state simultaneously, leading to a big change in the power-supply current in a short time, also known as a large di/dt, or change (delta) in current per unit change in time. When combined with the inherent inductance of the power supply network both on and off chip, this results in large voltage spikes on the chip's power buses. These spikes are already as large as plus or minus 5 percent of the power supply voltage and could increase in future chip generations.[10] The tuned TEAtime prototype's current safety margin can handle the effects of even plus or minus 9 percent supply voltage spikes. Such spikes are an issue with any large synchronous system, not just TEAtime. TEAtime designers can handle this and other system reliability constraints by increasing the tracking logic's safety margin delay. This can be done either at design time or at runtime, the latter with or without software assistance.

Many synchronous systems today have multiple clock domains with different unsynchronized frequencies, such as Intel-style PCs with unsynchronized CPU and PCI clocks. Therefore, TEAtime's varying clock should not be an issue in production systems; it can be handled by existing design techniques.

Developers can now take advantage of typical operating conditions and improve synchronous-digital-system performance dramatically. Such adaptable systems are also excellent candidates for mobile and military applications, in which digital systems undergo exposure to extreme environmental conditions. ■

References

1. M. Olivieri, A. Trifiletti, and A. De Gloria, "A Low-Power Microcontroller with On-Chip Self-Tuning Digital Clock Generator for Variable-Load Applications," *Proc. 1999 Int'l Conf. Computer Design*, IEEE CS Press, 1999, pp. 476-481.
2. A.K. Uht, "Achieving Typical Delays in Synchronous Systems via Timing Error Toleration," tech. report 032000-0100, Dept. Electrical and Computer Eng., Univ. of Rhode Island, 10 Mar. 2000; www.ele.uri.edu/~uht.
3. D. Ernst et al., "Razor: A Low-Power Pipeline Based on Circuit-Level Timing Speculation," *Proc. 2003 Int'l Symp. Microarchitecture*, IEEE CS Press, 2003, pp. 7-18.
4. A. Merchant et al., "Analysis of a Control Mechanism for a Variable Speed Processor," *IEEE Trans. Computers*, vol. 45, no. 7, 1996, pp. 968-976.
5. AMD Staff, "AMD PowerNow! Technology Brief," Advanced Micro Devices, Inc., 2002; www.amd.com/us-en/assets/content_type/DownloadableAssets/Power_Now2.pdf.
6. Intel Staff, "Intel Low-Power Technologies," Intel Corp., 2003; www.intel.com/ebusiness/products/related_mobile/wp021601_sum.htm?iid=ipp_battery+press_lowpow.
7. A.E. Sjogren and C.J. Myers, "Interfacing Synchronous and Asynchronous Modules within a High-Speed Pipeline," *Proc. 17th Conf. Advanced Research in VLSI* (ARVLSI 97), IEEE CS Press, 1997, pp. 47-61.
8. T. D. Burd et al., "A Dynamic Voltage Scaled Microprocessor System," *IEEE J. Solid-State Circuits*, vol. 35, no. 11, 2000, pp. 1571-1580.
9. A.J. Martin, "Design of an Asynchronous Microprocessor," tech. report CS-TR-89-02, Computer Science Dept., California Inst. of Technology, 1989.
10. E. Grochowski, D. Ayers, and V. Tiwari, "Microarchitectural di/dt Control," *IEEE Design & Test of Computers*, May-June 2003, pp. 40-47.

Augustus K. Uht *is a research professor in the Department of Electrical and Computer Engineering, University of Rhode Island. His research interests include computer architecture, especially the field of instruction-level parallelism, focusing on the study and implementation of resource flow computing and disjoint eager execution to realize the execution of tens of instructions per cycle. Uht received a PhD in electrical engineering from Carnegie Mellon University. Contact him at uht@ele.uri.edu.*