Foundations and Trends® in Electronic Design Automation
Vol. 11, No. 4 (2017) 362–461

DOI: 10.1561/1000000049

Error-Efficient Computing Systems

Phillip Stanley-Marbell
University of Cambridge
phillip.stanley-marbell@eng.cam.ac.uk

Martin Rinard
Massachusetts Institute of Technology
rinard@csail.mit.edu

Contents

Abstract

This survey explores the theory and practice of techniques to make computing systems faster or more energy-efficient by allowing them to make controlled errors. In the same way that systems which only use as much energy as necessary are referred to as being *energy-efficient*, you can think of the class of systems addressed by this survey as being *error-efficient*: They only prevent as many errors as they need to. The definition of what constitutes an error varies across the parts of a system. And the errors which are acceptable depend on the application at hand.

In computing systems, making errors, when behaving correctly would be too expensive, can conserve resources. The resources conserved may be time: By making some errors, systems may be faster. The resource may also be energy: A system may use less power from its batteries or from the electrical grid by only avoiding certain errors while tolerating benign errors that are associated with reduced power consumption. The resource in question may be an even more abstract quantity such as consistency of ordering of the outputs of a system.

This survey is for anyone interested in an end-to-end view of one set of techniques that address the theory and practice of making computing systems more efficient by trading errors for improved efficiency.

P. Stanley-Marbell and M. Rinard. *Error-Efficient Computing Systems*. Foundations and Trends® in Electronic Design Automation, vol. 11, no. 4, pp. 362–461, 2017.
DOI: 10.1561/1000000049.

1

Introduction

All software eventually works;
all hardware eventually fails.

— Clod Berrera.

This review explores the theory and practice of techniques to make computing systems faster or more energy-efficient by allowing them to make controlled errors. In the same way that systems which only use as much energy as necessary are referred to as being *energy-efficient*, you can think of the class of systems addressed by this review as being *error-efficient*: they only prevent as many errors as they need to.

There are numerous related fields relevant to understanding, designing, and evaluating systems which trade controlled errors for improved performance or energy efficiency. These related fields range from sub-areas of computer science, electrical engineering, and materials science, to applied mathematics and psychophysics (the study of perception). There are numerous techniques proposed by researchers in these diverse areas, with a vibrant and growing body of research results. This review focuses on two elements:

- **Fundamental concepts** that underpin any exploration of errors, time-efficiency (i.e., performance), and energy efficiency. These concepts have been developed over many decades in areas ranging from numerical analysis to the physics of semiconductor device behavior.

- **Practical hardware and software implementations** of error-efficient techniques to reduce energy usage in either practical engineering applications or experimental research platforms.

Throughout the review, we will focus specifically on the interplay between errors and the effects of errors as processed by human perception.

1.1 The Cost of Correctness

In computing systems, making errors when behaving correctly would be too expensive can conserve resources. The resources conserved in doing so may be *time*: by making some errors, they may be faster. The resource may also be *energy*: a system may use less power from its batteries or from the electrical grid by only avoiding certain errors while tolerating benign errors that are associated with reduced power consumption. The resource in question may be an even more abstract quantity such as consistency of ordering of the outputs of the system in question.

Which errors are acceptable depends on the application. The degree to which resources such as time or energy can be conserved likewise depends on the design of the computing system. And there are many different kinds of deviations in behavior which can be classified as “errors”. This Chapter provides an overview of the landscape of the applications, computing systems, and techniques that can be used to trade improved efficiency in exchange for occasional errors.

1.2 Historical Context

All hardware eventually fails. Reducing the likelihood of failure and the effects of failure comes at the cost of time, energy, or space. Making computing hardware more reliable was particularly important when the dominant applications of computing systems were in controlling weaponry and in financial applications. Today however, a large fraction of computing systems generate output solely for visual consumption.

Early computing systems based on vacuum tubes provided improvements in switching speed over their predecessors which were based on mechanical relays. They however also failed frequently: Failure rates in early vacuum-tube-based systems were as high as once every eight hours [von Neumann, 1956]. Because the possibility of intermittent and permanent failures has always been present in computing systems, the design of the basic elements of computation has evolved over time to inherently attempt to counteract the effects of failures.

One of the most fundamental techniques for dealing with the most basic source of failures (environmental noise) is to use digital logic, instead of performing computation directly in the analog signal domain. There is a rich body of work studying the tradeoffs between digital and analog computation, as well as on techniques to reduce both manufacture-time defects and runtime faults [Bushnell and Agrawal, 2000].

Redundancy, either in energy, space, or time, is a common approach used in digital logic to overcome the effects of noise. Error-correcting codes [Hamming, 1950] use redundancy in the representation of information to make it possible to detect and correct errors; the particular kinds and numbers of errors that can be detected and corrected depend on the amount of redundancy employed.

At a coarser grain, redundancy is also employed across complete computing systems, such as by replicating entire processors, complete servers, or even by replicating clusters and data centers. The challenges involved in such *fault-tolerant computing systems* are also the subject of a rich area of study [Avižienis et al., 2004].

Unlike traditional applications of computing systems, many modern applications of computation are in situations where the inputs to the system are from sources which are themselves noisy, unlike the inputs to a payroll application. Examples are the computations on sensor values in the many variants of health-tracking wearables. Similarly, the outputs of many applications are primarily for consumption via the human visual channel; an example is the rendering of images for a display. These applications could of course continue to be implemented with the level of redundancy used to guard against errors in traditional applications. Employing redundancy in space, time, and energy, independent of the needs of individual applications would likely have continued to be the way all computing systems are built. However, as the amount of energy used in a single logic operation reduced over time due to semiconductor process technology improvements, the overhead of the redundancy has become significant.

In those applications which do not require the same extremely low levels of errors, it is therefore now interesting to design systems which can trade errors for efficiency. And it is possible to go even further, to induce controlled amounts of errors if doing so would enable simpler, faster, cheaper, or more energy-efficient computing systems.

1.3 Why Precision Matters in Many Numerical Computations

There are many important computations whose implementations require careful attention to numerical stability, however few implementors of large-scale scientific computations have deep knowledge of numerical analysis. In the absence of such expertise, an alternative is to employ greater numerical precision [Bailey, 2005]. Because there are few automated techniques for transforming applications to improve their numerical stability [Panchekha et al., 2015], high-precision computations will continue to be important for a large class of applications. One example of a system where higher precision was used as an expedient solution to numerical instability is illustrated in the work of He and Ding [2001], who showed how problems with the reproducibility of climate-modeling applications could be eliminated by switching to using 128-bit floating-point arithmetic. A central theme throughout this review is that the types and magnitudes of errors permissible in an application must always be considered in the context of the tradeoff between errors and resource usage: a technique should permit only as many errors as an application and context can tolerate. Techniques should weigh permitted errors against the improvement in resource usage obtained from permitting errors. One way to achieve this in numerical simulations is to use multiple levels of precision across the phases of computations.

One cause of numerical instability in the presence of errors is that most general-purpose computations have great *arithmetic depth* [von Neumann and Kurzweil, 2012]. Small errors may therefore get amplified across the steps of a computation.

1.4 Why Some Applications Can Tolerate Errors

Despite the fact that many applications *cannot* tolerate any errors in their computations, there are also many applications which can. Typically, the applications that can tolerate errors are those that either:

1. **Operate on noisy inputs** (e.g., readings from sensors).

2. **Have computation outputs requiring limited precision**, e.g., because they are consumed primarily by human vision.

3. **Employ iterative or self-policing algorithms**. Examples of such algorithms are iterative methods where the computation will still produce the correct output in the presence of errors, provided that the computation makes progress in the right direction (on average) during each iteration.

4. **Do not have data-dependent control-flow**.

1.5 Examples of Improving Efficiency by Permitting Errors

Because displays account for a large fraction of the power dissipation in popular computing platforms such as mobile phones and wearable devices, trading errors for reduced resource usage in displays is an interesting prospect. Organic light-emitting diode (OLED) displays present an interesting opportunity for trading errors for efficiency: Unlike traditional LCD displays, their power dissipation varies significantly as a function of the content displayed. It is therefore possible to purposefully introduce errors into displayed images to reduce the display's power consumption. The earliest examples of such approaches were originated by Dong et al. [2009a] and Dong et al. [2009b], who developed several of the first techniques for trading display power for visual fidelity in OLED displays. Recent research has developed more efficient techniques as well as new approaches that analyze and transform both the color and shape content of the rendered images to save power.

Figure 1.1 shows two variants of the same image, which differ in power dissipation by over 40% when displayed on a representative commercial OLED display panel. The image and corresponding shape and color transformations to reduce power dissipation on displays that behave similar to OLEDs were generated using the Crayon system [Stanley-Marbell et al.,

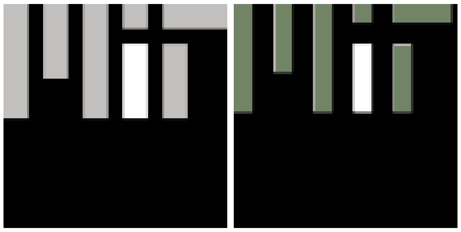


Figure 1.1: The image on the right dissipates more than 40% lower power than the one on the left when shown on OLED displays.

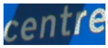
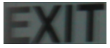
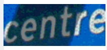
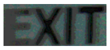
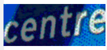
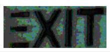
Tolerable Deviation	Image A	OCR Text	Transition Reduction	Image B	OCR Text	Transition Reduction
0%		“centre”	0%↓		“EXIT”	0%↓
10%		“centre”	66%↓		“LTXIT”	61%↓
20%		“centre”	73%↓		“”	73%↓

Figure 1.2: Encoding values so that they dissipate less power when transmitted can lead to significant power reductions before they begin to affect optical character recognition algorithms. This is despite the fact that the encoded images look very different to the human eye.

2016]. The difference between the original image and the modified one is that areas of the gray regions in the latter are reduced by 25% and the colors have been modified slightly. Chapter 4 explores techniques for exploiting tolerance in outputs in more depth.

Not all systems have displays however. In the increasingly important domain of embedded sensor-driven systems, because the power dissipated in the digital logic components has continued to drop over the years, a significant fraction of the system’s energy usage can result from the activation of sensors and the retrieval of data from them over their electrical communication interfaces.

Figure 1.2 shows how techniques that reduce the energy cost of transmissions by lossy encoding of the data can enable significant reductions in the energy required for transmitting the data. However, when the algorithms consuming the encoded data can tolerate the types of errors introduced by the encoding, they lead to minimal application-level errors, even though the perceived visual distortion may seem significant to the human eye.

Even though tolerating errors in the inputs and output communication of algorithms can be exposed in the syntax of programming languages [Stanley-Marbell and Marculescu, 2006], tolerating errors in the steps of algorithms is much more involved when compared to tolerating errors in the data algorithms process or errors in their outputs. Approaches to tackling this challenge range from annotating individual variables in algorithms as being ones that can tolerate errors (or not) [Sampson et al., 2011], annotating variables corresponding to the outputs of functions to specify which ones are permitted to incur errors [Misailovic et al., 2014], and using program analysis techniques to provide guarantees about the effects of errors as they propagate through the algorithm [Carbin et al., 2013].

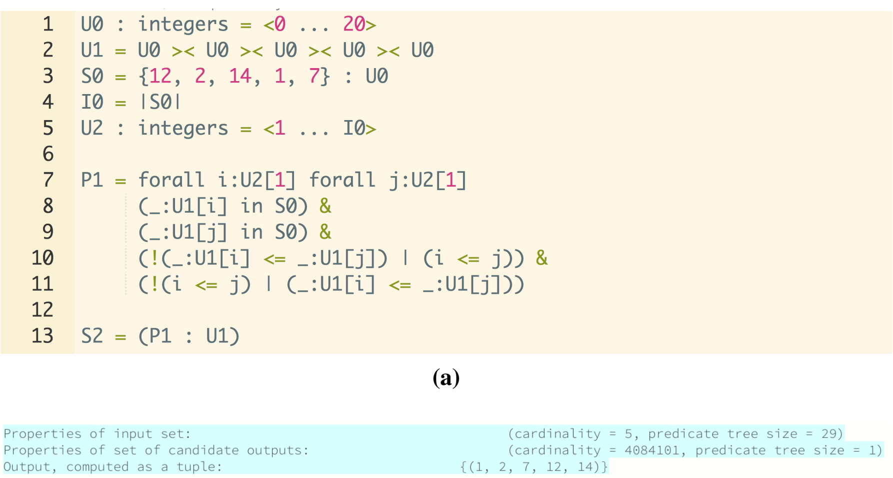
An alternative to providing specifications of the tolerable input or output error is to specify how much error is acceptable in the *relation between inputs and outputs*. Figure 1.3 illustrates the formal specification of the computation task of partial sorting, along with an example of an input-output pair that conforms to this computation behavior. This problem of obtaining a partial sort occurs in real applications: Partial sorting accounts for over 24% of the execution time of one popular discrete-event simulator [Jongerius et al., 2014]. One exciting open area of research is to synthesize algorithms (or hardware) that conform to such computation specifications and that permit some degree of error in the relation between their inputs and outputs.

1.6 Fundamental Physical Limits, Energy, and Noise

Computing systems are designed to avoid errors at all levels[1], from copying data from registers to their transmission to other systems or different processors. They prevent errors for all applications and, as a result, require error-correcting coding techniques at all levels; this introduces overheads that are unnecessary in some cases.

Because the traditional mechanisms for improving the density and power consumption of computing systems are reaching fundamental physical limits [Bennett and Landauer, 1985], there has been an increased interest in recent years to develop techniques to explore trading correctness for some tangible improvement in a system, such as improved speed or improved energy efficiency. Figure 1.4(a) shows the reduction in the energy required per bit of

[1] Within the limit of economic and performance constraints



```
U0 : integers = <0 ... 20>
U1 = U0 >< U0 >< U0 >< U0 >< U0
S0 = {12, 2, 14, 1, 7} : U0
I0 = |S0|
U2 : integers = <1 ... I0>

P1 = forall i:U2[1] forall j:U2[1]
     (_:U1[i] in S0) &
     (_:U1[j] in S0) &
     (!(_:U1[i] <= _:U1[j]) | (i <= j)) &
     (!(i <= j) | (_:U1[i] <= _:U1[j]))

S2 = (P1 : U1)
```

(a)

```
Properties of input set:                                    (cardinality = 5, predicate tree size = 29)
Properties of set of candidate outputs:                     (cardinality = 4084101, predicate tree size = 1)
Output, computed as a tuple:                          {(1, 2, 7, 12, 14)}
```

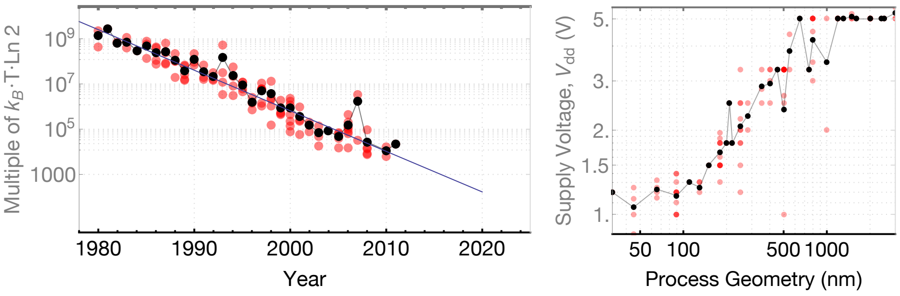
(b)

Figure 1.3: Computation specification (a) for the computation that sorts a sequence of integers, expressed in the Sal low-level computation specification language Stanley-Marbell [2010] and its output (b).

information processing, over several decades. Because the diminishing opportunities to reduce power consumption of computing systems is largely due to power delivery and cooling limitations, these challenges are unlikely to be easily resolved in the near future[2], making the exploration of error-efficient systems ever more important in the future.

The underlying physical phenomenon permitting such energy versus correctness tradeoffs is well understood: For a device technology to be useful in constructing computational systems in which logic devices are linked together by non-ideal conductors, it must exhibit the property of *gain* (amplification) [Keyes, 1985]. This amplification requires an input energy source and the extent to which amplification occurs affects the likelihood of errors due to noise. If some amount of noise is tolerable, its presence can be traded for energy efficiency or performance.

[2] Supply voltage scaling across technology nodes has ceased, as Figure 1.4(b) shows



(a) **(b)**

Figure 1.4: (a) The energy per logic transition in traditional circuit techniques is approaching the fundamental thermodynamic limit of $kT \ln 2$ Joules per bit of information (i.e., an ordinate value of 1 in (a) by ~2030). (b) One reason why energy usage in traditional CMOS logic is no longer scaling down, is that it is no longer feasible to decrease supply voltages. In both plots, the red points are published design data and the black points are the averages at a given abscissa [Stanley-Marbell et al., 2011].

1.7 Hardware and Software Systems That Exploit Errors

Techniques to improve system dependability have traditionally taken the approach of hiding (masking) faults in the hardware data-path and control-flow with spatial and temporal redundancy. Such an approach is desirable when there must be no change of system behavior in the presence of faults, except, perhaps, for a change in performance.

Applications of computing systems such as signal processing (in desktops and workstations), and sensor-driven applications (in embedded systems) often drive outputs that are only directly perceived by humans (e.g., the outputs of audio and video processing), or have inputs that are taken from noisy analog sources (e.g., in sensor network applications). In such applications, programs can often tolerate some amount of “going-wrong”. In particular, small deviations in values may be tolerable, and this is already exploited by some lossy compression algorithms for images (e.g., JPEG [Wallace, 1991]), audio, and video.

In many emerging applications of the recent decade, however, computing is moving from the sole purview of commercial business transaction management to more personal and pervasive applications such as embedded sensing and entertainment. In some of these new applications, such as embedded automotive control, there are still stringent requirements on correctness of machine state and computation. However, in many new applications, the need to maintain perfect error-free computation no longer exists.

As a result of these changes in applications of computing, a number of parallel research efforts have begun in recent years to explore ways to reduce the restrictions of perfect machine state. These efforts have ranged across:

- **Reducing the number of bits used to represent data values and datapaths**, either in storing those values or in synthesizing reduced-precision or reduced-accuracy logic in order to save energy (§ 1.7.1).
- **Explicitly exploiting human perception** to reduce resource usage (§ 1.7.2).
- **Circuits that perform logic operations on probability distributions of values**, rather than on unitary instance values (§ 1.7.3).
- **Hardware and software architectures for counteracting the effects of soft errors** (§ 1.7.4).
- **Architectures that assume applications can tolerate errors** in computation or timing, but have no contract with software on the permissible laxity (§ 1.7.5).
- **Programming languages and runtime systems that incorporate annotation of imprecision** in program state or operations, or exploit toleration of errors by applications (§ 1.7.6).
- **Investigation of application domains that can tolerate various forms of computation errors** or imprecision, in computation or state (§ 1.7.7).

These existing efforts have, however, mostly focused either only on adapting hardware independent of applications' requirements, or vice versa.

1.7.1 Reducing representation precision in values and datapaths

The earliest efforts at harnessing potential tolerance of imprecision, at the hardware level, involved reducing the number of bits used in both integer [Stephenson et al., 2000] and floating-point [Tong et al., 2000] representations. These efforts were not based on explicit information exposed by, or extracted from programs, but rather, on the assumption that signal-processing applications inherently deal with values obtained from noisy real-world measurements, and that real-number representations in computers are inherently approximations. Techniques that reduce the bit-level precision of arithmetic, and those that expose notions of incorrectness at the language level must contend with issues of numerical analysis. Kulisch [2008] provides a thorough background on the interaction between numerics of computation and the architectures that facilitate computing. In reducing the number of bits however, while the precision or dynamic range (or both) are reduced, computation proceeds deterministically and independent of the properties (value distributions) due to the applications it executes.

An alternative approach to simply providing reduced precision independent of application properties, is to synthesize logic circuits based on the distributions of values and the tolerance to reduced accuracy of specific applications, as investigated by Lingamneni et al. [2013]

1.7.2 Explicitly exploiting human perception

When the results of computation are consumed by the human aural or visual system, variations in accuracy, precision, or reliability may not always be perceptible. Such variations can be exploited directly in the generation of audio or display of results, for lower-energy, faster, or cheaper output devices (e.g., displays). For example, for displays, a few research efforts have investigated exploiting the variability in human sensitivity across the color spectrum. This phenomenon has been exploited to reduce power dissipation in OLED displays [Dong et al., 2009a, Zhao et al., 2013, Shin et al., 2011, Dong and Zhong, 2011, Harter et al., 2004, Li et al., 2014, Tan et al., 2013] as well as in those traditional LCDs that have coarse-grained controllable backlighting [Chuang et al., 2009]. Even when the results are consumed by non-human entities such as control systems, some amount of tolerance to imprecision, inaccuracy, and unreliability may still exist.

The interfaces for surfacing perceptual signals, such as displays and audio, contribute an increasing fraction of system energy usage in wearable and mobile systems. Because the phenomena underlying their operation (e.g., photon generation, mechanical displacement) are less amenable to improvements in transistor properties than computation is, their relative importance will likely grow in the future. Chapter 4 explores these concepts and implementations in more detail.

1.7.3 Probabilistic computation, probabilistic programming, and computing on probability distributions

In the traditional uses of probability in programming languages, the component which is probabilistic is the *behavior* of a computation, or a composition of concurrent processes [Stark and Smolka, 2000]. These approaches range from the *introduction of randomness into algorithms* [M. O. Rabin, 1976], the analysis of the behavior of randomized algorithms [Pnueli, 1983], and logics for probabilistic programs [Reif, 1980], to probabilistic parallel programs [Rao, 1994].

An alternative to the deterministic behavior of logic in hardware, whether of standard or of reduced precision, is to either employ randomness in the execution of hardware (to perform logic operations probabilistically [Palem, 2005, George et al., 2006]), or to consider the values of machine state due to executing applications, not as fixed instance values, but rather as probability distributions [Shanbhag et al., 2010, Vigoda et al., 2010, Vigoda, 2003]. The latter approach yields architectures that can be considered as forms of analog (as opposed to digital) computers.

1.7.4 Hardware and software architectures for counteracting the effects of soft errors

In the last decade, the observation that different applications (or classes thereof) may have differing tolerance to faults has been investigated [Wong and Horowitz, 2006], as have the possibility of applying different amounts of traditional software-based fault-tolerance techniques to different portions of an application [Reis et al., 2005a], as well as the influence of different hardware structures on the masking versus manifestation of faults as errors. These prior efforts, while recognizing the varying requirements for fault tolerance in applications and in hardware, have not attempted to tradeoff correctness for overheads.

There have been attempts to formalize the effects of soft-errors on the behavior of programs [Walker et al., 2006]. The model addressed in this recent work is one in which the goal is to attempt to nullify the effect of soft-errors (faults), by redundant computation.

The observation that different portions of programs or of hardware may require differing amounts of fault-protection has previously been applied to reduce the implementation overheads of hardware systems. This observation has been extended to phases of programs [Reis et al., 2005c] as well as to the design of error-resilient processor architectures and silicon implementations [Leem et al., 2010, Bau et al., 2007, Borodin et al., 2009, Rhod et al., 2007, Mehrara et al., 2007].

Several research efforts have explored adding architectural support for low-overhead detection and correction of the effects of soft errors, such as the *software anomaly treatment (SWAT)* system and its derivatives [Srinivasan et al., 2004], by determining the effect of soft errors in components of processor microarchitectures on application behavior [Li et al., 2005, 2008]. Purely-software-based approaches can also be used to trade correctness for speed or reduced resource usage. Two examples of such approaches include *loop perforation* [Sidiroglou-Douskos et al., 2011], and relaxing locking requirements in GPU kernels [Samadi et al., 2013].

1.7.5 "Better-than-worst-case" design and approximate hardware architectures

In probabilistic computing architectures (§ 1.7.3), non-determinism is used in a well-defined manner. This is in contrast to so-called better-than-worst-case hardware architectures [Austin et al., 2005, Wagner and Bertacco, 2007, Kahng et al., 2010], which aggressively bias system properties (e.g., power supply voltage) into regimes which may furnish significant energy savings, but increase the chance of failure. These architectures then use a variety of methods (e.g., shadow latches in the Razor system [Austin et al., 2004]) for ensuring infrequently-occurring erroneous state is not committed to final architectural state, or that critical data is not adversely affected (e.g., by reducing DRAM refresh rates, but only for non-critical data, in the Flicker system [Liu et al., 2009]).

Taking the idea of better-than-worst-case design further, are a class of architectures that argue that permitting occasional errors can reduce power consumption. When these platforms rely on applications and system software to deal appropriately with the errors that may result, we will refer to the platforms as *approximate hardware*. Examples of such approximate hardware range from processor architectures (or parts of processors such as ALUs) [Esmaeilzadeh et al., 2012b, Lingamneni et al., 2012], to complete accelerators [Esmaeilzadeh et al., 2012a, George et al., 2006, Sartori and Kumar, 2013], and to portions of the memory hierarchy [Sampson et al., 2013, Liu et al., 2009, Xu et al., 2004]. Techniques for approximation can be applied individually, or can be employed as part of a control system [Hoffmann, 2015] to ensure that a target energy reduction or accuracy constraint is satisfied.

As one example of these architectures, Truffle [Esmaeilzadeh et al., 2012a] defines an architecture in which individual operations (arithmetic instruction, memory accesses, etc.) may individually fail catastrophically with some probability, the rate at which they do so exhibiting a tradeoff with the amount of energy used. The manner in which this tradeoff is obtained is via the ability to set processor state and logic into a voltage-over-scaled (unreliable but energy-saving) state, with cycle-level granularity. Truffle relies on the programming language, compiler, and operating system to ensure that only individual instructions that can tolerate being in error are executed in the unreliable mode, and that unreliable state is appropriately quarantined from reliable state, with flow of data between reliable and unreliable computation obeying a well-defined set of constraints.

1.7.6 Programming languages and runtime systems

Program-level annotation provides an alternative to relegating to hardware all decisions about what machine state's accuracy can be traded for energy efficiency or performance. Language-level specification of tolerable imprecision has ranged from the specification of coarse regions of application code that can, in some broad sense, tolerate errors [Reis et al., 2005c, Walker et al., 2006, Baek and Chilimbi, 2010], memory locations that contain critical data [Pattabiraman et al., 2008], to the elision of loop iterations to tradeoff fidelity of computation results for energy efficiency or performance [Rinard et al., 2010, Rinard, 2006]. Program-level annotations of required precision such as the annotations provided by the EnerJ Java extension [Sampson et al., 2011] as well as tools to infer guarantees on correctness based on static program analysis [Carbin et al., 2013]. Detailed language-level facilities for specifying imprecision at the level of data types [Stanley-Marbell and

Marculescu, 2006] have also been developed, and extended to the declarative specification of the computation performed by a given subroutine, incorporating properties of imprecision [Stanley-Marbell, 2010].

1.7.7 Applications of "good-enough" computation in algorithms and software that are naturally resilient to errors

Given the aforementioned techniques for reduced precision arithmetic (§ 1.7.1), probabilistic computation (§ 1.7.3), hardware architectures and software techniques that take license with correctness (§ 1.7.4 and § 1.7.5), and language-level facilities for specifying how much incorrectness applications can tolerate (§ 1.7.6), a natural question is, which applications can best harness the possibilities afforded by these hardware and software innovations? Several proposals for potential application of such "good-enough" computation have been made in the research literature [Chakradhar and Raghunathan, 2010, Chippa et al., 2010, Breuer, 2010, 2005a, Meng et al., 2009, Chong and Ortega, 2007, Li and Yeung, 2007, Mohapatra et al., 2009, Breuer, 2005b, Salesin et al., 1989], however no consensus yet exists on a standard set of applications for evaluating proposed hardware and software techniques. Similarly, no commonly agreed-upon metrics exist for evaluating the degree to which behavior of benchmarks may deviate from correctness. Recent work has however taken an important step in this direction [Akturk et al., 2015].

One class of applications in which errors in computation are often tolerable is signal processing applications. This observation motivated some of the earliest work in trading correctness for performance and power from the work of Shanbhag on ANT [Hegde and Shanbhag, 1999, Shanbhag, 2002, Varatkar et al., 2009, Shanbhag et al., 2010], to silicon implementations of approximate signal processing from Amirtharajah and Chandrakasan [Amirtharajah and Chandrakasan, 2004] and Guo [Guo et al., 2006].

In addition to errors in values and control flow of computations, errors may occur in the timing of actions driven by computation, or in the latencies expected from computation. The term *imprecise computation* was coined in the nineties to denote real-time computing systems in which some deviation from temporal correctness was tolerable [Budin et al., 2004, Hull and Liu, 1993, Liu et al., 1991, Shih and Liu, 1995, Aydın et al., Liu et al., 1994, Kenny and Lin, 1991].

These efforts in computing systems and signal processing are of course predated by a large body of work in numerical analysis, uncertainty quantification (UQ) methods [Klir, 1994], tolerance graphs [Golumbic and Trenk, 2004], interval arithmetic [Hayes, 2003]), fuzzy logic and fuzzy set theory, approximation and randomized algorithms and, of course, existing work on in the broader field of fault-tolerant systems.

1.8 Outline of the Remainder of This Review

The present chapter provides a broad survey of the basic concepts explored in further detail throughout the review. It addresses the question of why error-efficient computing systems matter, and describes the context in which the material of the review is situated. It surveys the general state of the art in this area and positions the material of the review within it. Figure 1.5 summarizes the research referenced in this chapter. Chapter 2 (*Types of Errors and Randomization*) defines terminology, such as precision, accuracy, and reliability, which recur throughout the review and in any discussion of errors and of error efficiency. The definitions in Chapter 2 set the stage for the discussion of how errors affect efficiency in computing systems, in Chapter 3 (*Computation, Energy, and Noise*). Chapter 4 (*Tolerating Errors in Outputs*) addresses how many systems tolerate errors in their outputs. For example, any visual output that must be interpreted by a human may incur some amount of error before being perceptible. Chapter 5 (*Tolerating Errors in Inputs*) discusses the complementary problem of how many systems tolerate errors in their inputs. The review concludes in Chapter 6.

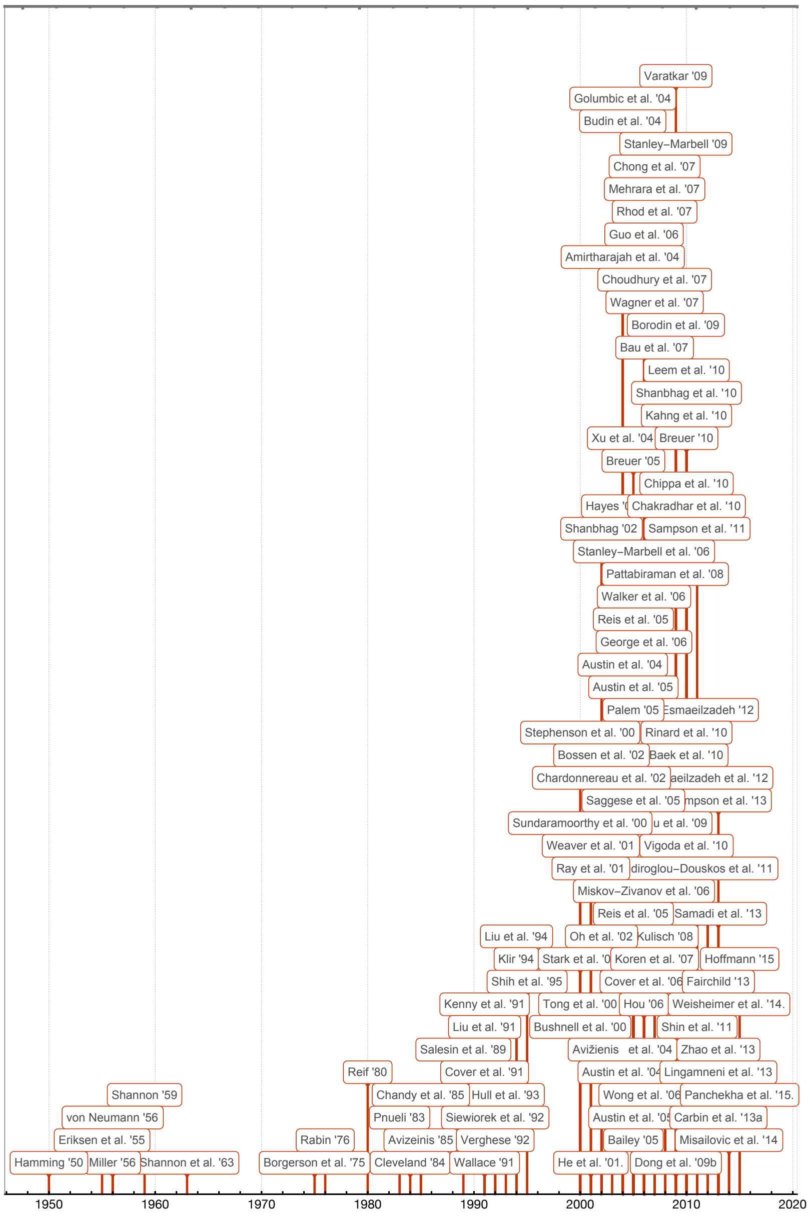


Figure 1.5: Timeline of referenced work in this chapter, listed by author.

2

Types of Errors and Randomization

> . . . even precision levels like $1 : 10^5$ are inadequate for a large part of important problems . . . The reasons for this surprising phenomenon are . . . that when they are broken down into their constituent elements, [the procedures] turn out to be very long . . . Now if there are large numbers of arithmetical operations, the errors occurring in each operation are superposed.
>
> — John von Neuman, *The Computer and the Brain*.

In common science and engineering usage, the term *accuracy* refers, broadly speaking, to distance from ground truth. Precision, on the other hand, refers to repeatability or spread around a mean. Accuracy and precision both imply that when things go wrong, the system still obtains an output and that this output differs from the correct output *to a quantifiable degree*. In contrast, *reliability* typically refers to the likelihood that a system component will fail. Tolerance of unreliability, or tolerance of faults which lead to failures, is the focus of the well-established discipline of *fault-tolerant and dependable computing systems*. Reliability and fault-tolerance were discussed in Chapter 1.

Tolerance of inaccuracy in numerical computations has been studied for well over a half century in the domain of *numerical analysis*. Tolerance of imprecision is well-studied, particularly in the context of imprecision in timing. There is a large body of work on *imprecise real-time systems*, dating back many decades.

In parallel with accuracy, precision, and reliability, the exploitation of randomness to improve algorithm performance has been explored in the area of *randomized algorithms* [Mitzenmacher and Upfal, 2005, Motwani and Raghavan, 2010]. Randomized algorithms employ randomness in the resolution of flow control in algorithms, such as by flipping a coin to determine which path on a branch to take. § 2.3 provides a concise introduction to randomized algorithms.

In contrast to this use of randomness in control flow, *stochastic computing* [Alaghi and Hayes, 2013] employs randomness in a different way. Rather than using randomness to choose which flow of control to follow as in the case of randomized algorithms, stochastic computing instead uses sources of entropy to generate distributions to represent different values to be used in arithmetic. § 2.4 introduces the concepts behind stochastic computing and explores how stochastic digital computing relates to analog-electrical computing.

Probabilistic programs [Goodman, 2013], like stochastic computers and randomized algorithms, also employ entropy in computation. However, in addition to employing values picked from some distribution in the steps of computation, they typically also infer or condition the distributions of values taken on by variables, based on values observed during computation. § 2.5 provides a brief overview of probabilistic programming.

2.1 Precision, Repeatability, Accuracy, and Reliability

The term *precision* usually refers to the resolution or spacing between values represented in a system. For example, a ruler with markings at every millimeter is more precise than a ruler with markings at each centimeter. Similarly, real-valued numbers can be represented in C programming language type `double` with finer spacing (precision) than they can be represented with type `float`. Precision is typically a property of a measurement instrument or computing system. In the context of a measuring device, precision can also be thought of as the *repeatability* or *spread* between values obtained in measuring an unchanging quantity.

The term *accuracy*, in contrast to precision, refers to the difference between a measured or computed value and its true or nominal value. For example, a measurement that reports the speed of light as $299792458\,m \cdot s^{-1}$ is accurate, while one that reports the speed of light as $299792459\,m{\cdot}s^{-1}$ is less accurate (but expressed in a representation that is as precise as the previously-stated value). All measurements of values in the real world have some degree of uncertainty due to systematic or random errors in measurement. Measurement values with high accuracy are those with low uncertainty. Measurement values with low accuracy are referred to as *approximate*.

The term *reliability*, in contrast to precision, repeatability, and accuracy, is typically used to refer to the behavior of a system. Reliability refers to the relative frequency with which a device fails or is otherwise unavailable for use, regardless of whether it is precise or accurate.

2.2 Accuracy of Models versus Precision of Computations

Accuracy is important when obtaining measurements of signals from the physical world. Once a measurement system has provided accurate measurements, higher precision in the data representation when storing or computing on the measured values may allow accurately-measured data to be used to obtain accurate results in data analyses. For example, a bar code scanner at a retail store must accurately determine the item being purchased. Once the scanning subsystem has accurately identified an item, subsequent computations such as charging a customers credit card must also occur accurately.

Not all measurement and computing systems require perfect accuracy however. There are many applications of data processing where the computing process into which measured data is fed is a model or algorithm that is itself an approximation of a poorly-understood physical process. Chapter 4 (tolerating errors in computing system inputs) and Chapter 5 (tolerating errors in computing system outputs) study two classes of systems where the users of computing systems and the algorithms consuming measurement data may tolerate varying degrees of inaccuracies in their inputs.

2.3 Randomized Algorithms: Making Randomized Decisions to Improve Algorithm Performance

Randomized algorithms are algorithms that make random decisions during their execution. These randomized decisions are based on *random* *sampling*—repeatedly choosing random values according to a specific probability distribution. An example of a randomized algorithm is a variant of Quicksort [Hoare, 1961] with a random pivot. Such randomization of control decisions (or of the algorithms inputs) may enable algorithms to deal with pathological inputs, by making all inputs lead to algorithm behavior that is characteristic of typical inputs. Because some NP-hard problems may be easy to solve for typical inputs, using randomization to make all inputs look like average-case inputs is one important tool for tackling intractable computational problems. Randomized algorithms can be classified into two main groups. *Monte Carlo* algorithms may fail or may provide an incorrect answer. *Las Vegas* algorithms on the other hand always return the right answer, but may take a variable amount of time to do so.

When an algorithm makes random decisions, its performance can no longer be deterministic. Moreover, even deterministic algorithm behavior may vary with inputs. *Probabilistic analysis of algorithms* is a closely related topic that deals with estimating bounds on behavior of algorithms.

Even though they employ randomness, the use of randomness in randomized algorithms is at well-defined control-flow decision points. Randomness in randomized algorithms is not simply the introduction of random errors through an algorithm's control or data path. It is therefore incorrect to assume that randomized algorithms are inherently a good match for systems that make errors. The techniques from probabilistic analysis of algorithms may however still prove useful in analyzing properties of the behavior of algorithms executing on platforms which may incur random errors.

2.3.1 Analyzing randomized algorithms: Random variables characterize the actions of algorithms

Each instance where a randomized algorithm employs a random sample to influence a decision corresponds to an *event*. For example, in the case of randomized Quicksort, each event corresponds to a specific member of the input being chosen as the pivot. For example, we might say "the algorithm randomly chose the fourth element as the pivot in this iteration".

There are several pieces of terminology that are essential in discussing randomized algorithms and the probabilistic analysis of algorithms in the remainder of this chapter and in the research literature. The randomized actions of an algorithm can be represented formally with *random variables*. A random variable, X, is a function on the elements of the sample space of possible values, Ω. A random variable, X on a sample space Ω is a real-valued function on Ω; i.e., $X : \Omega \to \mathbb{R}$. Events correspond to a random variable, say, X, (uppercase) taking on a specific value, say, x (lowercase). The probability of a random variable X taking on the specific value x is written as $\Pr\{X = x\}$ or $f_X(x)$. An event consists of the random variable taking on a specific instance value.

2.3.2 Probabilistic analysis of algorithms

Probabilistic analyses of deterministic and randomized algorithms use random variables to characterize properties of algorithm behavior and allow us to answer questions about the behavior of algorithms that make randomized decisions. For example, let X be a random variable denoting the number of comparisons made by a randomized version of Quicksort that randomly chooses its pivot. Then, we can use probabilistic analyses to answer questions such as the expected number of comparisons, $E[X]$, and hence the expected running time.

2.4 Stochastic Digital and Analog Computing: Computing by Exploiting Explicitly-Random Inputs

Stochastic computing systems represent values with distributions whose parameters are a function of the values intended to be represented. For example, the number 4 might might be represented with a collection of 32 randomly-generated binary digits in which on average four of the 32 digits are 1s. Stochastic computers then use these distributions in computations, exploiting the property that certain operations which are complex when applied to values (e.g., multiplication) are simple when applied to distributions (e.g., a logical AND). Some approaches to stochastic computing eschew the use of random or pseudorandom bit sequences for deterministic sequences with a given ratio of zeros to ones [Alaghi and Hayes, 2013]. These approaches are in principle *deterministic unary arithmetic systems*, not stochastic.

The term *analog computing*, technically and historically, has two meanings: (1) computing with continuous values and (2) operation by analogy (simulation of one physical system or process using a second physical system or process). We will use the term *analog* to mean continuously-varying electrical, unless noted otherwise.

If the systems being modeled, either computationally or by direct analogy, have similar stochastic behavior, then the limited accuracy and inherent stochasticity of analog computing (e.g., variations with temperature and time) are not a problem. Limited accuracy is also not an issue if the systems being modeled have similar resolution, or if the system being emulated has lower resolution. But, it is sometimes desired in modern science to simulate systems which have inherently different stochastics from those of analog computing systems, or require very high precision.

2.4.1 Bridging the analog/digital gap

There are many signal processing steps that are efficiently implemented in the analog domain. These signal processing steps include filtering, mixing, and heterodyning. If we are to perform information processing in the analog domain, we must of course have the data we are to process in an analog representation, or digital data must explicitly be converted to an analog signal. Analog to digital conversion is however expensive. Sundström, Murmann, and Svensson [Sundström et al., 2009] present a first-principles derivation of the energy efficiency of several analog-to-digital converter (ADC) topologies and show the energy usage per conversion to be exponential in the number of bits of precision.

2.4.2 Information processing in analog electronics

Techniques for performing non-signal-processing computations on analog computing systems have historically focused on using current-mode analog electrical circuits to solve differential equations. These techniques have largely exploited the integrative nature of charge accumulation on capacitors, summing operational amplifiers, and current multipliers. In some applications, the stochastic behaviors of the electrical circuit have been exploited in addition to the macro-scale numeric operations, to model systems such as gene transcription, with which they may have similar energy dynamics [Mandal and Sarpeshkar, 2009].

When exploiting the similarity of the energy dynamics of subthreshold analog electrical circuits to model biochemical processes, the primary benefit over traditional Gillespie simulation [Gillespie, 1977, 1976] is that the computationally-expensive generation of exponentially-distributed random samples, which is needed in the Gillespie method, is handled naturally by the exponentially-distributed noise in analog electronics. Instead of building analog circuits specifically to model biochemical circuits, it should be possible to attain most of the benefits simply by accelerating the random variate generation with analog circuits. Indeed, this idea has been explored by Marr and Hasler [Marr and Hasler, 2014].

There are however challenges to using analog circuits to generate random variates. If the random variate generation is based on thermal noise, then the process will be temperature dependent. Marr and Hasler [Marr and Hasler, 2014] and Bai and Lin [Bai and Lin, 2015] both acknowledge this challenge, and Marr and Hasler propose abandoning the thermal-noise-based approach altogether, in favor of chaos circuits.

2.4.3 Precision and analog information processing

Analog electric computations have traditionally suffered from limited precision, since their precision was limited by the ability to build components (e.g., capacitors) with precise values. For components with a given value and precision, those values must not drift with temperature or other environmental conditions. Preventing such drift is not always easy or even possible to achieve. In contrast to analog electric computing systems, because Boolean-valued digital systems treat all values as being of one of only two possible levels, they only require as much control as needed to generate values of two distinguishable levels. Analog computing systems also traditionally suffered from an inability to limit the propagation of noise injected at individual steps in a chain. This inability to limit noise propagation is because, since analog electrical circuits are typically continuous-value systems, there is never a reference level to which noisy signals can be restored or thresholded.

The operation of analog circuits bears many similarities with stochastic digital computation [Alaghi and Hayes, 2013] and with unary arithmetic. This analogy has been alluded to in the work of Vigoda [Vigoda, 2003], but a much more forceful comparison can be made. Both stochastic digital computation and analog electrical computation achieve their computation through the combination of probability distributions. The number of samples needed to assure a stochastic operation of a given resolution is analogous to the representation length needed to represent unary values with a given precision. Similarly, the tradeoff between analog and digital value representations as a function of the required precision, is analogous to both the stochastic sample count and unary representation length.

2.5 Probabilistic Programming

The term *probabilistic programming* [Goodman et al., 2012, Gordon et al., 2014a,b] refers to a style of programs in which programs can generate random variates from one or more distributions and in which the programs can make control flow decisions based on the values taken by these random variates at runtime. Probabilistic programming languages thereby provide support for probabilistic inference using language-level constructs. Using properties of random variables and the laws of probability, probabilistic programs may perform inference to determine properties of unobserved random variables based on properties of observed random variables and the structure of the probabilistic program. Although the idiom of probabilistic programming can in principle be implemented in many existing programming languages, several special purpose probabilistic programming languages have been developed in recent years to make programming in this idiom more succinct. Figure 2.1 summarizes the research referenced in this chapter.

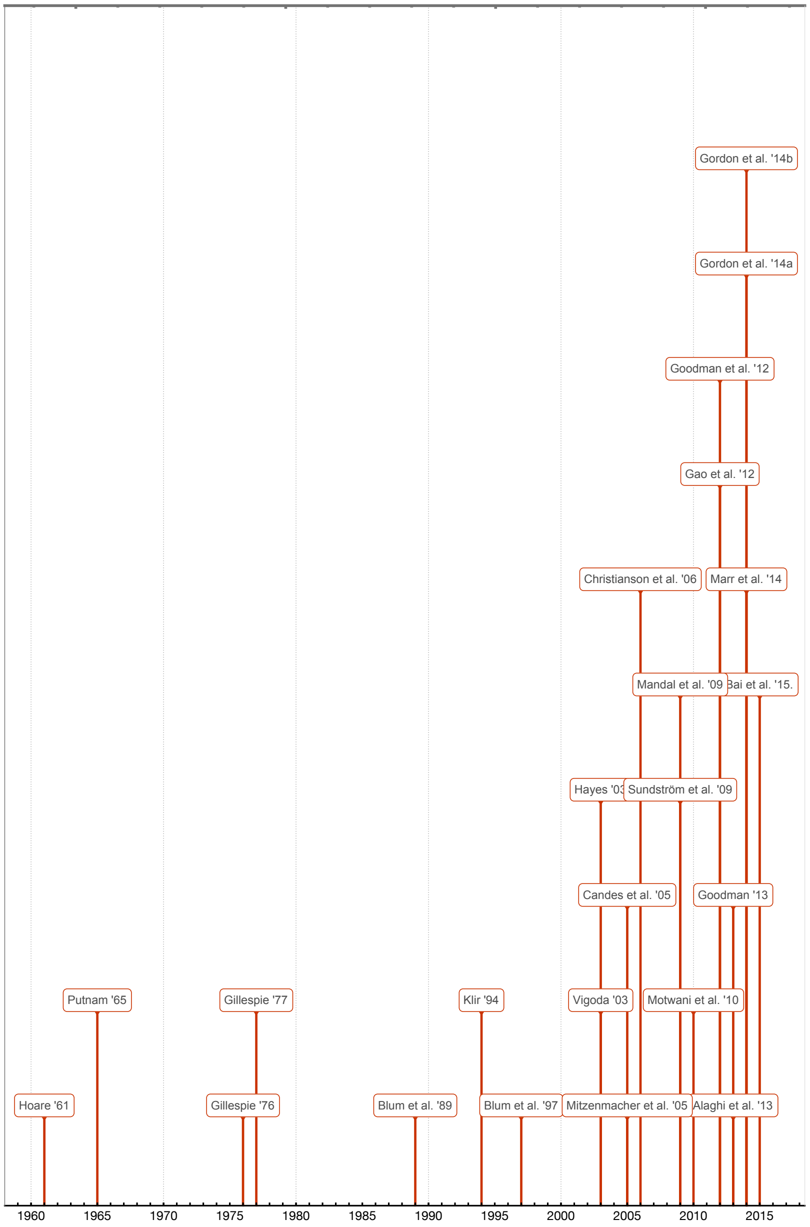


Figure 2.1: Timeline of referenced work in this chapter, listed by author.

3

Computation, Energy, and Noise

> These changes are indicated in terms of the dimensionless scaling factor κ . . . The power dissipation of each circuit is reduced by κ^2 due to the reduced voltage and current levels, so the power-delay product is improved by κ^3.
>
> — Dennard et al. [1974].

> Give a digital computer a problem in arithmetic, and it will grind away methodically, tirelessly, at gigahertz speed, until ultimately it produces the wrong answer.
>
> — Hayes [2003].

The real world is noisy. To achieve reliable information processing in the presence of this noise, the basic elements of computing systems use redundancy in signals. In binary-valued digital systems, this redundancy involves amplifying a signal to one of two logic levels after each logic stage. In analog systems on the other hand, redundancy is usually achieved by averaging signals over time. In both the digital and analog systems, the techniques for obtaining useful signals in the presence of noise effectively trade the depletion of one resource (energy, time) for the improvement of another (signal quality). This chapter explores the relationship between computation, energy, and noise. We begin in § 3.1 with an overview of how digital computing systems employ signal gain to guard against the accumulation of noise. We then explore the sources of noise in § 3.2 and review traditional fault-tolerant systems in § 3.3.

3.1 Devices Use Energy to Guard against Faults

Theis and Solomon [2010] give a cogent explanation of the lower limits on supply voltage necessary to counteract the effects of thermal noise. They start from the Johnson-Nyquist voltage noise, which follows a Gaussian distribution with standard deviation of voltage noise

$$V_n = \sqrt{\frac{k \cdot T}{C}}, \tag{3.1}$$

where k is Boltzmann's constant, T is temperature in Kelvin, and C is the load capacitance of a typical gate. They then take V_n as the minimum voltage needed to distinguish between two logic states. Thus, a logic voltage of m standard deviations will give a probability of reliable operation (logic value being greater than noise), given by the complementary error function, of

$$\frac{1}{2} \cdot \text{Erfc}\left(\frac{m}{\sqrt{2}}\right). \tag{3.2}$$

It is thus possible to tradeoff probability of logic error for supply voltage, and hence active power dissipation. Lower supply voltages, in the traditional CMOS bulk FET, comes with exponential increases in leakage current. There are however a number of promising new device technologies which are not subject to this exponential dependence of leakage current on supply voltage, which in bulk CMOS is constrained by the subthreshold slope of 60 mV per decade, such as tunnel field-effect transistors (TFETs) [Theis and Solomon, 2010].

3.2 Types and Sources of Noise and Faults

Design faults[1] are mistakes made in the specification or implementation of a system design. Manufacture-time defects (henceforth, *defects*) are the result

[1] These are usually referred to as design errors. For consistency, we will however reserve the term *error* for special use, as explained in this section. We refer to design-time "mistakes" as *design faults*.

of aberrations in the manufacturing process which contrary to the desire of a system's designers cause intermittent or permanent failures. Faults, or more elaborately, *lifetime-faults*, are those deviations from correct functionality, that occur during the lifetime of a system. They may be the result of aging-related processes, in which case their effects will be time-dependent and often irreversible. Faults may also be the result of intermittent external phenomena, such as electrical noise, high energy neutrons or α-particles. The term "soft-error" is usually used to refer to such intermittent or transient faults. In this review however, the term "error" is used in a more restricted sense.

Faults and defects may be *masked* by appropriate design or runtime actions, in which case the system will continue to function correctly in their presence. In a detailed gate-level simulation of an entire embedded microprocessor, Saggese *et al.* [Saggese and Vetteth, 2005] show that faults in data values in the register file, load-store unit and bus interface account for more than 50% of the faults that are not masked. We will refer to defects and faults which are *not* masked as *errors*, and to temporary errors as *soft errors*. Errors lead to perceptibly erroneous behavior in a system. A system may be able to tolerate errors, detecting them and continuing functioning. There are many ways to detect errors in hardware and in software. Some fault-tolerant embedded systems [Chardonnereau, Damien and Keulen, Raijmond and Nicolaidis, Michael and Dupont, Eric and Torki, Kholdoun and Faure, Fabien and Velazco, Raoul, 2002] and workstation-class systems [Bossen et al., 2002] implement detection circuits for errors in on-chip memory and logic.

The abstraction of the effects of soft errors employed in this chapter will be that of soft errors leading to *bit flips* or *inversion upsets*, where a high logic level (logic `1`) is incorrectly forced to a low logic level (logic `0`) or vice versa. Rather than causing a logic $\texttt{0} \rightarrow \texttt{1}$ transition or vice versa, a fault might cause a logic value to assume a state that is *ostensibly invalid* (i.e., neither `1` nor `0`). Such upsets will be referred to as *erasures* or *erasure upsets*.

Facilities such as *watchdog timers* may be employed to detect errors that manifest as system deadlocks or latency constraint violations, and communication erasures may be detected with techniques such as the use of message sequence numbers. The term *failure* will be used to refer to errors which lead to irrecoverable system demise.

3.2.1 Failure mechanisms and their sources

Soft errors have been observed in computing systems as far back as the late fifties, when they were first observed as intermittent failures of electronic equipment during nuclear bomb testing [Ziegler et al., 1996]. The mechanisms underlying their occurrence begun to be understood in the late seventies, and they have since been systematically studied from a variety of viewpoints. The sources of high-energy particles from device packaging material [May and Woods, 1979] and from cosmic rays, leading to a spectrum of energetic particles both on land and in avionics [Ziegler and Lanford, 1979, Taber and Normand, 1993], have been studied, as have the influence of such particles on integrated circuits [Ziegler and Lanford, 1981].

3.2.2 Semiconductor-process-related faults

Semiconductor process scaling occurs along multiple axes, all of which have some bearing on defects and faults. On one axis is the scaling of minimum feature sizes, e.g., minimum gate length and minimum half-pitch; this is usually the most publicized result of process scaling. Scaling the dimensions of device structures means that there are ever fewer atoms making up device features. For example, the *critical charge*, the number of electrons representing a logic value, is reduced across generations. As a result, lower-energy disturbances can cause a logic value at any point in a circuit, whether in combinational logic or in memory, to be changed from a `0` to a `1` and *vice versa*, or placed in an altogether invalid state.

Another source of defects and faults is variation in device properties, between devices (say, in the same cell library), between cell libraries, across an integrated circuit, within a wafer, across wafers that are part of the same batch (Si ingot), and across batches. For example, at the 65 nm technology node, approximately 100 atoms control the threshold voltage of a transistor, thus a variation in as little as one atom is significant — variations in dopant concentration have been a problem in semiconductor processes for more than two decades [Borkar et al., 2004].

At some technology nodes, feature scaling requires drastic changes to the implementation of devices in an integrated circuit. These changes are manifested in new technologies such as copper interconnects, low-κ dielectrics, and new device structures such as multi-gate or fin-fet structures. These new technologies introduce new defect and failure modes. For example, interconnect thickness variability is primarily a problem for copper interconnects and not for traditional aluminum interconnects. This particular problem is mitigated by planarization techniques such as chemical-mechanical polishing. These planarization techniques however impose new requirements on circuit topologies, such as requirements on continuity of material densities (e.g., polysilicon or metal) across a wafer.

Although we are primarily interested in intermittent faults, it is worth briefly mentioning some of the sources of permanent faults (in addition to manufacture-time defects). Sources of permanent logic upsets in integrated circuits include irreversible aging-related phenomena, such as *electromigration-induced failures* in interconnect, *negative bias temperature instability (NBTI)* and *hot-carrier injection (HCI)* related failures. Electromigration is a process that leads to the narrowing and eventual severing of metal traces in integrated circuits. NBTI and hot-carrier injection lead to degradation of transistor characteristics with age, and may thus also be the cause of failures.

3.2.3 Externally-induced intermittent faults

Temporary logic upsets might be the result of runtime disturbances in an integrated circuit, e.g., due to power supply noise, variation of device parameters with temperature, power supply droop, or ground bounce. These disturbances might in turn be due, for example, to dynamic system adaptation techniques, or to phenomena unrelated to the behavior of a system. Temporary faults have long been of concern in high-availability systems such as servers [Slegel et al., 1999, Horst et al., 1990].

The dominant source of intermittent faults in integrated circuits in recent decades has been cited as high-energy particles such as neutrons and α-particles [Baumann, 2005]. The α-particle flux, the number of particle strikes per m^2/s, varies with altitude (with a peak at approximately 60,000 feet), with time (varies with the 11 year solar cycle), with application domain (e.g., terrestrial, versus space applications), and also varies with latitude [Heidergott, 2005]. Some of the sources of temporary logic upsets are illustrated in Figure 3.1. The reactions (e.g., neutron capture, radiation emission) resulting

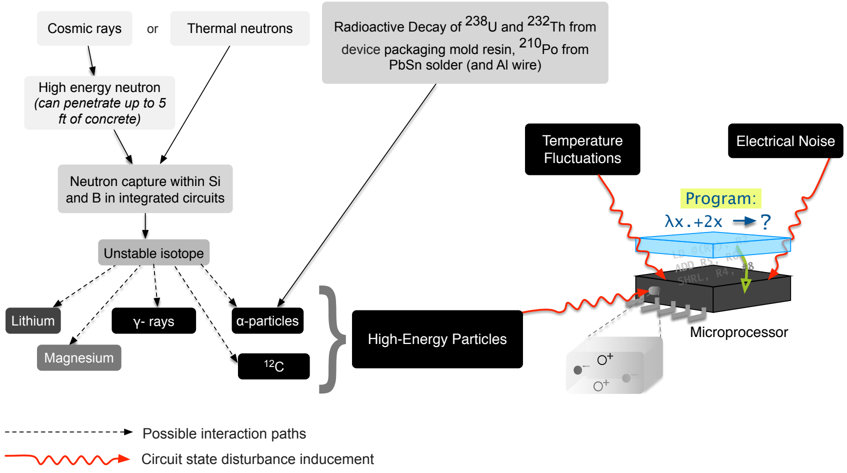

Cosmic rays
or
Thermal neutrons
Radioactive Decay of 238U and 232Th from device packaging mold resin, 210Po from PbSn solder (and Al wire)
High energy neutron (can penetrate up to 5 ft of concrete)
Neutron capture within Si and B in integrated circuits
Unstable isotope
Lithium
Magnesium
γ- rays
α-particles
12C
High-Energy Particles
Temperature Fluctuations
Electrical Noise
Program:
λx.+2x → ?
LD @(R4), R2
ADD R5, R0
SHRL, R4, #8
Microprocessor
O+
O+
Possible interaction paths
Circuit state disturbance inducement

Figure 3.1: Some sources of temporary logic upsets in hardware.

from interaction of high energy neutrons with integrated circuits are referred to as *spallation reactions*.

3.2.4 Characterizing fault rates

In characterizing the resilience of computing systems to soft errors, two approaches are generally employed — *accelerated testing* and *real-time testing*. Accelerated testing uses an artificial source of high-energy particles, such as radioactive material (e.g., ^{210}Po), or a neutron beam. In such accelerated tests, it is desirable for the spectrum of particle energies to match those in the foreseen deployment environment, e.g., at ground level for terrestrial applications. Real-time testing on the other hand deploys the units under test in naturally high-particle-flux environments, such as at high elevations. Such experiments have been carried out in places such as a laboratory in Jungfraujoch, Switzerland, (elevation 11,400 ft) [Nicolaidis and Chardonnereau, 2005] and an IBM testing facility in Leadville, Colorado, in the United States (elevation 10,152 ft) [Ziegler et al., 1996].

The changes in state caused by energetic particles are generally referred to as *single-event effects (SEEs)*. A temporary SEE that is not masked is termed a *soft-error*, and a permanent one a *hard fail*. One example of a hard fail is a *single-event latchup (SEL)*. It is possible that an SEE may lead to no erroneous change in state: For example, the transient pulse resulting from an SEE may be attenuated as it propagates through a combinational circuit, or it may occur outside the latching window in a sequential circuit. An SEE that *does* lead to a non-masked change in circuit state is usually referred to as a *single-event upset (SEU)*. The rate of occurrence of logic upsets (SEUs) is generally referred to as the *soft-error rate (SER)*. It is measured in units of *failures in time (FIT)*, where a soft-error rate of 1 FIT corresponds to one failure in a billion operation hours.

For memory technologies, the SER per bit usually decreases with decreasing feature size at a given operating voltage because the capture volume for interaction with high energy particles is smaller with smaller device area [Constantinescu, 2005]. Based on over 1000 tests performed on devices at different process technology nodes, Nicolaidis *et al.* [Nicolaidis and Chardonnereau, 2005] have shown that the FIT rate per megabyte decreases slightly across process nodes up to the 130 nm node, leveling off subsequently as manufacturing processes approached the 90 nm node.

To achieve constant-field scaling[2], semiconductor processes also scale supply voltages as they scale transistor dimensions [Dennard et al., 1974]. This lowering of supply voltages and the lower gate capacitances that result from scaling gate dimensions mean that there is less charge representing a logic value at smaller process geometries, possibly offsetting the smaller capture volume.

Overall, the SER *per integrated circuit die* for common applications like microprocessors increases due to increasing circuit complexity, with increasing number of transistors per die. The current FIT rate in memories is approximately 1000 FIT [Nicolaidis and Chardonnereau, 2005] to 2000 FIT [Jacquet, 2006] per megabit. In comparison, the failure rate due to NBTI and HCI is about 100 FIT [Jacquet, 2006].

3.3 Traditional Fault-Tolerant Systems

The construction of computing systems which continue to provide utility in the presence of faults [von Neumann, 1956] has been an active area of

[2] Keeping the electric field across the gate constant as the gate length is reduced between process technology generations.

research for many decades. With the advent of multi-processor systems in the 1960's, the idea of *gracefully-degrading systems* [Borgerson and Freitas, 1975] which tradeoff performance for utility in the presence of faults became of interest; the graceful degradation in these systems was graceful degradation with respect to their performance and not graceful degradation with respect to correctness (i.e., not what we might call adaptive error-efficient systems).

From the viewpoint of computing systems as information processors, the construction of fault-tolerant systems can be likened to the reliable transmission of information as considered in information theory [Shannon and Weaver, 1963]. The techniques employed in utilizing redundancy in providing reliable computation are analogous to the use of redundancy in information streams to enable forward error correction. The idea of trading off encoding overhead for the quality of the transmitted signal has been investigated in information theory, with the theoretical underpinnings of *rate distortion theory* [Berger, 1971], and with practical applications such as *unequal error protection* and *priority encoding transmission* [Albanese et al., 1996] taking advantage of semantic constraints on correctness of data streams. There have however been no equivalent efforts to trade off the correctness of computation for lower overheads in fault-tolerance. This could in part be seen as a result of the primary use of computers, prior to the present decade, in applications in which *any* form of (undetected) error was undesirable.

In contemporary digital computing systems, underlying physical processes (e.g., voltages) are usually treated as having only two states (logic `0` and logic `1`) at the level of individual *binary digits (bits)*. Across multiple bits in the internal representation of a system, collections of bits are typically required to retain their assigned value and to do so without incurring any anomalous deviations. To achieve this illusion, computing systems have, over the years, devised and employed a variety of techniques for identifying errors in collections of bits (e.g., *cyclic redundancy check (CRC)* codes), and for employing redundancy to enable the correction of errors (e.g., *error-correcting codes (ECC)*, and simpler arrangements of coarse-grained redundancy paired with majority voting). These techniques have been applied at all levels of computing system implementations, from registers and buses, to on-chip memories, data transmitted on interconnection networks, to data stored on a variety of media. All these techniques are important when absolutely no errant logic state should go uncorrected, or, at least, undetected, and is the subject of the important research areas of fault-tolerant systems [Koren and Krishna, 2007], coding theory [Cover and Thomas, 1991], and digital systems testing [Bushnell and Agrawal, 2000].

3.3.1 Dealing with faults in computation and communication

Because computing systems traditionally attempt to prevent the occurrence of errors, there have been several techniques developed to counteract faults in computation and communication systems. Approaches to counteract soft errors include circuit-level techniques such as the use of high-value polysilicon transistors in the feedback paths of static random-access memory (SRAM) cells, alternative SRAM cell topologies and inter-digitating the chip-level layout of the bits of different memory words to reduce the chances of multi-bit errors within a single machine word. Architectural and system-level techniques include the use of error correcting codes (ECC) for memory and array structures, and the use of redundancy (e.g., triple modular redundancy (TMR)) for entire functional blocks. There also exist software techniques aimed primarily at providing algorithmic and high-level program-module-based fault tolerance [Avizeinis, 1985].

Approaches for improving reliability of computation have traditionally been placed under four main classifications: fault avoidance; fault detection; masking redundancy; and dynamic redundancy [Siewiorek and Swarz, 1992]. *Fault avoidance* involves proactively designing systems, at the hardware or software level, that prevent the occurrence of faults. *Fault detection* techniques provide mechanisms for the incorrect status of hardware or software to be detected, a simple example being parity bits in memories. *Masking redundancy* approaches employ redundant hardware or software resources to mask the presence of faults, e.g., by taking a majority vote over redundantly performed computations, as in N-modular redundancy (NMR). Masking redundancy approaches implicitly employ a static organization of resources to mask the presence of faults. *Dynamic redundancy* techniques on the other hand take advantage of redundantly available hardware resources as they are needed and may tradeoff performance for reliability, e.g., by also using redundantly available devices for useful computation.

Techniques such as NMR can be seen as a form of channel coding in which N bits are employed to encode the value of each bit of an information source. As in the case of coding in communication systems, a particular encoding appropriate for a given system (in this case, a particular value of N) will provide the best tradeoff of redundancy overheads versus increased resilience to faults. Furthermore, the appropriate encoding will depend on the distribution of faults expected to be incurred.

3.3.2 Fault-tolerance beyond N-modular redundancy

All hardware and software systems are susceptible to failures. When the failure rates are acceptably small, the tradeoffs might be in favor of doing nothing to counteract their effects — this is the state of most consumer computing systems today. The tradeoffs change, when, either the sources of failures increase in intensity, or the susceptibility of hardware to already extant failure-inducing mechanisms is increased. Recent renewal of attention to the effects of failures in computing systems has been a result of the latter.

A large fraction of research into mechanisms for coping with failures attempts to nullify the effects of failures. From the perspective of failures in computing systems, one of the earliest directions was the study of the use of replication of computation to achieve fault-tolerance [von Neumann, 1956]. Hardware techniques in this area can be broadly classified as employing static organizations of redundancy (e.g., N-modular redundancy), standby-sparing systems (redundant hardware is swapped-in, on the occurrence of a failure), architectures which tradeoff performance for reliability (e.g., gracefully degrading systems [Borgerson and Freitas, 1975]), and architectures which employ dynamic organizations of redundancy through reconfiguration [Siewiorek and Swarz, 1992].

There exists a substantial body of work on software fault-tolerance, with methods such as checkpointing [Chandy and Lamport, 1985], N-version programming [Avizeinis, 1985], software-based fault-tolerance via redundant computations [Oh et al., 2002a], and redundant computations at the machine-instruction level [Reis et al., 2005a,c, Oh et al., 2002b]. For example, SWIFT [Reis et al., 2005a] uses the insertion of instructions at compile time to enable redundant computations within a single thread of execution. These redundant computations permit checking of addresses and data values before stores to memory, as well as the checking of control-flow. The overheads of software-only techniques can be reduced by employing hybrid hardware-software techniques [Reis et al., 2005b].

Similar to the evolution of software fault-tolerance from the level of software modules to the level of instructions, there have been efforts to enable fault-tolerance at the level of logic gates, within both combinational and sequential circuits. Recent efforts to improve the efficiency of hardware fault-tolerance include efforts to quantify the effect of faults at individual gates in a circuit on its primary outputs [Miskov-Zivanov and Marculescu, 2006]. Faults occurring within a processor microarchitecture may also be targeted by one of the many microarchitectural techniques for fault-tolerance [Ray et al., 2001, Weaver and Austin, 2001, Sundaramoorthy et al., 2000].

Several existing techniques for dealing with faults attempt to nullify their effects, by the repetition of computation or the duplication of data. These techniques attempt to achieve correct program behavior in the presence of failures as opposed to enabling the definition of a *reliability-tradeoff contract* between applications and the hardware that executes them. Rate distortion theory [Shannon, 1959, Shannon and Weaver, 1963] is a research area within information theory concerned with the tradeoff of encoding efficiency (*rate*) for deviation of encoded values (*distortion*). A *distortion function* or *distortion measure* specifies this distortion as a function of the original data and its encoded form. Examples of distortion functions include the *Hamming distortion function*, where the distortion is defined as the probability of error in the encoded data, and the *squared error distortion* [Cover and Thomas, 1991]. Distortion functions are specific to the domain to which they are applied. The deviations in sensor input and display output values that will be introduced in Chapter 4 and Chapter 5 are examples of integer distortion distances. Figure 3.2 summarizes the research referenced in this chapter.

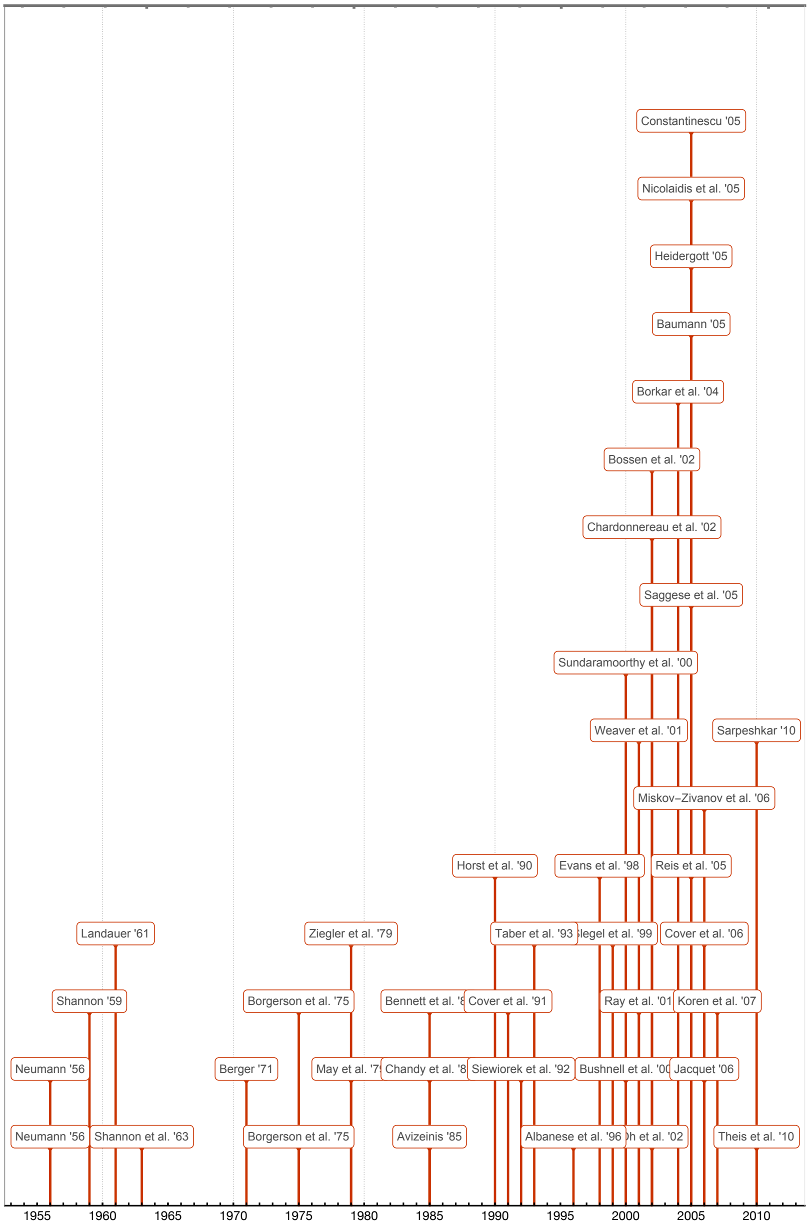


Figure 3.2: Timeline of referenced work in this chapter, listed by author.

4

Tolerating Errors in Outputs

> In this discussion the names of colors — “red”, “green”, “blue” and so on — will be reserved for the color sensation we have when we look at the world around us. In short, only our eyes can categorize the color of objects; spectrophotometers cannot. This point is not a trivial one because many people viewing some of our experiments for the first time will identify something as being red or green but will then ask, as if their eyes were being fooled. “What color is it really?” The answer is that the eye is not being fooled. It is functioning exactly as it must with involuntary reliability to see constant colors in a world illuminated by shifting and unpredictable fluxes of radiant energy.
>
> — Land [1977].

Consider a computing system that performs a task whose end result is only intended for display to a human observer. If the task can be altered such that it uses fewer resources, and if most human observers (or a specific one) cannot perceive any change in the end result, then the system can be made more efficient by either avoiding work or making errors as long as these errors are not perceptible, making the system error-efficient.

For computing tasks such as numeric solution of differential equations, computations which are part of a program for completing a tax return, or computing the cost of a sales order, obtaining the correct numeric result is almost always critical. Similarly, an alphanumeric display such as that dis-

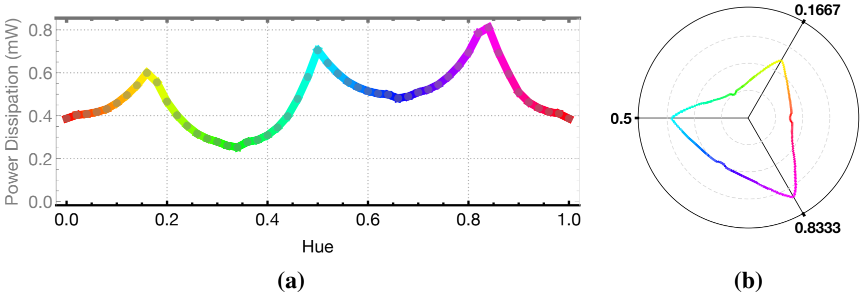


Figure 4.1: The power dissipation for a representative OLED panel as a function of a range of fully-saturated hues (the hue space wraps around), shown in linear coordinates (a) and in polar coordinates (b).

playing departures and arrivals at an airport or train station must display the exact information sent to it in order to be useful.

However not all computations and not all displayed information is of data that has a precise or quantitative nature. Computations whose results only feed into determining the color of a pixel in a temporary on-screen image have their requirements on accuracy bounded by the limits of human color perception (and attention spans). Similarly, because the display panel on a phone or smart watch can consume different amounts of power based only on the color content of images (Figure 4.1), changes to the color and shape content of images can affect the power dissipation of the display.

Because displays constitute a large fraction of the power dissipation of many modern mobile platforms it is possible to improve the battery life of many platforms if images could be adapted in ways that exploit display properties without being visually perceptible. In order to perform such changes however, we would need to have quantitative answers to several questions, such as:

- **How sensitive are users to changes in color?**
- **Are there colors that are indistinguishable to humans** but lead to significant changes in display power dissipation of some display types?
- **How sensitive are users to changes in shape?**

4.1 Human Perception of Color

Humans with normal color vision have three types of color sensitive cells (cones) in their retinas. Each of the three types of cone cells is sensitive to a broad range of wavelengths of light, but the different types each have peak sensitivities in the short-, medium- and long-wavelength portions of the visible spectrum. Because of the locations of these peak sensitivities, most people have most of their color sensitivity in the green portion of the visible spectrum. However, even though most of the sensitivity is to the portion of the visible spectrum close to green, most people are not necessarily able to easily distinguish between different wavelengths of light in the green portion of the visible spectrum as well as they do for other wavelengths.

For beams of light made up of a single wavelength (spectral colors), controlled colorimetric studies have been used to quantify how much a spectral color of a given wavelength must be changed to longer or shorter wavelengths for the change to be perceptible by humans. Several of these studies, such as studies by Wright and Pitt in 1934, and by Bedford and Wyszecki in 1958 [Wyszecki and Stiles, 2000] showed that the ability of human observers to differentiate spectral colors varies with wavelength. Changes in wavelength of spectral colors at wavelengths close to 450 nm (approximately, indigo) and 525 nm (approximately, green) were markedly more difficult for observers to differentiate than changes in other parts of the visible spectrum outside of the extremes. At 450 nm and 525 nm, wavelength changes of up to 4 nm were necessary for an observer to perceive a change in color, compared to wavelength changes of just 1 nm which were perceptible in the rest of the central portion of the visible spectrum.

The perception of spectral colors in color-matching experiments is different from the perceived color in images. The colors humans perceive in natural images, in contrast to the color sensations perceived in color-matching experiments, are not a direct result of the wavelengths of spectral light or combinations of the spectra that objects reflect or project. In a series of experiments in 1959 [Land, 1983, 1986, 1977, 1959b,a,c] and developed over twenty years, Edwin Land (the founder of the Polaroid Corporation) and his colleagues first showed how pairs of bands of wavelengths (e.g., red and white) when used to illuminate a pair of black-and-white negatives taken with red and green filters yield full color perceived images, even though the same illumination without the black-and-white negatives simply yielded a pink screen. Land and his colleagues also later demonstrated that when they illuminated differently color patches with controlled intensities of red, green, and blue light, so that the amount of red, green, and blue reflected off the patches and reaching the eye were identical, observers would still observe different colors. These and other investigations led them to the conclusion that perceived color was not just the result of the magnitudes of the constituent wavelengths of light reaching the observer, but rather the result of the relative spatial distributions: The visual system captures images corresponding to short- medium- and long-wavelength photoreceptors in the retina (the s-, m-, and l- cones) and does not simply average these images. Instead, it compares the relative intensities of nearby points in the scene captured by the normal human eye's three photoreceptors, to deduce what we perceive as color.

4.2 Quantifying Errors in Images

A way to quantify errors and efficiency is a prerequisite to meaningfully trading errors for efficiency. For images, there are several commonly-used ways to quantify errors. Let r be a reference image and let t be an image to be compared to r, with both r and t being w by h pixels in size. The most basic measure of difference between the original and modified images is the sum of squared errors (SSE) between the reference image r and the transformed image t, defined as

$$\mathrm{SSE} = \sum_{x=0}^{w-1} \sum_{y=0}^{h-1} \left(r_{x,y} - t_{x,y}\right)^2 . \tag{4.1}$$

The mean squared error (MSE) is defined as

$$\mathrm{MSE} = \frac{1}{w \cdot h} \mathrm{SSE}. \tag{4.2}$$

The signal to noise ratio (SNR) is the ratio of the sum of squared signal to the SSE

$$\mathrm{SNR} = 10 \log_{10} \left[\frac{\sum_{x=0}^{w-1} \sum_{y=0}^{h-1} r_{x,y}^2}{\mathrm{SSE}} \right] . \tag{4.3}$$

The peak signal to noise ratio (PSNR) takes the ratio of the maximum of all the squared signal (instead of the sum) to the mean squared error:

$$\mathrm{PSNR} = 10 \log_{10} \left[\frac{\max(r_{x,y}^2)}{\frac{1}{w \cdot h}\mathrm{SSE}} \right]. \tag{4.4}$$

Let μ_r and μ_t be the mean pixel values for two grayscale images r and t. Let σ_{rt} be the covariance between the images r and t and let σ_r and σ_t be the standard deviations of pixel values in r and t. Let C_1 and C_2 be constants derived from the dynamic range of the image representation format with b bits per pixel and equal to $(0.01 \cdot 2^b - 1)^2$ and $(0.03 \cdot 2^b - 1)^2$ respectively. Then, the structural similarity (SSIM) between the two images r and t is defined as:

$$\mathrm{SSIM}(r,t) = \frac{(2\mu_r\mu_t + C_1)(2\sigma_{rt} + C_2)}{(\mu_r^2 + \mu_t^2 + C_1)(\sigma_r^2 + \sigma_t^2 + C_2)}. \tag{4.5}$$

4.3 Display Technology

Displays account for up to 40% of the power usage in mobile devices such as phones, smart watches, and tablets. This makes it of great interest to manufacturers to develop ways to reduce display energy usage, as doing so would enable improvements in battery lifetime.

The most common type of display is the liquid crystal display or LCD. LCDs consist of a light source, called a backlight, which illuminates a collection of color filters at the individual pixels; whether each of these color filters receives light is controlled by a layer beneath the color filters, made up of (liquid) crystals, which can be made opaque or transparent, acting like miniature shutters, based on an electrical signal. Because the backlight is on all the time even though individual pixels might be blocking its light, the power dissipation of LCD displays is dominated by the backlight.

In contrast to LCDs, which have a single light source, organic light-emitting diode (OLED) displays typically have dedicated red, green, and blue light sources made from organic compounds, for each individual pixel. Because they are made up of a smaller number of layers (no need for the LCD layer or for a color filter), OLED displays can often be made thinner than LCDs.

The efficiencies of converting electrical current into light in red, green, and blue OLED pixels varies across the different organic compounds employed in making them. For example, on some OLED displays, a fully-bright blue pixel can dissipate almost twice as much power as a fully-bright green one. And over the lifetime of display, the efficiencies also degrade, with blue sub-pixels degrading in efficiency more quickly than red and green pixels. As a result, to achieve the same brightness as an OLED panel ages, blue pixels need to be driven with higher currents (and hence dissipate more power) to achieve the same level of brightness as blue pixels on a new OLED display. This challenge is referred to by researchers in the area of organic LEDs as "the blue problem". Because of these varied phenomena, the amount of power dissipated by an OLED display depends on the image being displayed.

Almost a decade ago, a number of researchers, starting with Lin Zhong at Rice University in Houston Texas, begun exploiting the color-dependence of OLED power dissipation to improve the efficiency of OLED displays by changing colors in ways in which an observer might find acceptable (such as by changing an operating system's or individual application's color theme), but for which the change leads to a reduction in display power usage.

4.4 Exploiting Perception for Display Energy Efficiency

The interfaces for surfacing perceptual signals, such as displays and audio, contribute an increasing fraction of system energy usage in wearable and mobile systems. Because the phenomena underlying their operation (e.g., photon generation, mechanical displacement) are less amenable to improvements in transistor properties than computation is, their relative importance will likely grow in the future.

A number of techniques for error efficiency, targeted primarily at legacy backlit LCDs, have been developed to reduce display power dissipation [Cheng et al., 2004, Ranganathan et al., 2006]. Prior work on trading image fidelity for power or performance can be classified broadly into six directions: Color transformation by convex optimization; color transformation in restricted applications such as web browsers and by color remapping; color transformation by electrical control of the display panel; selective dimming based on a user's visual focus; and image fidelity tradeoff analyses that employ perceptual user studies.

4.4.1 Color transformation by convex optimization

Reducing display power dissipation under an image-distortion constraint can be framed as an optimization problem. How efficiently one can solve this optimization problem, and the quality of the solution (global versus local minima) however depends on the power model and perceptual model that are used in the optimization problem formulation. Dong *et al.* employ a power model that has a count of parameters that is exponential in the display's color depth (e.g., $3 \cdot 2^{3 \cdot 8}$ or 48 million parameters for a 24-bit color display) [Dong and Zhong, 2011]. Their color optimization is thus expensive to do even offline, and not practical to do online (i.e., in real-time). Because the optimal solution of their optimization problem requires an operation count that is exponential in the number of pixels, the authors propose a greedy heuristic that is $O(n^3)$ for n-pixel images. As a result of their formulation, their color optimizations take many hours to complete [Dong and Zhong, 2011].

4.4.2 Application-specific transformations and color remapping

When color transformations are applied in restricted contexts such as in color schemes for infographics [Chuang et al., 2009, Wang et al., 2012] or in graphical user interface (GUI) color schemes [Dong et al., 2009b,a, Dong and Zhong, 2011], colors that are more power-expensive on OLED displays may be substituted for ones that lead to lower display power dissipation.

Another approach to exploiting the color dependence of OLED power dissipation, for improved energy efficiency is to directly modify individual applications such as games [Anand et al., 2011], web browsers [Dong and Zhong, 2011, Li et al., 2015], and web servers [Li et al., 2014] which provide content to devices with OLED displays. Application-specific tradeoff techniques have the disadvantage that modifications must be repeated for each new application. Application-specific techniques can also be complex: For example, the color-adaptive Chameleon web browser [Dong and Zhong, 2011] employs several techniques including designing color schemes for specific popular websites, inverting colors in web pages, and requiring users to explicitly select schemes. Chameleon requires color maps to be calculated offline, using an optimization method which the authors themselves describe as "compute-intensive", partly because it requires a large number of parameters: $3 \cdot 2^b$ for a display with b bits of color depth.

4.4.3 Power analysis and color transformation by electrical control

One way to control the power dissipation of displays is to manipulate their electrical interfaces, such as their backlights, display drivers, and so on, so that these components use less power. To understand the effect that such electrical control has on image quality however requires detailed electrical and perceptual characterization. As a result of this need for characterization, there have been several studies of LCD power dissipation as a function of backlight level and perceptual metrics [Chang et al., 2004, Choi et al., 2002, Cheng et al., 2004, 2006, Schuchhardt et al., 2015], as well as studies of OLED power dissipation as a function of color and luminance [Dong et al., 2009b, Shin et al., 2011, Mittal et al., 2012, Chen et al., 2013], and models to estimate OLED display power [Harter et al., 2004].

Once displays have been characterized, they can also be controlled to exploit properties observed in the characterization. The power versus perceptual quality tradeoff techniques that have been explored for LCDs include contrast scaling [Cheng et al., 2004, Pasricha et al., 2004], luminance scaling [Chang et al., 2004], and tone mapping [Iranli and Pedram, 2005, Anand et al., 2011]. A number of recent research efforts have attempted to apply similar hardware techniques to OLED displays. These techniques have included dynamic voltage scaling of OLED display driver amplifiers [Shin et al., 2011].

4.4.4 Selective area dimming

A number of research efforts selectively dim portions of an OLED display panel, based on heuristics of a user's focus of attention [Tan et al., 2013]. The techniques are obtrusive, and when they guess the user's focus of attention incorrectly, can render a device unusable. Other research efforts have used heuristics to guess which part of a display is occluded by a user's hand [Chen et al., 2014]; these latter techniques must, among other things, guess whether a user is left- or right-handed, how large their hands are, whether they are using a stylus, and so on.

4.4.5 User studies of image fidelity versus power tradeoffs

The value of techniques that trade perceptual quality for power or performance depend on how accurately the techniques quantify their effect on human perception. The two primary approaches to quantifying perceptual quality are quantitative metrics such as the structural similarity (SSIM) and peak signal to noise ration (PSNR), introduced in § 4.2, and human user studies. Several studies of the tradeoffs between image quality and power dissipation of displays have employed perceptual studies. These studies have however all involved only a small number of participants. For example, Harter *et al.* [Harter et al., 2004] employed a study size of 12 users in their analysis of the effects of selective display area dimming for OLED displays, while Tan *et al.* [Tan et al., 2013] employed 30 users in evaluating a similar technique. Li *et al.* [Li et al., 2014] conducted a perceptual study with 17 users to evaluate their color-adaptive server-side color transformations, while Dong *et al.* [Dong and Zhong, 2011] employ 20 participants to evaluate their color-adaptive web browser. Anand *et al.* [Anand et al., 2011] conducted a user study with 60 users to evaluate a display brightness and image tone mapping technique. All of these prior efforts provided valuable insight into the challenges and benefits of performing perceptual user studies.

4.5 Exploiting Perceptual Flexibility in End-To-End Systems

Crayon [Stanley-Marbell et al., 2016] is a system for exploiting perceptual flexibility to reduce display power dissipation. At the core of the Crayon system is an intermediate representation for representing bitmapped graphics (e.g., photographic images) and vector graphics (e.g., the drawing operations in a user interface). Across these representations, Crayon applies several techniques to reduce power dissipation in exchange for visual accuracy.

In bitmap images, a color transform is applied to each individual pixel independently in order to reduce power dissipation. The transform optimizes a simple convex function that trades visual fidelity (with a least-squares penalty on deviation from the target color) for reduced power dissipation; the penalty is based on an experimentally-measured model of the power dissipation for each color. The result of the optimization is a simple color transform that picks the closest color within a given tolerable deviation that minimizes energy dissipation.

For vector graphics, Crayon applies shape transformations that slightly enlarge or reduce the width of rectangles, lines, and polygons (or any closed path) to reduce power dissipation in exchange of imperceptible modifications to the geometry of the displayed shapes. Figure 4.2 summarizes the research referenced in this chapter.

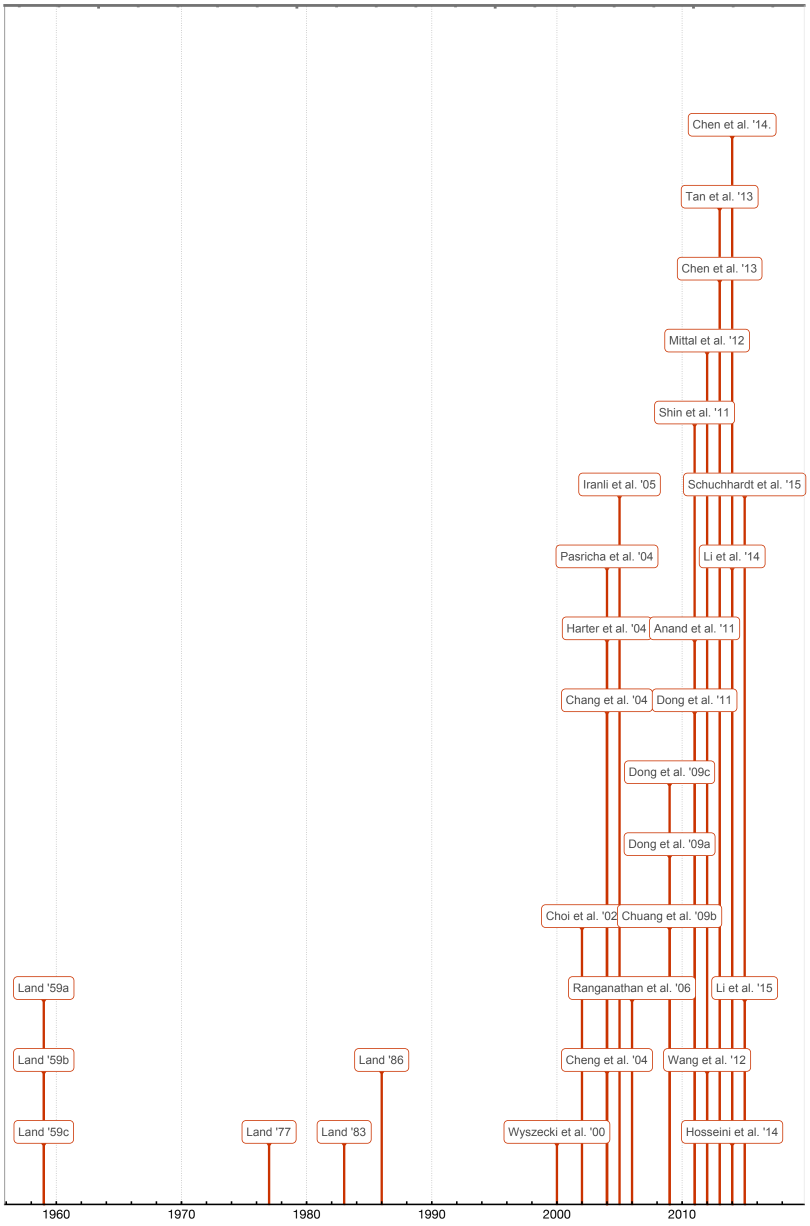


Figure 4.2: Timeline of referenced work in this chapter, listed by author.

5

Tolerating Errors in Inputs

> In most problems of applied mathematics and engineering the data are no better than $1 : 10^3$ or $1 : 10^4$ . . . and the answers are not required or meaningful with higher precisions either.
>
> —John von Neuman, *The computer and the brain*.

The data that serve as input for many important computational problems in the real world increasingly come from sensors of physical phenomena of various kinds. These sensors may range from accelerometers and gyroscopes in wearable and health-tracking systems, to continent-spanning radioastronomy telescopes.

Because there may be transient or persistent noise in sensor data, the computational problems which consume them, and hence the algorithms which embody these compute problems, can often operate on data of varying accuracy, precision, or reliability. Sensors, however, typically require different amounts of time and energy resources to generate data of different degrees of fidelity (see § 2.4.1 and Chapter 3). When hardware and system software permit, tolerance of imprecision, inaccuracy, and unreliability can be exploited: The tolerance can be harnessed to achieve sensor activation and sensor data acquisition which uses less energy, which is faster, or which is cheaper to build.

Lax [Stanley-Marbell and Rinard, 2015a] is one example of a system that builds on these insights. Lax improves the energy efficiency of sensor-driven systems by controlling the power supplies of sensors such as gyroscopes, so that they provide inaccurate, imprecise, or unreliable data, but consume significantly less power. Because of the empirically-observed variation in the type, frequency, and severity of sensor data errors with supply voltage, Lax uses descriptions of the amount of error that can be tolerated by applications to determine how much energy to save. These descriptions of tolerable error are provided in Lax's domain-specific language, but could in principle also be inferred by a compiler.

In addition to circuit techniques to reduce the power dissipation of sensor integrated circuit *operation* [Stanley-Marbell and Rinard, 2015a], it is possible to develop value encodings which reduce the power dissipated in *moving data* between sensing and computing devices, at the cost of controlled data infidelity [Stanley-Marbell and Rinard, 2015b, 2016, 2015c].

5.1 Lax

Lax uses two techniques, a combination of software and minimal hardware support, to trade efficiency for accuracy:

❶ **Device abstractions for approximation.** Lax provides a device driver abstraction, detailed in § 5.1.3, through which it enables sensor device accessors to specify the tolerable degree of imprecision and unreliability. Some sensors [Freescale Semiconductor, 2014b] already have limited support for modes which trade resolution for access power; for these, Lax can exploit the extant hardware facilities.

❷ **Sensor supply scaling.** Regardless of already existent hardware support for trading precision, accuracy, or reliability for power consumption, many sensors can be operated outside their specified supply voltages. This enables usable tradeoffs between the reliability of data acquisition, fidelity of data provided, and power dissipation. § 5.1.6 details the implementation of this hardware support.

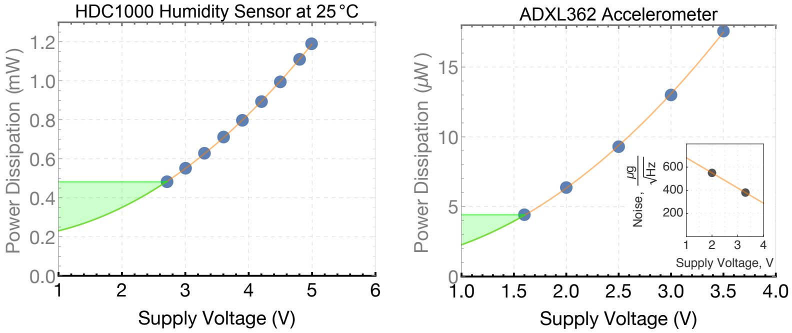


Figure 5.1: Manufacturer-reported power dissipation [Texas Instruments, 2014b, Analog Devices, 2014] for reliable operation at recommended supply voltages (points), extrapolated to lower voltages using a physics-based model (line). It falls by $60.6(1-k^2)$ and $62.6(1-k^2)$ percent respectively (shaded region), for each factor-k reduction from the lowest voltage for reliable operation.

5.1.1 The prospects for Lax

To illustrate the potential power dissipation versus error tradeoffs of Lax under sensor supply scaling, Figure 5.1 plots power dissipation and retrieved sample noise properties as a function of supply voltage for two state-of-the-art sensors. For both the humidity sensor and accelerometer, power dissipation decreases by a factor of ~4× with a halving of supply voltage. For the accelerometer, the noise in the retrieved signal (measured, by convention, in micro-gravities per square-root-Hertz, $\mu\text{g}/\sqrt{\text{Hz}}$) increases by a factor of ~1.5× with a halving of the supply voltage [Analog Devices, 2014] (inset).

5.1.2 Example: Lax in systems

Using contemporary operating systems, applications cannot specify how much precision, accuracy, or reliability they require from a sensor. Figure 5.2 shows the block diagram for a pedometer application, as might be incorporated into popular wearable health-tracking platforms. The figure shows the data flow from an accelerometer sensor through blocks of the signal processing needed to perform step counting [Zhao, 2010]. All the facilities for Lax can be logically interposed in the interface to data acquisition, as shown in the figure.

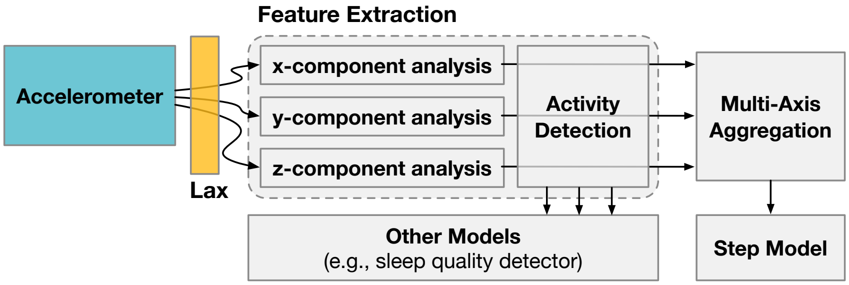
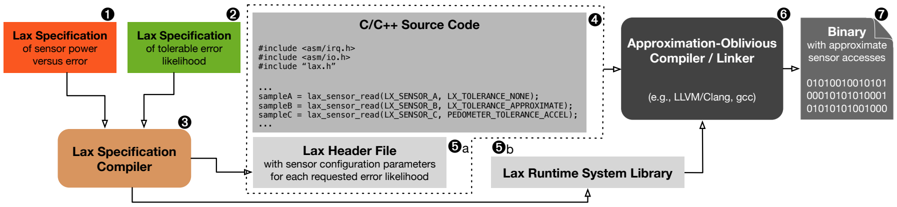


Figure 5.2: Data flow in a pedometer application. Samples from the x-, y-, and z-components of acceleration are typically first low-pass filtered, then fed into an activity detection algorithm. If the signature matches walking, these acceleration components are fed into a model for predicting steps from acceleration signatures.



```
#include <asm/irq.h>
#include <asm/io.h>
#include “lax.h”

...
sampleA = lax_sensor_read(LX_SENSOR_A, LX_TOLERANCE_NONE);
sampleB = lax_sensor_read(LX_SENSOR_B, LX_TOLERANCE_APPROXIMATE);
sampleC = lax_sensor_read(LX_SENSOR_C, PEDOMETER_TOLERANCE_ACCEL);
...
```

Figure 5.3: In this example, software uses Lax primitives to request sensor values (block ❹). The amount of inaccuracy, imprecision, or unreliability that is tolerable in responses to those requests is either specified using defaults such as **`LX_TOLERANCE_NONE`**, or application-specific tolerances such as **`PEDOMETER_TOLERANCE_ACCEL`**. Although not mandatory, if used, the meaning of these optional constants are specified explicitly in the tolerable error specification (block ❷). The Lax specification compiler must combine these with the hardware error characteristics (block ❶) to emit source and headers that implement the approximate sensor access (block ❺).

5.1.3 The Lax device interface

System software interfaces for approximate device access should enable the specification of three types of tolerance to deviations from correct behavior:

- **Latency tolerances.** Different applications may be able to tolerate differing latencies in retrieving values from a sensor. This can be exploited by the hardware facilities described in § 5.1.6 to reduce the energy required per sensor sample acquisition.

- **Loss or throughput tolerances.** When the algorithms consuming sensor data can tolerate occasional wholly-incorrect or missing samples, knowledge of this tolerance of unreliability can be used to reduce sample acquisition energy.

- **Value deviation tolerances.** When the algorithms consuming sensor values can tolerate small deviations from accuracy or precision in sensor readings, this can yet again be exploited to reduce sample acquisition energy.

Lax enables driver writers to specify tolerances to these types of behavioral aberrations. These specifications can then be exploited in existing system architectures, as illustrated in Figure 5.3.

5.1.4 Slax: Specifying Lax sensor access

Tolerances should be specified in the context of a given sensor type and should be statically checked because it is possible to specify meaningless, unattainable, or mutually-contradictory tolerance specifications. Lax provides a small domain-specific language, *Slax*, for defining tolerances. Slax captures the latency, loss, and value-deviation tolerances of sensor data acquisition and is thus complementary to interface definition languages such as Devil [Mérillon et al., 2000], which are intended to ease the construction of complete device drivers.

The grammar for Slax is shown in Figure 5.4, and an example specification for an accelerometer is given in Figure 5.5. A Slax specification comprises one or more `sensor` or `tolerance` blocks. The `sensor` blocks describe the error properties of sensors at various operating points, while the

```
unsignedImm    ::= "0" | "1..9" {"0..9"} .
stringConst    ::= "\"" {Unicode Character} "\"" .
integerConst   ::= ["+" | "-"] unsignedImm .
dRealConst     ::= unsignedImm "." "0..9" {"0..9"} .
eRealConst     ::= (dRealConst | integerConst) ("e" | "E") integerConst .
realConst      ::= dRealConst | eRealConst .
rationalConst  ::= integerConst "/" integerConst .
numConst       ::= integerConst | rationalConst | realConst .

slaxSpec       ::= specHead {defn} .
specHead       ::= "specification" ident ";" .
ident          ::= {Unicode Character} .
defn           ::= sensorDefn | toleranceDefn .
sensorDefn     ::= "sensor" ident ["@"numConst units] "=" "{" {sensorStmt} "}".
toleranceDefn  ::= "tolerance" ident "=" "{" {toleranceStmt} "}" .
sensorStmt     ::= "provide" "(" eClass ")" "=" "{" cStmt {";" cStmt} "}" .
toleranceStmt  ::= "require" "(" eClass ")" "=" "{" cStmt {";" cStmt} "}" .
eClass         ::= "deviation" | "latency" | "loss" | "throughput" .
cStmt          ::= cmpOp numConst units ":" likelihoodExpr | alwaysExpr .
likelihoodExpr ::= "likelihood" cmpOp numConst "in" numConst "readings" .
alwaysExpr     ::= "always" cmpOperator numConst .
cmpOp          ::= ">" | ">=" | "<" | "<=" | "==" .
units          ::= "s" | "ms" | "us" | "ns" | "W" | "mW" | "uW" | "nW" | "%" .

reservedTokens ::= "%" | "(" | ")" | ":" | ";" | "<" | "=" | ">" | "always"
                 | "deviation" | "in" | "latency" | "likelihood" | "loss"
                 | "ms"  |  "ns"  |  "occurs"  |  "provide"  |  "readings"
                 | "require"  |  "s"  |  "sensor"  |  "specification"
                 | "throughput"  |  "tolerance"  |  "us"  |  "{"  |  "}"  .
```

Figure 5.4: EBNF [Wirth, 1977] grammar for Slax, a domain-specific language for specifying latency, throughput, and value deviation tolerances for sensor access.

```
specification AccelerometerSensor;

sensor PLATFORM_ACCELEROMETER_A @ 1.6V = {
  provide (latency) {
     > 1 ms : likelihood < 1 in 1E6 readings;
  }
  provide (deviation) {
    > 1%  : likelihood < 1 in 1E6 readings;
    > 10% : likelihood < 1 in 1E9 readings;
  }
  provide (loss) {
    occurs: likelihood < 1 in 1E6 readings;
  }
}
```

```
specification PedometerApp;

tolerance PEDOMETER_TOLERANCE_ACCEL = {
  require (deviation) {
     > 1% : likelihood < 1 in 1000 readings;
  }
  require (latency) {
     > 1ms : likelihood < 1 in 1000 readings;
  }
  require (loss) {
     occurs : likelihood < 1 in 1000 readings;
  }
}
```

Figure 5.5: Example Slax specifications. The **`sensor`** and **`tolerance`** blocks capture sensor provisions and application requirements.

tolerance blocks denote groups of error tolerance settings that are required together at various points in an application.

In practice, a driver may use a Lax-default or driver-specified tolerance specification in accessing a given sensor, as illustrated in block ❹ of Figure 5.3. For example, given the Slax specifications in Figure 5.5, the following C fragment would employ the configuration implied by the constants **PLATFORM_ACCEL_A** and **PEDOMETER_TOLERANCE_ACCEL**:

```
/*   Use Lax to achieve lowest power for required accuracy.  */
sampleC = lax_sensor_read(PLATFORM_ACCEL_A,
                          PEDOMETER_TOLERANCE_ACCEL);
```

The Lax runtime must use the provided tolerance indicator to determine the best device operating point. When integrated into contemporary operating systems, it would then set the properties of the device using, e.g., **ioctl()** or equivalent system calls (our proof-of-concept implementation presented in § 5.1.6 runs over bare metal). The **sensor** blocks on the other hand must be based on hardware characterizations. They would ideally be provided by a hardware platform designer or vendor, but could be overridden by a driver writer's own **sensor** block. § 5.1.6 provides examples of the necessary characterizations that yield **sensor** blocks.

5.1.5 Challenges

Even though the potential benefits of exploiting tolerance to imprecision, inaccuracy, and unreliability are significant, there are several challenges to implementing a system that can effectively trade those tolerances for performance or energy efficiency. For example, a simplistic solution to determining a valid operating point from the **sensor** block for a given sensor might be straightforward, using, e.g., a lookup table. On the other hand, efficiently picking the operating point that satisfies the multiple constraints of deviation, latency, loss, and throughput tolerances, along with timing performance, average power, and overall energy usage, will be challenging. Other challenges include:

- **Obtaining Slax tolerance specifications**. These could be written by hand, as in Figure 5.5, when there are known sensor data fidelity requirements, or could be synthesized based on dataflow analyses of applications to determine their error-propagation properties [Linderman

Table 5.1: Sensor devices evaluated, their power dissipation, and supply voltage ranges for reliable operation.

Sensor	Power Dissipation (μW)	Supply Range (V)
Gyroscope		
L3G4200D [ST Microelectronics, 2010]	18300	2.4–3.6
IR Temperature		
TMP006B [Texas Instruments, 2014c]	528	2.5–5.5

et al., 2010, Benz et al., 2012].

- **Obtaining Slax `sensor` specifications**. As a first step, these could be constructed from manufacturer-provided data (such as in Figure 5.1), or from offline hardware measurements (§ 5.1.6). It would however be more versatile to be able to construct `sensor` specifications *in situ*, but such a facility would require appropriate hardware support.

- **Validity checking** of Slax specifications.

- **Dynamic adaptation** of the chosen operating point based on instantaneous environment conditions (e.g., temperature), based on OS or application feedback, or based on temporal histories of these.

Although we are implementing Slax using traditional compiler techniques, we are also investigating the potential benefits of integration with existing tools that ease DSL construction in systems software contexts, such as FoF [Dagand et al., 2009], HAIL [Sun et al., 2005], and Termite-2 [Ryzhyk et al., 2014].

5.1.6 Hardware prototype and evaluation

To verify that the energy savings for Lax are achievable, the following presents data integrity measurements at different degrees of power savings for two sensors. The sensors, listed in Table 5.1, are both targeted at mobile and wearable computing systems and each dissipate more power when active than the processor shown in Figure 5.6. In a typical system, they will also be sampled whenever the processor wakes from sleep, making their portion of the system's overall energy usage also significant.

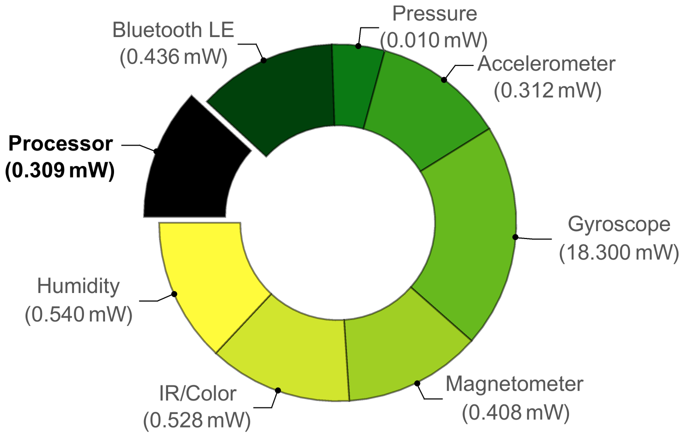


Figure 5.6: Logarithmically-scaled sector plot of relative power dissipation while active, for several state-of-the-art sensors [ST Microelectronics, 2010, 2014, Bosch Sensortec, 2014, Texas Instruments, 2014c,b], a Bluetooth Low Energy radio [Texas Instruments, 2014a] in *advertising/discoverable* mode, and an implementation of the lowest-power ARM architecture variant currently available (ARM Cortex M0+ [Freescale Semiconductor, 2014a]) running a **`while(1)`** loop from its on-chip SRAM at 2 MHz and 3.0 V. All but one of the sensors use more power than the processor.

The evaluation operates each sensor at a range of voltages below their nominal operating points and characterized the types of errors encountered. The possible errors under these conditions are of two types: ❶ *sample loss*, or *erasures* (in the information-theoretic [Cover and Thomas, 1991] sense), where communication with a sensor fails; ❷ *value deviations*, where values are retrieved from a sensor, but they are different from those that would have been retrieved when operating the sensor at its nominal operating voltage. Where appropriate, we modified the low-level interface code for accessing the sensors to recover gracefully from access failures (e.g., replacing assertions with more graceful return status codes); this worked well for our bare-metal embedded implementation. When Lax is integrated into a sophisticated operating system, such changes might still suffice, or may be augmented with, e.g., techniques such as microreboots [Candea et al., 2004] or tools such as Carburizer [Kadav et al., 2009].

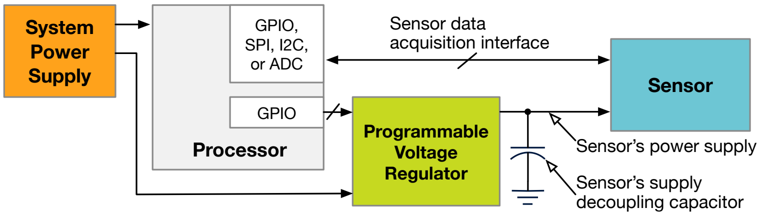

System Power Supply
GPIO, SPI, I2C, or ADC
GPIO
Processor
Sensor data acquisition interface
Programmable Voltage Regulator
Sensor
Sensor's power supply
Sensor's supply decoupling capacitor

Figure 5.7: Measurement setup for empirical validation of the feasibility of Lax.

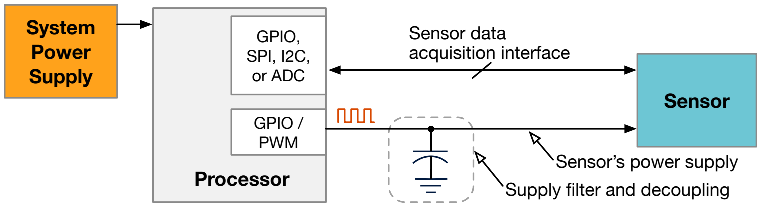


Figure 5.8: Programmable voltage regulators typically only output a discrete set of voltages. In some cases, they can be mimicked using a software-controlled pulse train and a filter.

Measurement setup

Figure 5.7 illustrates the measurement setup. The evaluation uses an ARM Cortex-M0+ processor [Freescale Semiconductor, 2014a] evaluation board to interface with the sensors, which are mounted on separate breakout boards. The processor also controls a pair of programmable voltage regulators [Texas Instruments, 2014d,e] which enable the sensors to be operated at nine discrete voltages between 1.2 V and 2.5 V, with the operating voltage dynamically switchable under software control. These regulators have small circuit board footprint and low overheads, with quiescent currents in the nano-Amperes. Alternatively, the configuration shown in Figure 5.8 could be used to achieve even finer-grained control of sensor power supplies, with lower circuit overhead and possibly better efficiency in powering the sensor device than the programmable voltage regulator, but at the cost of additional software on the control processor.

Preliminary results and relation to Slax

Figure 5.9 shows the reduction in dynamic power dissipation from operating sensors below their nominal voltages, along with the measured sensor data

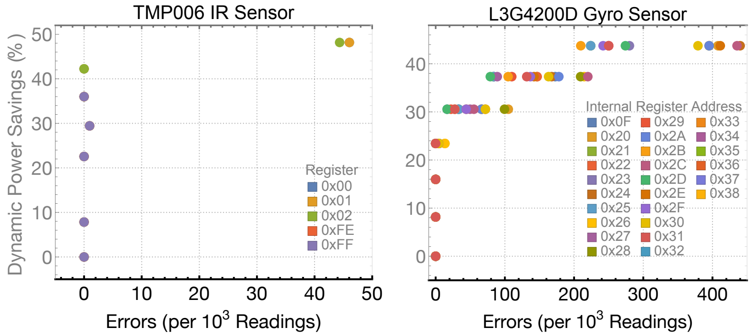
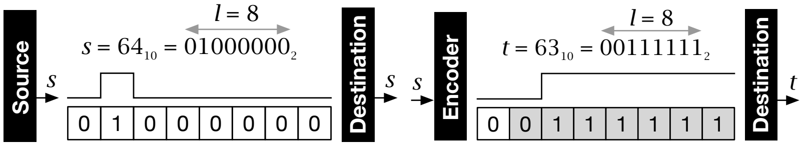


Figure 5.9: Power savings per access versus acquisition error rate for a state-of-the-art infrared sensor [Texas Instruments, 2014c] (left) and gyroscope [ST Microelectronics, 2010] (right).

acquisition error rates (i.e., erasures). The measurements were performed at sensor I2C [NXP Semiconductors, 2014] interface data rates of 1 kb/s. The savings in dynamic power dissipation for both sensors are significant: up to 48% for the IR temperature sensor, and up to 42% for the gyroscope. In both cases, savings of up to 16% are possible with no data acquisition errors, and error rates increase with increasing savings. The error rate at the configuration of maximum power reduction is less than 5 errors per 100 accesses for the IR temperature sensor, but as high as 1 out of every 2 accesses for the gyroscope at maximum savings.

The data in the figure are precisely the information required to construct a `provide(loss){...}` block of a `sensor` statement in Slax, as occurs, e.g., in the example Slax specification in Figure 5.5. Similarly, analysis of sensor values relative to a ground truth would enable synthesis of a `provide(deviation){...}` block, while data acquisition at different data rates will yield the necessary information for `provide(latency){...}` and `provide(throughput){...}` specifications.

Most sensors with digital interfaces can be queried for multiple distinct pieces of information, such as the sensor's configuration, silicon junction temperature (for software compensation algorithms), and so on. Each type of information is typically accessed via a different register internal to the sensor, and the various types of information are not equally important. The error rates observed in Figure 5.9 vary with which of the sensor's registers are accessed.



(a) Without VDBS encoding.

(b) With VDBS: fewer transitions.

Figure 5.10: VDBS encoding reduces transitions while incurring a value deviation, $|s - t|$, of 1. In this example, assume the tolerable deviation, m, is 13 (i.e., 5 % of 255). VDBS encoding halves the number of transitions while incurring a value deviation, $|s - t|$, of just 0.39 % of the full-scale range. All bits except the most-significant bit are modified (shown shaded in (b)), not just the lower $\lceil \log_2(|s - t|) \rceil$ bits.

Slax `sensor` specifications built from such characterization data can capture this variation through the use of separate `sensor` blocks for each register, or by incorporating the observed variation in the `likelihood` expressions.

5.2 VDBS Encoding

Dynamic power dissipation in serial interfaces occurs when consecutive serialized bits of the same word differ. The number of such transitions between consecutive bits of the same word are referred to as the *serial transition count* (STC). The maximum STCs occur when words have alternating 0s and 1s in their binary representations. Figure 5.10 shows how modifying transmitted words can reduce the STC at the cost of small deviations from accuracy.

5.2.1 Formal definition of VDBS encoders

VDBS encoding generalizes the idea illustrated in Figure 5.10. Considering both STC reduction and induced deviation, the Pareto-optimal VDBS encoders either minimize the induced deviation, maximize the STC reduction, or both.

Definition 1 (*Family of optimal VDBS encoders*)**.**
Let s and t be two unsigned l-bit integers representing unencoded and encoded words, respectively. Let m be the difference in numeric value between s and t, and let $\#_\delta(k)$ be the STC for an integer k. The Boolean predicate $\mathrm{P}_{s,t,m}$ denotes the constraint satisfied by all VDBS encoders that maintain or

reduce STCs while inducing a deviation less than or equal to m:

$$\mathrm{P}_{s,t,m} = (|s-t| \leq m) \wedge ((\#_\delta(s) - \#_\delta(t)) \geq 0)\,.$$

Let $\Delta_{s,t} = |\#_\delta(s) - \#_\delta(t)|$ be the difference in serial transition counts between two words s and t. Given an input word s and integer m indicating how much deviation in s is acceptable, there are four possible encoding functions that satisfy the Boolean predicate $\mathrm{P}_{s,t,m}$. These functions define the bounds on transition reduction and value deviation:

$$\begin{aligned}
\mathrm{e}_1(s,m) &= \left(\tau \text{ s.t. } \mathrm{P}_{s,\tau,m} \wedge \left(|s-\tau| = \min_{0<i<2^l-1} |s-i|\right)\right),\\
\mathrm{e}_2(s,m) &= \left(\tau \text{ s.t. } \mathrm{P}_{s,\tau,m} \wedge \left(|s-\tau| = \max_{0<i<2^l-1} |s-i|\right)\right),\\
\mathrm{e}_3(s,m) &= \left(\tau \text{ s.t. } \mathrm{P}_{s,\tau,m} \wedge \left(\Delta_{s,\tau} = \min_{0<i<2^l-1} \Delta_{s,i}\right)\right),\\
\mathrm{e}_4(s,m) &= \left(\tau \text{ s.t. } \mathrm{P}_{s,\tau,m} \wedge \left(\Delta_{s,\tau} = \max_{0<i<2^l-1} \Delta_{s,i}\right)\right). \blacksquare
\end{aligned}$$

In what follows, we restrict our treatment to unsigned integers. The analysis easily extends to two's-complement, fixed-, and floating-point representations.

5.2.2 Properties of the optimal VDBS encoders

The four functions e_1 through e_4 bound the amount by which VDBS encoders reduce STCs and bound the deviation they induce:

- $\mathbf{e_1(s,m)}$ causes the smallest deviations.
- $\mathbf{e_2(s,m)}$ causes the largest deviations.
- $\mathbf{e_3(s,m)}$ reduces STCs the least.
- $\mathbf{e_4(s,m)}$ reduces STCs the most.

Our objective is to obtain a method for VDBS encoding whose behavior encompasses the best of the properties of all the above encoders: induced deviation close to that of e_1 and STC reduction close to that of e_4.

The subset of three encoder types e_1, e_3, and e_4 are Pareto-optimal when considering both serial transition reduction and deviation. Because it is strictly dominated by e_4, the encoder e_2 is not in the Pareto set. The behavior of the simplistic encoder that for a given tolerable deviation m only removes transitions from the lower $\lceil \log_2(m) \rceil$ bits is similar to e_2 (§ 5.2.6).

Two essential components in the formulation of VDBS encoders are:

❶ **The number of serial transitions** that occur when a single value s is transmitted over a serial link (two transitions in Figure 5.10(a)). This is the *serial transition count* (STC) of the word or value s.

❷ **The difference in serial transition counts** between two words (a difference of one between s and t in Figure 5.10).

Throughout this work, the values considered will be unsigned.

Definition 2 (*Serial transition count function,* $\#_\delta(s)$)**.**
Let s be an l-bit unsigned integer with bits $s_0, s_1, \ldots, s_{l-1}$, from least- to most-significant bit. Then, we define $\#_\delta(s)$, the number of signal transitions in the serialization of s, as

$$\#_\delta(s) = \sum_{i=0}^{l-2} s_i \oplus s_{i+1}. \qquad \blacksquare$$

Definition 3 (*Serial transition count difference,* $\Delta_{s,t}$)**.**
Let s and t be two l-bit words. Then, we define $\Delta_{s,t}$, as the absolute value of their difference in serial transition counts:

$$\Delta_{s,t} = |\#_\delta(s) - \#_\delta(t)|. \qquad \blacksquare$$

5.2.3 Properties of function $\#_\delta(n)$

For an l-bit value n, the properties of the serial transition count (STC) function $\#_\delta(n)$, which we explore next, give insights into the efficiency limits of VDBS encoders.

Proposition 1 (Maximum serial transition count pattern)**.**
When l is even, the maximum serial transition count occurs when the l-bit word has $\frac{l}{2}$ 0s and the same number of 1s. □

Proof (Maximum serial transition count pattern)**.**
To maximize the serial transition count, there should be a transition in moving

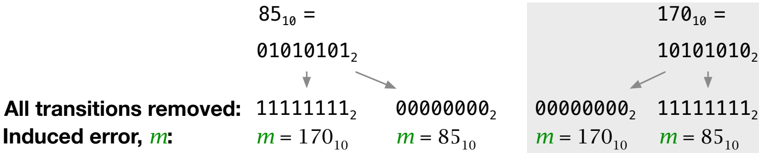


Figure 5.11: The maximum serial transition counts for l-bit values occur when they have alternating 0s and 1s in their binary representations.

between every neighboring pair of bit positions. Thus, when l is even, words with maximum serial transition count have $\frac{l}{2}$ 0s and the same number of 1s. ■

Corollary 1 (Maximum serial transition count basis values)**.**
There are two values with maximum serial transition count. When l is even, these values are

$$\hat{b}_1 = \sum_{i=0}^{\frac{l}{2}-1} 2^{2i} = \tfrac{1}{3}(2^l - 1) \tag{5.1}$$

and

$$\hat{b}_2 = 2^l - 1 - \sum_{i=0}^{\frac{l}{2}-1} 2^{2i} = \tfrac{2}{3}(2^l - 1).$$

□

This follows directly from Proposition 1. For example, Figure 5.11 illustrates how, for $l = 8$, the maximal-serial-transition-count words are 85 and 170.

Lemma 1 (Maximum serial transition count)**.**
For every l-bit word n, $\#_\delta(n) \leq l - 1$. □

Proof (Maximum serial transition count)**.**
The number of bits (l) in a word is a natural number. When l is 1, there are no transitions in the word, by definition of the serial transition count. For all other l, the maximum serial transition count occurs when all adjacent bits of the word differ. There are four cases in which this could happen, corresponding to whether l is even or odd, and whether the least-significant bit (LSB) is a 1 or a 0.

First, consider the cases when l is even. When l is even, there are $\frac{l}{2}$ ones and $\frac{l}{2}$ zeros. If the LSB is 0, there will be one transition in moving from the LSB towards the most-significant bit (MSB), and each of the remaining $\frac{l}{2}-1$ bits which are 0 will have two associated transitions. There will therefore be a total of $l-1$ transitions. A similar argument applies if the LSB is 1.

Next, consider the cases when l is odd. When l is odd, there are either $\lfloor \frac{l}{2} \rfloor$ bits which are 1 and $\lceil \frac{l}{2} \rceil$ bits which are 0, or vice versa. The bit polarity appearing in the LSB will occur $\frac{l-1}{2}+1$ times, and the opposite polarity to the LSB will occur $\frac{l-1}{2}$ times.

There will be one transition moving out of the LSB towards the MSB, followed by transitions in the remaining $l-1$ bits. Since l is odd, it follows that $l-1$ is even. But we showed above that such an even number of bits could contain at most $(l-1)-1$ transitions. Thus, when l is odd, the maximum number of transitions is also $1+(l-1)-1$. That is, the maximum number of transitions is $l-1$. ■

Theorem 1 (Serial transitions and Gray code)**.**
Let s be an l-bit integer, let $\mathrm{GrayCode}(s)$ denote the sth value in Gray code order for l-bit values, and let $\#_1(n)$ denote the count of 1s in an l-bit integer n. Then, $\#_\delta(s) = \#_1(\mathrm{GrayCode}(s))$. □

For example, for $l = 8$ and $s = 30$ (00011110_2), $\mathrm{GrayCode}(s) = 17$ (00010001_2) and $\#_1(\mathrm{GrayCode}(s)) = 2$.

We will use the following, the Gray code theorem of Wilf [Wilf and Nijenhuis, 1989], in the proof of Theorem 1. We include a self-contained adaptation of Wilf's original proof here so that our discussion stands on its own.

Theorem 2 (Wilf's Gray Code Theorem)**.**
Let s be an l-bit integer with bits $s_0, s_1, \ldots, s_{l-1}$, from least- to most-significant bit. Let g be the sth l-bit integer in Gray code order, with bits $g_0, g_1, \ldots, g_{l-1}$, from least- to most-significant bit. For $l = 1$ we have $g_0 = s_0$. In general, for $l \geq 2$,

$$g_i \equiv s_i + s_{i+1} \pmod 2 \quad (i = 0, \ldots, l-2)$$

and

$$g_{l-1} = s_{l-1}. \qquad \square$$

$\mathcal{L}_0 = \{\}$	$l = 0$
$\mathcal{L}_0^0 = \{0\}$	
$\overline{\mathcal{L}_0}^1 = \{1\}$	
$\mathcal{L}_1 = \{0, 1\}$	$l = 1$
$\mathcal{L}_1^0 = \{00, 01\}$	
$\overline{\mathcal{L}_1}^1 = \{11, 10\}$	
$\mathcal{L}_2 = \{00, 01, 11, 10\}$	$l = 2$
$\mathcal{L}_2^0 = \{000, 001, 011, 010\}$	
$\overline{\mathcal{L}_2}^1 = \{110, 111, 101, 100\}$	
$\mathcal{L}_3 = \{000, 001, 011, 010, 110, 111, 101, 100\}$	$l = 3$

Figure 5.12: Illustration of the construction of the list $\mathcal{L}_l$ of l-bit strings in Gray code order, for $l = 1$, $l = 2$, and $l = 3$.

For example, consider the 8-bit value 63. The string of rank 63 in the 8-bit Gray code, that is, the 63rd Gray code value, can be constructed as follows: For the ith bit, simply take the ith and $i+1$th bits of 63, and add them modulo 2.

Proof (Wilf's Gray Code Theorem)**.**
Let $\mathcal{L}_l$ be the list of l-bit strings in Gray code order. $\mathcal{L}_0$ is the empty list. The list $\mathcal{L}_l$ can be constructed recursively as follows:

- Let $\mathcal{L}_{l-1}^0$ be the list obtained by prefixing every element of $\mathcal{L}_{l-1}$ with an additional 0.
- Let $\overline{\mathcal{L}_{l-1}}^1$ be the list obtained by prefixing every element of the list $\mathcal{L}_{l-1}$, in reverse order, with an additional 1.
- $\mathcal{L}_l$ is the concatenation of $\mathcal{L}_{l-1}^0$ and $\overline{\mathcal{L}_{l-1}}^1$.

The construction of the list $\mathcal{L}_l$ is illustrated in Figure 5.12 for $l = 1$, $l = 2$, and $l = 3$. By construction therefore, the 2^l entries for an l-bit Gray code will be identical to the first 2^l entries for an $(l + 1)$-bit Gray code; we use this property below.

We prove by induction on l that the property of Theorem 2 holds for all l-bit integers s. When $l = 0$, $\mathcal{L}_l$ is the empty list, and the property we seek to prove is vacuously true. Suppose the property of Theorem 2 holds for all strings on the list $\mathcal{L}_{l-1}$. By construction of $\mathcal{L}_l$, we know the property must also hold for the first 2^{l-1} items on $\mathcal{L}_l$. Suppose then, that $s \geq 2^{l-1}$. Let $s' = 2^l - 1 - s$. Then the property of Theorem 2 holds for the string that has Gray code rank s', since it is by its definition less than 2^{l-1}.

Again by construction of the Gray code lists $\mathcal{L}_l$ from $\mathcal{L}_{l-1}$, it is the case that for any given value $0 \leq k < 2^{l-1}$, the first $l-1$ bits of the Gray code strings g and g' with ranks $s = k$ and $s' = k$ are identical. Furthermore, the most-significant bits, g_{l-1} and g'_{l-1}, of these corresponding strings, have the relation

$$g_{l-1} \equiv 1 + g'_{l-1} \pmod 2.$$

At the same time, the binary representations of the integers s and s' have the relation

$$s_i \equiv 1 + s'_i \pmod 2 \quad (i = 0, \ldots, l-1),$$

and the property of Theorem 2 continues to hold for all strings on the list $\mathcal{L}_l$. ■

We now use Wilf's Gray code theorem to prove the property of Theorem 1, which relates properties of transitions within a single word, s, when serialized, to properties of the rank-s Gray code.

Proof (Serial transitions and Gray code)**.**
The proof is a direct result of Theorem 2. Let g be the Gray code representation for l-bit integer s. That is, g is the rank-s l-bit Gray code. The number of 1s in g, $\#_1(g)$, is

$$\begin{aligned}\#_1(g) &= \sum_{i=0}^{l-1} g_i \\ &= \sum_{i=0}^{l-2} \left(s_i + s_{i+1} \pmod 2\right), \quad \text{from Theorem 2} \\ &= \sum_{i=0}^{l-2} \left(s_i \oplus s_{i+1}\right).\end{aligned}$$

But this is exactly the $\#_\delta(s)$ from Definition 2. ■

5.2.4 Bounds on serial transition count reduction

We can reduce the number of serial transitions in words without changing the word size, by introducing errors into the values represented by words. The maximum number of serial transitions we can remove by doing so, is limited:

Property 1 (Bound on serial transition count difference)**.**
For any two l-bit words s and t, the serial transition count difference, $\Delta_{s,t}$ is less than or equal to $l - 1$. □

Proof (Bound on serial transition count difference)**.**
By construction, the serial transition count, $\#_\delta(s)$ for a non-negative integer s, is a natural number. Therefore, the largest serial transition count difference, will occur when either $\#_\delta(s)$ is zero and $\#_\delta(t)$ takes on the maximum value in the codomain of $\#_\delta(t)$, or vice versa. From Lemma 1, this maximum value is $l-1$. Thus the maximum serial transition count difference, $\Delta_{s,t}$ is $l-1$. ■

Across all possible l-bit words, the deviation induced when transitions are reduced by the maximum of $l - 1$, is bounded:

Property 2 (Minimum and maximum deviation at maximum serial transition count difference)**.**
Let s and t be two l-bit words with l even. If s and t differ in serial transition count by the maximum possible amount ($l - 1$), then their difference in numeric value is bounded by:

$$\min_{\Delta_{s,t}=l-1} \{|s-t|\} = \tfrac{1}{3}\left(2^{\Delta_{s,t}+1} - 1\right), \tag{5.2}$$

and

$$\max_{\Delta_{s,t}=l-1} \{|s-t|\} = \tfrac{2}{3}\left(2^{\Delta_{s,t}+1} - 1\right). \tag{5.3}$$

□

Proof (Minimum and maximum deviation at maximum serial transition count difference)**.**
Follows directly from Corollary 1. ■

For example, for $l = 8$, we have from Lemma 1 that the maximal serial transition count difference is $l - 1 = 7$. The minimum deviation between two words which have this maximum serial transition count difference, from Property 2, is 85. Therefore, to reduce the serial transition count of an 8-bit word by 7 transitions, one cannot do so with a replacement word that deviates from it by less than 85.

The bounds of Property 2 are only specified for the case of maximal changes in serial transition count, not for any arbitrary reduction in serial transition count. General bounds across all possible values of serial transition count reduction are desirable, because they would enable us to answer questions such as:

- **By how much can serial transition counts differ** for a given value deviation? This will be captured by Definition 4 and Theorem 3 below.
- **By how much can values differ** for a given difference in serial transition count? Property 2 answers this question for the restricted case of a serial transition count difference of $l - 1$. The answer for the general case will be captured by Definition 5 below.

Definition 4 (Serial transition difference bound function)**.**
Given an l-bit integer m, let $\mathrm{f}(m)$ be a function yielding the amount by which the serial transition counts of two unsigned l-bit words s and t can differ if $|s - t| = m$. That is,

$$\mathrm{f}(m) = \max_{|s-t|=m} \{\Delta_{s,t}\}\,. \qquad \blacksquare$$

Why $\mathrm{f}(m)$ **is important:** The function $\mathrm{f}(m)$ is interesting because, if one had an exact expression or tight bounds for $\mathrm{f}(m)$, then an algorithm that searched for the serial-transition-reducing encoding for a value s could terminate as soon as it found a value t such that $\Delta_{s,t} = \mathrm{f}(m)$, since no better value than t is possible.

Theorem 3 (Bound on $\mathrm{f}(m)$)**.**
The function $\mathrm{f}(m)$ of Definition 4, for any l-bit value, m (with l even), is not monotone. The best linear monotone bound on $\mathrm{f}(m)$ is $\mathrm{f}(m) \leq l - 1$. □

Proof (Bound on $\mathrm{f}(m)$)**.**
Let s and t be two unsigned l-bit words, and let m be $|s - t|$, a value in the

domain of f. If m is 0, then s is identical to t, and must have identical serial transition count, thus $\#_\delta(s) = \#_\delta(t)$ and therefore $\mathrm{f}(0) = 0$. If m is $2^l - 1$, then either s is $2^l - 1$ and t is zero, or vice versa. In both cases, their serial transition counts are 0 by definition, that is $\#_\delta(s) = \#_\delta(t) = 0$. Thus, when m is $2^l - 1$, $\mathrm{f}(m) = 0$.

From Corollary 1 and Lemma 1, the maximum value of $\mathrm{f}(m)$ is $l - 1$, and it occurs at two values, $\hat{b}_1$ and $\hat{b}_2$ from Equation 5.1. Both $\hat{b}_1$ and $\hat{b}_2$ are greater than 0 and less than $2^l - 1$. Since $\mathrm{f}(0)$ is 0, $\mathrm{f}(\hat{b}_1)$ is $l-1$, $\mathrm{f}(\hat{b}_2)$ is $l-1$, and $\mathrm{f}(2^l - 1)$ is 0, it follows that $\mathrm{f}(m)$ is not monotone.

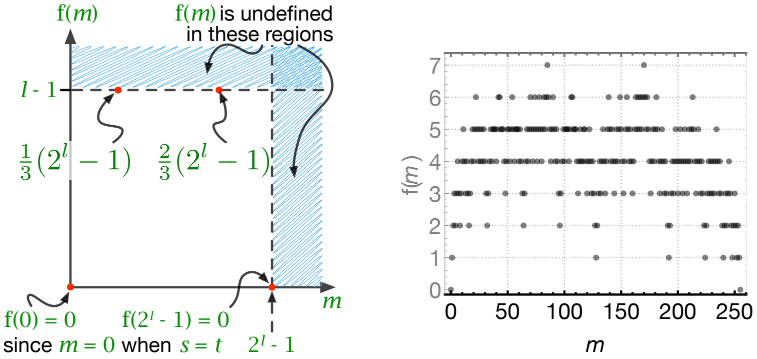
From Corollary 1 and Lemma 1, since there are two values of m for which $\mathrm{f}(m)$ takes on its maximum value of $l - 1$, it follows that the tightest linear bound on $\mathrm{f}(m)$ must pass through these points. Thus the tightest linear bound on $\mathrm{f}(m)$ is $l - 1$. ■

Figure 5.13(a) illustrates several properties of $\mathrm{f}(m)$, and Figure 5.13(b) shows an empirical exact enumeration of $\mathrm{f}(m)$ across all possible unsigned 8-bit values. The maximum value of m is $2^l - 1$ and the maximum value of $\mathrm{f}(m)$ is $l - 1$, as indicated by the shaded region in Figure 5.13(a). There can be no reduction in serial transition count when the accompanying deviation in value is 0, and thus $\mathrm{f}(0) = 0$. Similarly, when the deviation induced by encoding is $2^l - 1$ (i.e., the original and encoded values are 0 and $2^l - 1$ or vice versa), there can be no reduction in serial transition count, and thus $\mathrm{f}(2^l - 1) = 0$. The maxima of $\mathrm{f}(m)$ occur at $m = \frac{1}{3}(2^l - 1)$ and $m = \frac{2}{3}(2^l - 1)$.

Definition 5 (Value deviation bound functions)**.**
Let $\mathrm{g}(d)$ be the minimum amount by which two integers s and t can differ if their difference in serial transition count, $\Delta_{s,t}$, is d. Similarly, let $\hat{\mathrm{g}}(d)$ be the maximum amount by which two integers s and t can differ if their difference in serial transition count, $\Delta_{s,t}$, is d. That is,

$$\mathrm{g}(d) = \min_{\Delta_{s,t}=d} \{|s - t|\}, \quad \text{and} \quad \hat{\mathrm{g}}(d) = \max_{\Delta_{s,t}=d} \{|s - t|\}. \ ■$$

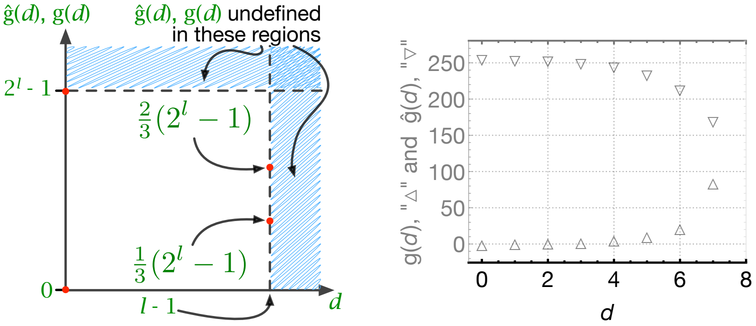
Figure 5.14(a) illustrates several properties of $\mathrm{g}(d)$ and $\hat{\mathrm{g}}(d)$, and Figure 5.14(b) shows an empirical exact enumeration of $\mathrm{g}(d)$ and $\hat{\mathrm{g}}(d)$ for unsigned 8-bit values. When there is no difference in serial transition count ($d = 0$ in the figures) the original and encoded values may be identical ($\mathrm{g}(0) = 0$) or may be 0 and $2^l - 1$ or vice versa ($\hat{\mathrm{g}}(0) = 2^l - 1$). At the



(a) Illustration of $\mathrm{f}(m)$.

(b) Numerical evaluation of $\mathrm{f}(m)$.

Figure 5.13: The function $\mathrm{f}(m)$ yielding the amount by which the serial transition counts of two words s and t can differ if $|s - t| = m$, is not monotone.



(a) Illustration of $\mathrm{g}(d)$ and $\hat{\mathrm{g}}(d)$.

(b) Numerical evaluation: $\mathrm{g}(d)$, $\hat{\mathrm{g}}(d)$.

Figure 5.14: At minimum serial transition count (STC) difference, $d = 0$, either s and t are identical, or they are different but take on values $s = 0$ and $t = 2^l - 1$ or vice versa.

maximum possible serial transition count reduction ($d = l - 1$), the incurred deviation cannot be reduced below $\frac{1}{3}(2^l - 1)$; the worst-case deviation at this maximal-transition-reduction point is however also limited, at $\frac{2}{3}(2^l - 1)$.

5.2.5 Rake: Efficient VDBS encoding

Given an unencoded value s in which an application can tolerate a value deviation m, the family of encoders of Definition 1 specify the possible optimum ways in which encoding can reduce serial transitions in s. The encoders of Definition 1 also determine the amount of deviation that an encoding will induce, for a given selected deviation that applications can tolerate. Exact algorithms for the optimal encoders must however select an encoded value for s out of a set whose size is exponential in the word size of s. A brute-force application of the predicate in Definition 1 is therefore inefficient even if applied offline to generate a lookup table (LUT) and is impractical for large word sizes.

To address the cost of the Pareto-optimal encoders of Definition 1, particularly for large word sizes, we present Rake, an efficient algorithm for VDBS encoding. Rake's execution time is linear in the word size of the values it encodes. For a specified deviation m in its encoded values, Rake reduces transitions more than the basic technique that simply removes all transitions from the lower-order $\lceil \log_2(m) \rceil$ bits. At the same time, Rake reduces transitions almost as much as the Pareto-optimal VDBS encoder e_4 that minimizes the serial transition count for a given tolerable deviation. On average, Rake incurs value deviations smaller than all the Pareto-optimum VDBS encoders except e_1 (which minimizes value deviation). We call the algorithm Rake because it operates in two sweeps of a word, accumulating metadata in the first sweep and leveling out transitions in the second. The Rake algorithm (Algorithm 1) operates as follows.

In the first phase (lines 1 to 6), moving across the l-bit input word s from least-significant bit (LSB) to most-significant bit (MSB), Rake stores the number of transitions seen to-date in the *transition count register*, nt. Rake stores the indices of these transitions in the *transition indices array*, tr (line 2). For each transition, Rake stores the length of the run of 0s or 1s leading to the transition, in the *run length temporary register*, rl (line 3). Each such run of 0s or 1s could be bit-wise negated to either increase or decrease

Algorithm 1: Outline of Rake algorithm for VDBS encoding.

```
/* First phase, from LSB to MSB; the transition count
   register, $nt$, stores count of transitions seen.          */
for $nt \leftarrow 0; i \leftarrow 0, i < l, i \leftarrow i + 1$ do
    /* If adjacent bits differ, store transition location
       in $tr[]$.                                               */
    if $(i < l - 1)$ and $(s_i \neq s_{i+1})$ then
        $tr[nt] \leftarrow i$
        if $(nt > 0)$ then
            /* $rl$ gets length of 0- or 1-run that transition
               abuts; $rc$ gets contrib.                        */
            $rl \leftarrow tr[nt] - tr[nt - 1]$
            $rc \leftarrow (2^{rl} - 1) << tr[nt - 1]$
        if $(nt > 0)$ and $(s_i = 0)$ then
            /* Store contrib. of run of 0s in $cr0c$.           */
            $cr0c[i] \leftarrow rc$
        if $(nt > 0)$ and $(s_i = 1)$ then
            /* Store contrib. of run of 1s in $cr1c$.           */
            $cr1c[i] \leftarrow rc$
        $nt \leftarrow nt + 1$
    else if $i > 0$ then
        /* Pad cumulative count arrays.                         */
        $cr0c[i] \leftarrow cr0c[i - 1]$
        $cr1c[i] \leftarrow cr1c[i - 1]$
/* Second phase, from MSB to LSB; inspect only the $nt$ bit
   positions that have transitions.                             */
while $nt > 0$ do
    /* $rl$: run length; $rc$ run's contrib. if its bits were
       flipped to remove transition:                            */
    $rl \leftarrow tr[nt] - tr[nt - 1]$
    $rc \leftarrow (2^{rl} - 1) << tr[nt - 1]$
    /* Can deviation caused by negating bits be offset by
       negating runs of lower-order bits?:                      */
    if $(s_{tr[nt-1]} = 0)$ and $((rc - cr1c[nt - 1]) \leq m)$ then
        return $(s + rc - cr1c[nt - 1])$
    if $(s_{tr[nt-1]} = 1)$ and $((rc - cr0c[nt - 1]) \leq m)$ then
        return $(s - rc + cr0c[nt - 1])$
    $nt \leftarrow nt - 1$
return $s$
```

the value of s. Rake stores the change in value that such a negation would contribute, in the *cumulative run contribution arrays*, $cr0c$ for runs of 0s and $cr1c$ for runs of 1s (lines 4 and 5).

In the second phase (lines 7 to 10), Rake moves across the input in the opposite direction, from MSB to LSB, inspecting only the nt bit positions that have transitions. Rake previously stored these locations in tr. For each of the nt transition locations in tr, Rake checks whether the value deviation incurred by negating the bits that constitute a transition could be offset by the runs of lower-order bits of opposite polarity, as represented by the contents of $cr0c$ and $cr1c$ (lines 8 and 9). Rake removes the first transition that passes this check and completes. Rake takes l steps as it traverses from the LSB to the MSB, followed by at most $nt - 2$ steps in the opposite direction. The maximum value of nt is $l-1$, thus Rake takes a maximum of $2l-3$ steps. For example, for 24-bit values, Rake requires only 45 steps, compared to having to explore a space of 16 million values for the exact optimal solution.

Rake is not only efficient, but also effective: Rake reduces transitions almost as much as the optimal VDBS encoder e_4 as we show in § 5.2.6. By contrast, the naive approach of simply removing transitions from the lower-order $\log_2(m)$ bits for a tolerable value deviation of m does not reduce transitions as much as Rake does.

5.2.6 Numerical evaluation

Two objective metrics are important for VDBS encoders:

❶ **The average serial transition count reduction** for a given word size and tolerable deviation.

❷ **The average actual deviation** that is induced by encoders for a given tolerable deviation.

We evaluate both the ideal encoders of § 5.2 as well as the Rake encoder of § 5.2.5 under these two measures, by applying the encoders to all possible unsigned words with sizes of 8 and 16 bits. These sizes are representative of the range of word sizes for sensor and ADC values used in real-world systems. (We provide detailed end-to-end application evaluations in § 5.3.)

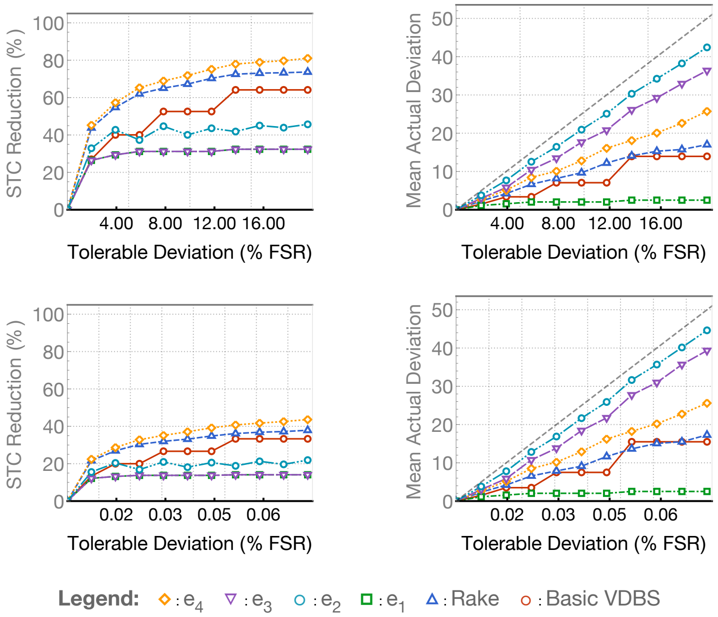


Figure 5.15: Mean serial transition count (STC) reduction and actual deviation versus tolerable deviation expressed as a fraction of the full-scale range (FSR). 8-bit values (top row) and 16-bit values (bottom row). The dashed gray line in the right column shows where actual deviation equals tolerable deviation.

Transition reduction and induced deviation

We evaluate Rake and the Pareto-optimal encoders by applying them to all possible unsigned values for a given word size, l. The word sizes we evaluate are $l = 8$ and $l = 16$. For each word size, we select 10 values of tolerable deviation, m, uniformly spaced between 0 and 50. For each value of tolerable deviation, we apply each of the Rake and Pareto-optimal encoders to all l-bit values. From the resulting 2^l encoded values for each encoder, we compute the mean serial transition reduction at each value of tolerable deviation m. From the encoded values paired with their original unencoded values, for each encoder, we compute the mean induced deviation at each tolerable deviation m. Figure 5.15 presents the results. The figure plots the percentage reduction in serial transition count and the average value deviation resulting from encoding, as functions of the tolerable deviation specified during encoding (expressed as a percentage of the full-scale range of l-bit values).

The top row of Figure 5.15 shows the results for 8-bit values. For a tolerable deviation of 10 % of the full-scale range of 8-bit values, Rake reduces signal transitions by 67 %. For this *tolerable deviation*, the *mean actual deviation* is 4 % of the full-scale range (i.e., 10). The results in Figure 5.15 show that Rake reduces serial transitions more than all but one of the Pareto-optimal encoders: Rake reduces transitions by only 5 percentage points less than the optimal encoder that minimizes serial transitions (e_4). The average deviation induced by Rake is also better than all but one of the Pareto-optimal encoders: At a tolerable deviation of 10 % of the full-scale range, Rake's induced deviation is less than 4 percentage points worse than the optimal encoder that minimizes deviation (e_1). Even at moderate tolerable deviations of 5 % of the full-scale range, Rake reduces transitions almost twice as much as existing encoding techniques for deviation-free serial buses [Chiu et al., 2013].

The results for 16-bit words follow a similar trend (bottom row of Figure 5.15). For a tolerable deviation of 0.12 % of the full-scale range, Rake reduces signal transitions by 41 % on average, while inducing deviations of 0.05 % of the full-scale range, on average.

Effective number of bits of encoded values

The effective number of bits (ENOB) denotes the number of unique levels representable by encoded values and is computed as $\log_2(|\{\text{unique encoder output values}\}|)$. Representing values with fewer bits reduces the number of signal transitions within transmitted words and in the clock signal. Figure 5.16 presents the serial transition count reduction as a function of the ENOB, for Rake-encoded 8-bit words as well as for progressively shorter unencoded words. For a given ENOB, Rake encoding of 8-bit words reduces transitions up to 60 % more than simply employing shorter unencoded words that have the same ENOB.

VDBS encoders such as Rake have several additional advantages over simply employing smaller word sizes. VDBS encoding reduces transitions without requiring changes to the datapath of applications (e.g., without requiring changes to algorithms to use 5-bit data instead of 8-bit data). And VDBS encoding provides 7.4-times finer-grained control of the amount of transition reduction, because it enables fractional steps in the ENOB.

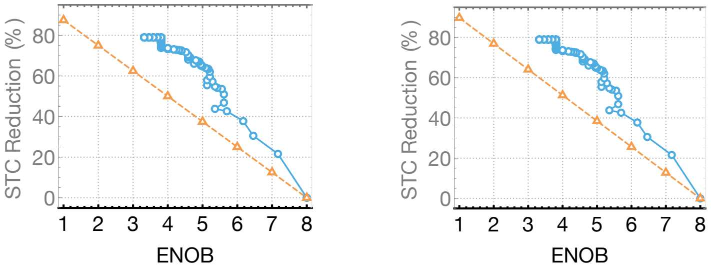


Figure 5.16: Rake encoding (○) reduces the serial transition count (STC) by up to 24 percentage points (i.e., a 60% improvement in the fraction of transitions removed) more than shorter words of equal ENOB (△), considering only intra-word transitions (left), and total intra-word and clock transitions (right).

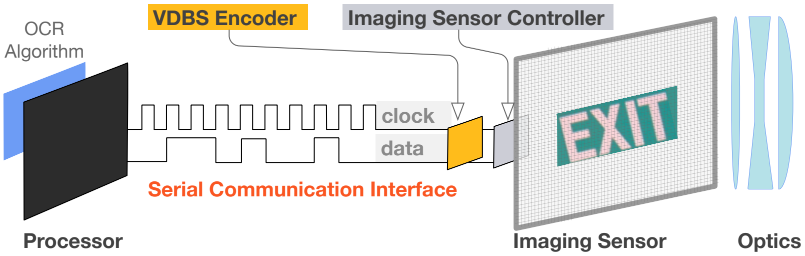


Figure 5.17: Illustration of a VDBS encoder in an optical character recognition application.

5.3 End-to-end Evaluation

We evaluate Rake in two end-to-end application settings. The evaluation results indicate that Rake can significantly reduce signal transitions in exchange for small deviations in encoded values. Because these deviations are often masked by the data-flow of common sensor signal processing algorithms, the deviations lead to only small errors at the application level.

5.3.1 Encoding data in a text-recognition system

We apply Rake to images in transfer between a camera and processor in a text-recognition system such as that illustrated in Figure 5.17. Text recognition is an important component of many applications, such as augmented reality systems. We evaluate the amount by which Rake reduces data transfer signal transitions as well as its end-to-end effect on optical character recognition (OCR) errors.

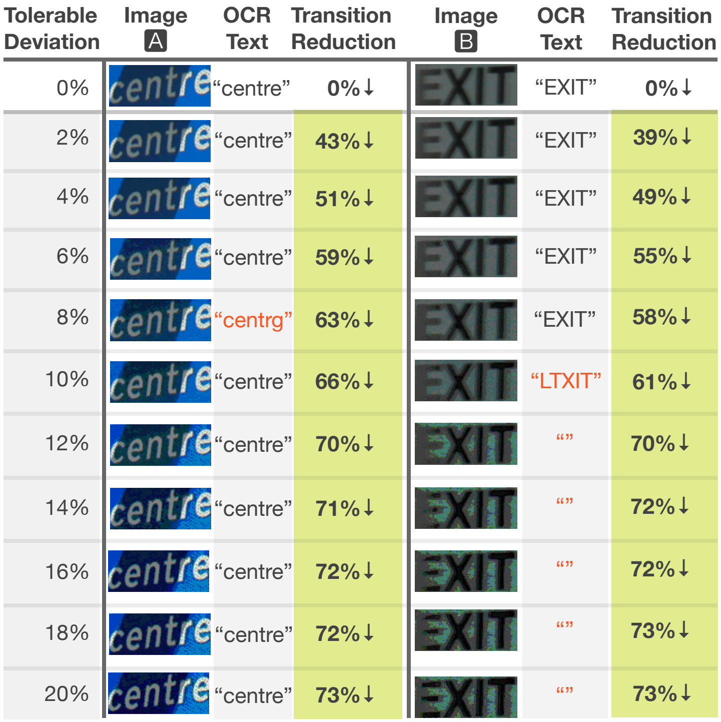
We use version 3.02 of the Tesseract OCR system [Smith, 2007], widely

RESTAURANT CHOCOMILK Accommodation COLONNADE GOVERNMENT PIGEONS
OPPORTUNITIES SUPERKINGS CHOCOMILK DOUBLEMINT DEPARTMENT FUNFASHION YOU
MONITORING unauthorised DOUBLEMINT Mini-Statement CHEWING MAYNARD information
CHAMBER VEHICLES REPAIRS HOSPITAL Management REMINDER ANALYSIS Counselling
CRAFTSMAN GLASS PENALTY ESPRESSO Breakfast SMOKING SPRINKLER JOURNAL is
Contractors September Department PLEASE MAXIMUM LINGERIE FAVOURITE BELFAST HSBC
HARWICH interest HOTMAIL accessorize ESPRESSO BURTON BUFFETT available Psychology
FASHION COURSES September TALLEST MUSEUM SERVICE Residence HISTORY GALLERY to
Committee AMERICAN NATURAL ROGERS EPISODE CONRAD operation DARKNESS Bookshop Castle
September SCHOOL Doubles CROSSING Committee SERVICE STATION Hygiene University Chairman in
AGENTS BEWARE Oldenburg, university TRINITY Peacocks Lifelines Organising CLONES SIMPSON Mastering
DIXON Camden alternative PHARMACY ANTIQUES HOMER Greenstead Factory University Truman WHARF OF
Departure HOUSE anyone TICKET! systems Oldenburg SOUTH CANARY TRAFFIC Meditative FEED London
STREET WAIT COURT House Tourist LADIES OFFICE MAYFAIR Tiffany's CELMAC Secretary STAFF
HOUSE Conference KINDA RUGBY Mastering CRAFT QUEEN Section CABOT Entrance SNOUT EQUAL Centre
Ordering Estates Permit PRIVATE starting WALK William LITTER leading Station Support Balance Nursery
Second Account PLANT Wolfson FLATS centre USERS tickets SHOP Builders Cheque machines available up
HEART Special BANK eyecare ROUTE Morris CLEANED iiyama Please Alarm and DISTRIBUTION Capote BLACK
Whisky SUPPLIES STORE South Essex Houses Towers FAMOUS cycling NATIVE system HOSE FIRED centre II
Whisky GUM one Library CHURCH HERE GUM Fitted KNOW JOSEPH SHOP THE Longridge Keynes, HOT
Essex Tetley MILK County GIFTS EARTH ONLY Culver Eastern parcel Enter Coach Fidelity CLUB National is
Tawney PARK Office DATA NEW centre Essex Know YOUR HOSE MILK White cashier Grapes Meet Court
Spirit clear STORAGE buses work Videos ITEMS riser ENDS Ipswich THE CITY GAS Used HOT Open area!
COM Buses Express Please Cycles KEEP apollo Essex Better Next YOU REEL PARK from Travel ONLY WALK WAY of
Germ the hour Keep Save EXIT FOR Germ DO centre HILL this Peacocks Book ARE THE CAR Centre Year aid
HFC Centre Centre one DRY MILL Post Centre dept this Parks KEY EAST AND Sports Rose Parks clinic CAR Trail First
(1958) YOU First EXIT into ANY High E14 Adult ARE OF Here and and lens find Town inch this PLUS here end cash
Fire out wait aid The the for Park line John Do ship the SV and for the Car for Car the the IT CANS In 2,0 the THE S
A12 owns the big No.1 at IN of The the cut to of of

Figure 5.18: 392 image subset from the ICDAR text recognition dataset [Wong et al., 2003] used in evaluation. This is the subset for which Tesseract [Smith, 2007] correctly reports OCR text identical to the benchmark-supplied ground truth.

regarded to be the most accurate open-source OCR package. For input, we use the test set from the ICDAR text image dataset [Wong et al., 2003] and select as our baseline the 392 images (Figure 5.18) for which Tesseract returns the same recognition text as the benchmark's ground truth. We then apply Rake to each of these 392 text images, with degrees of tolerable deviation ranging from 0 % to 20 % of the full-scale range of the 8-bit per-color-channel pixel values. We quantify the errors in text recognition using the standard edit-distance-based metric used in the text-recognition literature [Rice et al., 1997]. Figure 5.19 presents an example of the effect of Rake on two input text images, as well as the effect on OCR accuracy and on transitions in the serialized image data. The examples in Figure 5.19 illustrate how Rake applied to image data can significantly reduce transitions without affecting the output of OCR algorithms applied to the images.

Figure 5.20 presents Rake's serial transition count reduction and its induced reduction in OCR accuracy as functions of the tolerable deviation in encoded values. The results in Figure 5.20 show that Rake reduces transitions significantly with minimal effect on OCR error. With a target tolerable deviation of 5 %, Rake reduces serial transitions by over 55 %, while maintaining an OCR accuracy of over 90 % for previously-correctly-recognized text.



Tolerable Deviation	Image A	OCR Text	Transition Reduction	Image B	OCR Text	Transition Reduction
0%		"centre"	0%↓		"EXIT"	0%↓
2%		"centre"	43%↓		"EXIT"	39%↓
4%		"centre"	51%↓		"EXIT"	49%↓
6%		"centre"	59%↓		"EXIT"	55%↓
8%		"centrg"	63%↓		"EXIT"	58%↓
10%		"centre"	66%↓		"LTXIT"	61%↓
12%		"centre"	70%↓		""	70%↓
14%		"centre"	71%↓		""	72%↓
16%		"centre"	72%↓		""	72%↓
18%		"centre"	72%↓		""	73%↓
20%		"centre"	73%↓		""	73%↓

Figure 5.19: For two example images (first row, second and fifth columns), higher tolerable deviation enables more transition reduction, at the cost of OCR errors.

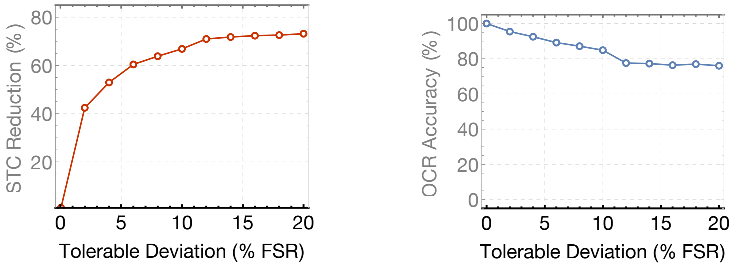


Figure 5.20: Averaged across the 392 images, for a target tolerable deviation of 5 %, Rake reduces the serial transition count (STC) by 55 % while keeping OCR accuracy above 90 %.

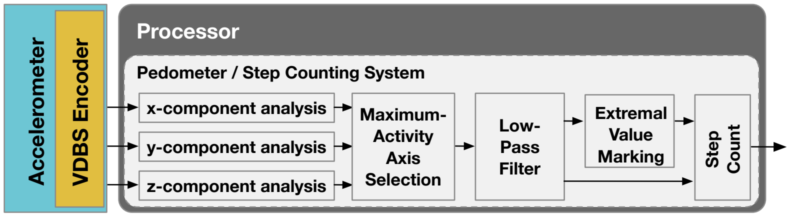


Figure 5.21: A VDBS encoder within a pedometer system.

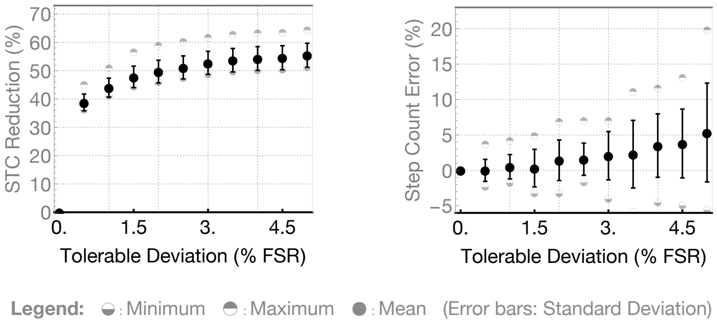


Figure 5.22: For a target tolerable deviation of 4 %, Rake reduces the serial transition count (STC) by 54 % on average, while inducing step count errors of less than 5 % on average.

5.3.2 Encoding data in a pedometer system

We apply Rake to accelerometer data in a pedometer system (Figure 5.21). Pedometer facilities are central to many health and wellness applications and these applications constitute a growing market with important positive societal impact.

We use 3-axis accelerometer data sampled at 20 Hz, a total of 334377 samples or over 4.6 hours worth of walking. The samples are taken from 12 different users in the publicly-available WISDM activity recognition dataset [Kwapisz et al., 2011]. The WISDM dataset provides real-valued samples. In practice, however, actual accelerometer sensors provide a fixed number of bits of resolution, either directly or through the use of an ADC. We therefore convert the samples to 13-bit values to match the resolution of a state-of-the-art accelerometer [Zhao, 2010]. We then apply Rake to the 13-bit data, with degrees of tolerable deviation ranging from 0 % to 5 % of the full-scale range of values, before passing the encoded data to a step counting algorithm [Zhao, 2010]. Figure 5.22 presents the resulting reduction in serial transition count and the induced step count errors as functions of the tolerable deviation. The results show that at target tolerable deviations of 4 %, Rake reduces transitions by up to 63 % with a mean of 54 %, inducing step counting errors of less than 5 % on average.

6

Conclusion

The traditional mechanisms for improving computing systems performance and energy efficiency, by shrinking transistor dimensions based on Dennard scaling rules are approaching fundamental physical limits. As a result, computing systems have ceased to provide the efficiency and performance benefits on which modern society has come to depend. Without these continued improvements, new applications ranging from personalized health and computer vision applications for the visually impaired, to big data analysis for climate modeling, may be at risk. Error-efficient hardware, software, and algorithm design are a new class of techniques to achieve improved performance and energy efficiency by only providing as much accuracy, precision, or reliability as applications require. They provide a new and compelling alternative to the traditional approaches to computing system design and an alternative path to continued improvements of computing performance and efficiency from which modern society will continue to benefit. Research in this research area has however to-date been conducted by separate research communities, each focusing either on computer-aided design , circuits, computer architecture, systems software, or programming languages.

This review provided an introduction and review of error-efficient computing systems, from their historical context, to the fundamental concepts that underpin them (faults, errors, randomization), to the the reasons why they are interesting (tradeoffs between correctness and resource usage). The goal of the review is to provide a holistic picture of the challenges in error-efficient systems, from device technology to human perception and psychophysics.

References

Ismail Akturk, Karen Khatamifard, and Ulya R Karpuzcu. On quantification of accuracy loss in approximate computing. In *Workshop on Duplicating, Deconstructing and Debunking (WDDD)*, pages 15–, 2015.

Armin Alaghi and John P. Hayes. Survey of stochastic computing. *ACM Trans. Embed. Comput. Syst.*, 12(2s):92:1–92:19, May 2013.

Andres Albanese, Johannes Blömer, Jeff Edmonds, Michael Luby, and Madhu Sudan. Priority encoding transmission. *Information Theory, IEEE Transactions on*, 42(6):1737–1744, 1996.

R. Amirtharajah and A. P. Chandrakasan. A micropower programmable DSP using approximate signal processing based on distributed arithmetic. *IEEE Journal of Solid-State Circuits*, 39(2):337–347, 2004.

Analog Devices. *ADXL362 Micropower, 3-Axis,* ±2 g / ±4 g / ±8 g *Digital Output MEMS Accelerometer*, Data Sheet, 2014.

Bhojan Anand, Karthik Thirugnanam, Jeena Sebastian, Pravein G. Kannan, Akhihebbal L. Ananda, Mun Choon Chan, and Rajesh Krishna Balan. Adaptive display power management for mobile games. In *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services*, MobiSys '11, pages 57–70, New York, NY, USA, 2011. ACM.

Todd Austin, David Blaauw, Trevor Mudge, and Krisztián Flautner. Making typical silicon matter with razor. *Computer*, 37:57–65, March 2004.

Todd Austin, Valeria Bertacco, David Blaauw, and Trevor Mudge. Opportunities and challenges for better than worst-case design. In *Proceedings of the 2005 Asia and South Pacific Design Automation Conference*, ASP-DAC '05, pages 2–7, New York, NY, USA, 2005. ACM.

A. Avizeinis. The N-Version Approach to Fault-Tolerant Software. *IEEE Transactions of Software Engineering*, SE-11(12):1491–1501, December 1985.

A. Avižienis, J.C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE transactions on dependable and secure computing*, pages 11–33, 2004.

H. Aydın, R. Melhem, and D. Mossé. Incorporating Error Recovery into the Imprecise Computation Model. In *The Sixth International Conference on Real-Time Computing Systems and Applications (RTCSA '99)*.

Woongki Baek and Trishul M. Chilimbi. Green: a framework for supporting energy-conscious programming using controlled approximation. In *Proceedings of the 2010 ACM SIGPLAN conference on Programming language design and implementation*, PLDI '10, pages 198–209, New York, NY, USA, 2010. ACM.

Yu Bai and Mingjie Lin. Energy-efficient discrete signal processing with field programmable analog arrays (fpaas). In *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '15, pages 84–93, New York, NY, USA, 2015. ACM.

David H Bailey. High-precision floating-point arithmetic in scientific computation. *Computing in science & engineering*, 7(3):54–61, 2005.

J. Bau, R. Hankins, Q. Jacobson, S. Mitra, B. Saha, and A.R. Adl-Tabatabai. Error resilient system architecture (ERSA) for probabilistic applications. In *IEEE Workshop on Silicon Errors in Logic-System Effects, SELSE*, 2007.

Robert C. Baumann. Radiation-Induced Soft Errors in Advanced Semiconductor Technologies. 5(3):305–316, September 2005.

Charles H Bennett and Rolf Landauer. The fundamental physical limits of computation. *Scientific American*, 253(1):48–56, 1985.

Florian Benz, Andreas Hildebrandt, and Sebastian Hack. A dynamic program analysis to find floating-point accuracy problems. In *Proceedings of the 33rd ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '12, pages 453–462. ACM, 2012.

Toby Berger. Rate-distortion theory. *Encyclopedia of Telecommunications*, 1971.

B. R. Borgerson and R. F. Freitas. A reliability model for gracefully degrading and standby-sparing systems. *IEEE Transactions on Computers*, c-24:517–525, May 1975.

Shekhar Borkar, Tanay Karnik, and Vivek De. Design and reliability challenges in nanometer technologies. In *Proceedings of the 41st annual conference on Design automation*, pages 75–75. ACM Press, 2004.

D. Borodin, B. H. H. B. Juurlink, S. Hamdioui, and S. Vassiliadis. Instruction-Level Fault Tolerance Configurability. *Journal of Signal Processing Systems*, 57(1): 89–105, 2009.

Bosch Sensortec. *BMX055 Small, Versatile 9-axis Sensor Module*, Data Sheet, November 2014.

Douglas C. Bossen, Joel M. Tendler, and Kevin Reick. Power4 system design for high reliability. *IEEE Micro*, 22(2):16–24, 2002.

Melvin Breuer. Multi-media applications and imprecise computation. In *Proceedings of the 8th Euromicro Conference on Digital System Design*, pages 2–7, Washington, DC, USA, 2005a. IEEE Computer Society.

Melvin Breuer. Hardware that produces bounded rather than exact results. In *Proceedings of the 47th Design Automation Conference*, DAC '10, pages 871–876, New York, NY, USA, 2010. ACM.

Melvin A. Breuer. Multi-media applications and imprecise computation. In *Digital Systems Design, Euromicro Symposium on*, pages 2–7, Los Alamitos, CA, USA, 2005b. IEEE Computer Society.

L. Budin, D. Jakobović, and M. Golub. Genetic algorithms in real-time imprecise computing. *Journal of Computing and Information Technology*, 8(3):249, 2004.

M. L. Bushnell and V. D. Agrawal. *Essentials of electronic testing for digital, memory, and mixed-signal VLSI circuits*. Springer Netherlands, 2000.

George Candea, Shinichi Kawamoto, Yuichi Fujiki, Greg Friedman, and Armando Fox. Microreboot — a technique for cheap recovery. In *Proceedings of the 6th Conference on Symposium on Operating Systems Design & Implementation - Volume 6*, OSDI'04, pages 31–44. USENIX Association, 2004.

Michael Carbin, Sasa Misailovic, and Martin C. Rinard. Verifying quantitative reliability for programs that execute on unreliable hardware. In *Proceedings of the 2013 ACM SIGPLAN International Conference on Object Oriented Programming Systems Languages & Applications*, OOPSLA '13, pages 33–52, New York, NY, USA, 2013. ACM.

Srimat T. Chakradhar and Anand Raghunathan. Best-effort computing: re-thinking parallel software and hardware. In *Proceedings of the 47th Design Automation Conference*, DAC '10, pages 865–870, New York, NY, USA, 2010. ACM.

K. Mani Chandy and Leslie Lamport. Distributed snapshots: determining global states of distributed systems. *ACM Trans. Comput. Syst.*, 3(1):63–75, 1985.

Naehyuck Chang, Inseok Choi, and Hojun Shim. Dls: dynamic backlight luminance scaling of liquid crystal display. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 12(8):837–846, Aug 2004.

Chardonnereau, Damien and Keulen, Raijmond and Nicolaidis, Michael and Dupont, Eric and Torki, Kholdoun and Faure, Fabien and Velazco, Raoul. 32-Bit RISC Processor Implementing Transient Fault-Tolerant Mechanisms and its Radiation Test Campaign Results. In *Single-Event Effects Symp.*, NASA, April 2002.

Xiang Chen, Yiran Chen, Zhan Ma, and Felix C. A. Fernandes. How is energy consumed in smartphone display applications? In *Proceedings of the 14th Workshop on Mobile Computing Systems and Applications*, HotMobile '13, pages 3:1–3:6, New York, NY, USA, 2013. ACM.

Xiang Chen, Kent W. Nixon, Hucheng Zhou, Yunxin Liu, and Yiran Chen. Fingershadow: An oled power optimization based on smartphone touch interactions. In *6th Workshop on Power-Aware Computing and Systems (HotPower 14)*, Broomfield, CO, 2014. USENIX Association.

Wei-Chung Cheng, Yu Hou, and Massoud Pedram. Power minimization in a backlit tft-lcd display by concurrent brightness and contrast scaling. In *Proceedings of the Conference on Design, Automation and Test in Europe - Volume 1*, DATE '04, pages 10252–, Washington, DC, USA, 2004. IEEE Computer Society.

Wei-Chung Cheng, Chih-Fu Hsu, and Chain-Fu Chao. Temporal vision-guided energy minimization for portable displays. In *Low Power Electronics and Design, 2006. ISLPED'06. Proceedings of the 2006 International Symposium on*, pages 89–94, Oct 2006.

Vinay K. Chippa, Debabrata Mohapatra, Anand Raghunathan, Kaushik Roy, and Srimat T. Chakradhar. Scalable effort hardware design: exploiting algorithmic resilience for energy efficiency. In *Proceedings of the 47th Design Automation Conference*, DAC '10, pages 555–560, New York, NY, USA, 2010. ACM.

Ching-Te Chiu, Wen-Chih Huang, Chih-Hsing Lin, Wei-Chih Lai, and Ying-Fang Tsao. Embedded transition inversion coding with low switching activity for serial links. *IEEE TVLSI*, 21(10):1797–1810, October 2013.

Inseok Choi, Hojun Shim, and Naehyuck Chang. Low-power color tft lcd display for hand-held embedded systems. In *Proceedings of the 2002 International Symposium on Low Power Electronics and Design*, ISLPED '02, pages 112–117, New York, NY, USA, 2002. ACM.

I. S. Chong and A. Ortega. Power Efficient Motion Estimation using Multiple Imprecise Metric Computations. In *2007 IEEE International Conference on Multimedia and Expo*, pages 2046–2049, 2007.

Johnson Chuang, Daniel Weiskopf, and Torsten Möller. Energy aware color sets. *Computer Graphics Forum*, 28(2):203–211, 2009.

Cristian Constantinescu. Neutron ser characterization of microprocessors. In *DSN '05: Proceedings of the 2005 International Conference on Dependable Systems and Networks (DSN'05)*, pages 754–759, Washington, DC, USA, 2005. IEEE Computer Society.

Thomas M. Cover and Joy A. Thomas. *Elements of information theory*. Wiley series in telecommunications. John Wiley & Sons, New York-Chichester-Brisbane-Toronto-Singapore, 1991.

Pierre-Evariste Dagand, Andrew Baumann, and Timothy Roscoe. Filet-o-fish: Practical and dependable domain-specific languages for os development. In *Proceedings of the Fifth Workshop on Programming Languages and Operating Systems*, PLOS '09, pages 5:1–5:5. ACM, 2009.

Robert H Dennard, Fritz H Gaensslen, V Leo Rideout, Ernest Bassous, and Andre R LeBlanc. Design of ion-implanted mosfet's with very small physical dimensions. *IEEE Journal of Solid-State Circuits*, 9(5):256–268, 1974.

Mian Dong and Lin Zhong. Chameleon: A color-adaptive web browser for mobile oled displays. In *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services*, MobiSys '11, pages 85–98, New York, NY, USA, 2011. ACM.

Mian Dong, Yung-Seok Kevin Choi, and Lin Zhong. Power-saving color transformation of mobile graphical user interfaces on oled-based displays. In *Proceedings of the 2009 ACM/IEEE International Symposium on Low Power Electronics and Design*, ISLPED '09, pages 339–342, New York, NY, USA, 2009a. ACM.

Mian Dong, Yung-Seok Kevin Choi, and Lin Zhong. Power modeling of graphical user interfaces on oled displays. In *Proceedings of the 46th Annual Design Automation Conference*, DAC '09, pages 652–657, New York, NY, USA, 2009b. ACM.

Hadi Esmaeilzadeh, Adrian Sampson, Luis Ceze, and Doug Burger. Architecture support for disciplined approximate programming. In *Proceedings of the Seventeenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XVII, pages 301–312, New York, NY, USA, 2012a. ACM.

Hadi Esmaeilzadeh, Adrian Sampson, Luis Ceze, and Doug Burger. Neural acceleration for general-purpose approximate programs. In *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-45, pages 449–460, Washington, DC, USA, 2012b. IEEE Computer Society. .

Freescale Semiconductor. *Kinetis KL03 32 KB Flash 48 MHz Cortex-M0+ Based Microcontroller*, Data Sheet, August 2014a.

Freescale Semiconductor. *MMA8451Q 3-Axis, 14-bit/8-bit Digital Accelerometer*, Data Sheet, November 2014b.

J. George, B. Marr, B. E. S. Akgul, and K. V. Palem. Probabilistic arithmetic and energy efficient embedded signal processing. In *Proceedings of the 2006 International Conference on Compilers, Architecture and Synthesis for Embedded Systems*, CASES '06, pages 158–168, New York, NY, USA, 2006. ACM.

Daniel T Gillespie. A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. *Journal of computational physics*, 22(4): 403–434, 1976.

Daniel T Gillespie. Exact stochastic simulation of coupled chemical reactions. *The journal of physical chemistry*, 81(25):2340–2361, 1977.

M. C. Golumbic and A. N. Trenk. *Tolerance graphs*. Cambridge Univ Pr, 2004.

Noah D. Goodman. The principles and practice of probabilistic programming. In *Proceedings of the 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '13, pages 399–402, New York, NY, USA, 2013. ACM.

Noah D. Goodman, Vikash Mansinghka, Daniel M Roy, Keith Bonawitz, and Joshua B Tenenbaum. Church: a language for generative models. *arXiv preprint arXiv:1206.3255*, 2012.

Andrew D. Gordon, Thore Graepel, Nicolas Rolland, Claudio Russo, Johannes Borgstrom, and John Guiver. Tabular: A schema-driven probabilistic programming language. In *Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '14, pages 321–334, New York, NY, USA, 2014a. ACM. .

Andrew D. Gordon, Thomas A. Henzinger, Aditya V. Nori, and Sriram K. Rajamani. Probabilistic programming. In *Proceedings of the on Future of Software Engineering*, FOSE 2014, pages 167–181, New York, NY, USA, 2014b. ACM. .

L. Guo, M. Scott, and R. Amirtharajah. An energy scalable computational array for sensor signal processing. In *IEEE Custom Integrated Circuits Conference, 2006. CICC'06*, pages 317–320, 2006.

Richard W Hamming. Error detecting and error correcting codes. *Bell System technical journal*, 29(2):147–160, 1950.

Tim Harter, Sander Vroegindeweij, Erik Geelhoed, Meera Manahan, and Parthasarathy Ranganathan. Energy-aware user interfaces: An evaluation of user acceptance. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '04, pages 199–206, New York, NY, USA, 2004. ACM.

B. Hayes. A lucid interval. *American Scientist*, 91(6):484–488, 2003.

Yun He and Chris H. Q. Ding. Using accurate arithmetics to improve numerical reproducibility and stability in parallel applications. *J. Supercomput.*, 18(3):259–277, March 2001.

Rajamohana Hegde and Naresh R. Shanbhag. Energy-efficient signal processing via algorithmic noise-tolerance. In *Proceedings of the 1999 International Symposium on Low Power Electronics and Design*, ISLPED '99, pages 30–35, New York, NY, USA, 1999. ACM. .

William Heidergott. SEU Tolerant Device, Circuit and Processor Design. In *DAC '05: Proceedings of the 42nd annual conference on Design automation*, pages 5–10, New York, NY, USA, 2005. ACM Press.

C. A. R. Hoare. Algorithm 64: Quicksort. *Commun. ACM*, 4(7):321–, July 1961.

Henry Hoffmann. Jouleguard: Energy guarantees for approximate applications. In *Proceedings of the 25th Symposium on Operating Systems Principles*, SOSP '15, pages 198–214, New York, NY, USA, 2015. ACM.

Robert W. Horst, Richard L. Harris, and Robert L. Jardine. Multiple instruction issue in the NonStop cyclone processor. In *ISCA '90: Proceedings of the 17th annual international symposium on Computer Architecture*, pages 216–226, New York, NY, USA, 1990. ACM Press.

D. Hull and J. Liu. ICS: A system for imprecise computations. In *Proc. AIAA Computing in Aerospace*, volume 9, 1993.

Ali Iranli and Massoud Pedram. Dtm: Dynamic tone mapping for backlight scaling. In *Proceedings of the 42nd Annual Design Automation Conference*, DAC '05, pages 612–617, New York, NY, USA, 2005. ACM.

Francois Jacquet. Design of SRAMs in Scaled CMOS Technologies. Seminar, Center for Silicon System Implementation (CSSI), Carnegie Mellon University, 2006.

R. Jongerius, P. Stanley-Marbell, and H. Corporaal. Quantifying the Common Computational Problems in Contemporary Applications (Extended version), 2014. IBM Research Report RZ3885.

Asim Kadav, Matthew J. Renzelmann, and Michael M. Swift. Tolerating hardware device failures in software. In *Proceedings of the ACM SIGOPS 22Nd Symposium on Operating Systems Principles*, SOSP '09, pages 59–72. ACM, 2009.

Andrew B. Kahng, Seokhyeong Kang, Rakesh Kumar, and John Sartori. Recovery-driven design: a power minimization methodology for error-tolerant processor modules. In *DAC '10: Proceedings of the 47th Design Automation Conference*, pages 825–830, New York, NY, USA, 2010. ACM.

Kevin B. Kenny and Kwei-Jay Lin. Building flexible real-time systems using the flex language. *Computer*, 24(5):70–78, 1991.

Robert W Keyes. What makes a good computer device? *Science*, 230(4722):138–144, 1985.

G.J. Klir. The many faces of uncertainty. *Machine Intelligence and Pattern Recognition*, 17:3–3, 1994.

I. Koren and C. M. Krishna. *Fault Tolerant Systems*. Morgan Kaufmann Publishers Inc. San Francisco, CA, USA, 2007.

U. Kulisch. *Computer arithmetic and validity: theory, implementation, and applications*. de Gruyter, 2008.

Jennifer R. Kwapisz, Gary M. Weiss, and Samuel A. Moore. Activity recognition using cell phone accelerometers. *SIGKDD Explor. Newsl.*, 12(2):74–82, March 2011.

Edwin H Land. Color vision and the natural image. part i. *Proceedings of the National Academy of Sciences*, 45(1):115–129, 1959a.

Edwin H Land. Color vision and the natural image part ii. *Proceedings of the National Academy of Sciences*, 45(4):636–644, 1959b.

Edwin H Land. Experiments in color vision. *Scientific American*, 200(5):84, 1959c.

Edwin H Land. The retinex theory of color vision. *Scientific American*, 237(6):108, 1977.

Edwin H Land. Recent advances in retinex theory and some implications for cortical computations: color vision and the natural image. *Proceedings of the National Academy of Sciences*, 80(16):5163–5169, 1983.

Edwin H Land. An alternative technique for the computation of the designator in the retinex theory of color vision. *Proceedings of the national academy of sciences*, 83(10):3078–3080, 1986.

L. Leem, H. Cho, J. Bau, Q.A. Jacobson, and S. Mitra. ERSA: Error Resilient System Architecture for Probabilistic Applications. In *Proc. Design Automation and Test in Europe*, 2010.

Ding Li, Angelica Huyen Tran, and William G. J. Halfond. Making web applications more energy efficient for oled smartphones. In *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, pages 527–538, New York, NY, USA, 2014. ACM.

Ding Li, Angelica Huyen Tran, and William G. J. Halfond. Nyx: A display energy optimizer for mobile web apps. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2015, pages 958–961, New York, NY, USA, 2015. ACM.

Man-Lap Li, Pradeep Ramachandran, Swarup Kumar Sahoo, Sarita V Adve, Vikram S Adve, and Yuanyuan Zhou. Understanding the propagation of hard errors to software and implications for resilient system design. In *ACM SIGARCH Computer Architecture News*, volume 36, pages 265–276. ACM, 2008.

Xiaodong Li, Sarita V Adve, Pradip Bose, Jude Rivers, et al. Softarch: an architecture-level tool for modeling and analyzing soft errors. In *International Conference on Dependable Systems and Networks (DSN 2005)*, pages 496–505. IEEE, 2005.

Xuanhua Li and Donald Yeung. Application-level correctness and its impact on fault tolerance. In *Proceedings of the 13th International Symposium on High Performance Computer Architecture*, pages 181–192, 2007.

Michael D. Linderman, Matthew Ho, David L. Dill, Teresa H. Meng, and Garry P. Nolan. Towards program optimization through automated analysis of numerical precision. In *Proceedings of the 8th Annual IEEE/ACM International Symposium on Code Generation and Optimization*, CGO '10, pages 230–237. ACM, 2010.

Avinash Lingamneni, Kirthi Krishna Muntimadugu, Christian Enz, Richard M. Karp, Krishna V. Palem, and Christian Piguet. Algorithmic methodologies for ultra-efficient inexact architectures for sustaining technology scaling. In *Proceedings of the 9th Conference on Computing Frontiers*, CF '12, pages 3–12, New York, NY, USA, 2012. ACM. .

Avinash Lingamneni, Christian Enz, Krishna Palem, and Christian Piguet. Synthesizing parsimonious inexact circuits through probabilistic design techniques. *ACM Transactions on Embedded Computing Systems (TECS)*, 12(2s):93, 2013.

J. W. S. Liu, K. J. Lin, W. K. Shih, A. C. Yu, J. Y. Chung, and W. Zhao. Algorithms for scheduling imprecise computations. *Computer*, 24(5):58–68, 1991.

J. W. S. Liu, Wei-Kuan Shih, Kwei-Jay Lin, R. Bettati, and Jen-Yao Chung. Imprecise Computations. *Proceedings of the IEEE*, 82(1):83–94, January 1994.

S. Liu, K. Pattabiraman, T. Moscibroda, and B. G. Zorn. Flicker: Saving Refresh-Power in Mobile Devices through Critical Data Partitioning. Technical Report MSR-TR-2009-138, Microsoft Research, October 2009.

M. O. Rabin. Probabilistic Algorithms. In *Algorithms and Complexity*, pages 21 – 40, New York, NY, USA, 1976. Academic Press.

S. Mandal and R. Sarpeshkar. Circuit models of stochastic genetic networks. In *Biomedical Circuits and Systems Conference, 2009. BioCAS 2009. IEEE*, pages 109–112, Nov 2009.

H. Bo Marr and Jennifer Hasler. Compiling probabilistic, bio-inspired circuits on a field programmable analog array. *Frontiers in Neuroscience*, 8(86), 2014.

T. C. May and M. H. Woods. Alpha-particle-induced soft errors in dynamic memories. *IEEE Trans. Elect. Dev.*, 26:2, 1979.

M. Mehrara, M. Attariyan, S. Shyam, K. Constantinides, V. Bertacco, and T. Austin. Low-cost protection for SER upsets and silicon defects. In *Design, Automation & Test in Europe Conference & Exhibition, 2007. DATE'07*, pages 1–6, 2007.

Jiayuan Meng, Srimat Chakradhar, and Anand Raghunathan. Best-effort parallel execution framework for recognition and mining applications. In *Parallel and Distributed Processing Symposium, International*, pages 1–12, Los Alamitos, CA, USA, 2009. IEEE Computer Society.

Fabrice Mérillon, Laurent Réveillère, Charles Consel, Renaud Marlet, and Gilles Muller. Devil: An idl for hardware programming. In *Proceedings of the 4th Conference on Symposium on Operating System Design & Implementation - Volume 4*, OSDI'00, pages 2–2. USENIX Association, 2000.

Sasa Misailovic, Michael Carbin, Sara Achour, Zichao Qi, and Martin C Rinard. Chisel: reliability-and accuracy-aware optimization of approximate computational kernels. In *Proceedings of the 2014 ACM International Conference on Object Oriented Programming Systems Languages & Applications*, pages 309–328. ACM, 2014.

Natasa Miskov-Zivanov and Diana Marculescu. Mars-c: modeling and reduction of soft errors in combinational circuits. In *DAC '06: Proceedings of the 43rd annual conference on Design automation*, pages 767–772, New York, NY, USA, 2006. ACM Press.

Radhika Mittal, Aman Kansal, and Ranveer Chandra. Empowering developers to estimate app energy consumption. In *Proceedings of the 18th Annual International Conference on Mobile Computing and Networking*, Mobicom '12, pages 317–328, New York, NY, USA, 2012. ACM.

Michael Mitzenmacher and Eli Upfal. *Probability and computing: Randomized algorithms and probabilistic analysis*. Cambridge University Press, 2005.

Debabrata Mohapatra, Georgios Karakonstantis, and Kaushik Roy. Significance driven computation: a voltage-scalable, variation-aware, quality-tuning motion estimator. In *ISLPED '09: Proceedings of the 14th ACM/IEEE international symposium on Low power electronics and design*, pages 195–200, New York, NY, USA, 2009. ACM.

Rajeev Motwani and Prabhakar Raghavan. *Randomized algorithms*. Chapman & Hall/CRC, 2010.

Michael Nicolaidis and Damien Chardonnereau. Soft-Error Testing: Key Points. *Computer*, 38(2):44 (sidebar), 2005.

NXP Semiconductors. UM10204, *I2C-bus specification and user manual*, April 2014.

Nahmsuk Oh, Subhasish Mitra, and Edward J. McCluskey. ED4I: Error detection by diverse data and duplicated instructions. *IEEE Trans. Computers*, 51(2):180–199, 2002a.

Nahmsuk Oh, Philip Shirvani, and Edward J. McCluskey. Error detection by duplicated instructions in super-scalar processors. *IEEE Transactions on Reliability*, 51(1):63–75, March 2002b.

Krishna V. Palem. Energy aware computing through probabilistic switching: A study of limits. *IEEE Trans. Comput.*, 54:1123–1137, September 2005.

Pavel Panchekha, Alex Sanchez-Stern, James R. Wilcox, and Zachary Tatlock. Automatically improving accuracy for floating point expressions. In *Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI 2015, pages 1–11, New York, NY, USA, 2015. ACM.

Sudeep Pasricha, Manev Luthra, Shivajit Mohapatra, Nikil Dutt, and Nalini Venkatasubramanian. Dynamic backlight adaptation for low-power handheld devices. *IEEE design & test of computers*, (5):398–405, 2004.

Karthik Pattabiraman, Vinod Grover, and Benjamin G. Zorn. Samurai: protecting critical data in unsafe languages. In *Proceedings of the 3rd ACM SIGOPS/EuroSys European Conference on Computer Systems 2008*, Eurosys '08, pages 219–232, New York, NY, USA, 2008. ACM.

Amir Pnueli. On the extremely fair treatment of probabilistic algorithms. In *STOC '83: Proceedings of the fifteenth annual ACM symposium on Theory of computing*, pages 278–290, New York, NY, USA, 1983. ACM Press.

Parthasarathy Ranganathan, Erik Geelhoed, Meera Manahan, and Ken Nicholas. Energy-aware user interfaces and energy-adaptive displays. *Computer*, 39(3):31–38, March 2006.

Josyula R. Rao. Reasoning about probabilistic parallel programs. *ACM Trans. Program. Lang. Syst.*, 16(3):798–842, 1994.

Joydeep Ray, James C. Hoe, and Babak Falsafi. Dual use of superscalar datapath for transient-fault detection and recovery. In *Proceedings of the 34th annual ACM/IEEE international symposium on Microarchitecture*, pages 214–224. IEEE Computer Society, 2001.

John H. Reif. Logics for probabilistic programming (extended abstract). In *STOC '80: Proceedings of the twelfth annual ACM symposium on Theory of computing*, pages 8–13, New York, NY, USA, 1980. ACM Press.

George A. Reis, Jonathan Chang, Neil Vachharajani, Ram Rangan, and David I. August. Swift: Software implemented fault tolerance. In *CGO '05: Proceedings of the international symposium on Code generation and optimization*, pages 243–254, Washington, DC, USA, 2005a. IEEE Computer Society.

George A. Reis, Jonathan Chang, Neil Vachharajani, Ram Rangan, David I. August, and Shubhendu S. Mukherjee. Design and evaluation of hybrid fault-detection systems. In *ISCA '05: Proceedings of the 32nd Annual International Symposium on Computer Architecture*, pages 148–159, Washington, DC, USA, 2005b. IEEE Computer Society.

George A. Reis, Jonathan Chang, Neil Vachharajani, Ram Rangan, David I. August, and Shubhendu S. Mukherjee. Software-controlled fault tolerance. *ACM Trans. Archit. Code Optim.*, 2:366–396, December 2005c.

EL Rhod, CA Lisbôa, and L. Carro. A low-SER efficient core processor architecture for future technologies. In *Design, Automation & Test in Europe Conference & Exhibition, 2007. DATE'07*, pages 1–6, 2007.

S. V. Rice, H. Bunke, and T. A. Nartker. Classes of cost functions for string edit distance. *Algorithmica*, 18(2):271–280, 1997.

Martin Rinard. Probabilistic accuracy bounds for fault-tolerant computations that discard tasks. In *Proceedings of the 20th annual international conference on Supercomputing*, ICS '06, pages 324–334, New York, NY, USA, 2006. ACM.

Martin Rinard, Henry Hoffmann, Sasa Misailovic, and Stelios Sidiroglou. Patterns and statistical analysis for understanding reduced resource computing. In *Proceedings of the ACM international conference on Object oriented programming systems languages and applications*, OOPSLA '10, pages 806–821, New York, NY, USA, 2010. ACM.

Leonid Ryzhyk, Adam Walker, John Keys, Alexander Legg, Arun Raghunath, Michael Stumm, and Mona Vij. User-guided device driver synthesis. In *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation*, OSDI'14, pages 661–676. USENIX Association, 2014.

Giacinto Paolo Saggese and Anoop Vetteth. Microprocessor sensitivity to failures: Control vs execution and combinational vs sequential logic. In *DSN '05: Proceedings of the 2005 International Conference on Dependable Systems and Networks (DSN'05)*, pages 760–769, Washington, DC, USA, 2005. IEEE Computer Society.

D. Salesin, J Stolfi, and L. Guibas. Epsilon geometry: building robust algorithms from imprecise computations. In *SCG '89: Proceedings of the fifth annual symposium on Computational geometry*, pages 208–217, New York, NY, USA, 1989. ACM.

Mehrzad Samadi, Janghaeng Lee, D. Anoushe Jamshidi, Amir Hormati, and Scott Mahlke. Sage: Self-tuning approximation for graphics engines. In *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-46, pages 13–24, New York, NY, USA, 2013. ACM.

Adrian Sampson, Werner Dietl, Emily Fortuna, Danushen Gnanapragasam, Luis Ceze, and Dan Grossman. Enerj: Approximate data types for safe and general low-power computation. In *Proceedings of the 32Nd ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '11, pages 164–174, New York, NY, USA, 2011. ACM.

Adrian Sampson, Jacob Nelson, Karin Strauss, and Luis Ceze. Approximate storage in solid-state memories. In *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-46, pages 25–36, New York, NY, USA, 2013. ACM.

John Sartori and Rakesh Kumar. Exploiting timing error resilience in processor architecture. *ACM Trans. Embed. Comput. Syst.*, 12(2s):89:1–89:25, May 2013.

Matthew Schuchhardt, Susmit Jha, Raid Ayoub, Michael Kishinevsky, and Gokhan Memik. Optimizing mobile display brightness by leveraging human visual perception. In *Proceedings of the 2015 International Conference on Compilers, Architecture and Synthesis for Embedded Systems*, CASES '15, pages 11–20, Piscataway, NJ, USA, 2015. IEEE Press.

Naresh R. Shanbhag. Reliable and energy-efficient digital signal processing. In *Proceedings of the 39th Annual Design Automation Conference*, DAC '02, pages 830–835, New York, NY, USA, 2002. ACM.

Naresh R. Shanbhag, Rami A. Abdallah, Rakesh Kumar, and Douglas L. Jones. Stochastic Computation. In *Proceedings of the 47th Design Automation Conference*, pages 859–864. ACM, 2010.

Claude E. Shannon. Coding theorems for a discrete source with a fidelity criterion. *IRE National Convention Record*, 7(4):142–163, 1959.

Claude E. Shannon and Warren Weaver. *The Mathematical Theory of Communication*. University of Illinois Press, Urbana, Illinois, 1963.

W. K. Shih and J. W. S. Liu. Algorithms for scheduling imprecise computations with timing constraints to minimize maximum error. *IEEE Transactions on Computers*, 44(3):466–471, 1995.

Donghwa Shin, Younghyun Kim, Naehyuck Chang, and Massoud Pedram. Dynamic voltage scaling of oled displays. In *Proceedings of the 48th Design Automation Conference*, DAC '11, pages 53–58, New York, NY, USA, 2011. ACM.

Stelios Sidiroglou-Douskos, Sasa Misailovic, Henry Hoffmann, and Martin Rinard. Managing performance vs. accuracy trade-offs with loop perforation. In *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*, ESEC/FSE '11, pages 124–134, New York, NY, USA, 2011. ACM.

D. P. Siewiorek and R. S. Swarz. *Reliable Computer Systems, Design and Evaluation*. Digital Press, 2nd edition, 1992.

T. J. Slegel, R. M. Averill III, M. A. Check, B. C. Giamei, B. W. Krumm, C. A. Krygowski, W. H. Li, J. S. Liptay, J. D. MacDougall, T. J. McPherson, J. A. Navarro, E. M. Schwarz, K. Shum, and C. F. Webb. IBM's S/390 G5 Microprocessor design. *IEEE Micro*, 19:12–23, March 1999.

R. Smith. An Overview of the Tesseract OCR Engine. *ICDAR*, 7(1):629–633, 2007.

Jayanth Srinivasan, Sarita V Adve, Pradip Bose, and Jude A Rivers. The case for lifetime reliability-aware microprocessors. In *ACM SIGARCH Computer Architecture News*, volume 32, page 276. IEEE Computer Society, 2004.

ST Microelectronics. *L3G4200D MEMS Motion Sensor: Ultra-stable Three-axis Digital Output Gyroscope*, Data Sheet, December 2010.

ST Microelectronics. *LPS25H MEMS Pressure Sensor: 260–1260 hPa Absolute Digital Output Barometer*, Data Sheet, January 2014.

Phillip Stanley-Marbell. Sal/svm: an assembly language and virtual machine for computing with non-enumerated sets. In *Virtual Machines and Intermediate Languages*, VMIL '10, pages 1:1–1:10, New York, NY, USA, 2010. ACM.

Phillip Stanley-Marbell and Diana Marculescu. A Programming Model and Language Implementation for Concurrent Failure-Prone Hardware. In *Proceedings of the 2nd Workshop on Programming Models for Ubiquitous Parallelism, PMUP '06*, September 2006.

Phillip Stanley-Marbell and Martin Rinard. Lax: Driver interfaces for approximate sensor device access. In *15th Workshop on Hot Topics in Operating Systems (HotOS XV)*, Kartause Ittingen, Switzerland, May 2015a. USENIX Association.

Phillip Stanley-Marbell and Martin Rinard. Value-deviation-bounded serial data encoding for energy-efficient approximate communication. Technical Report MIT-CSAIL-TR-2015-022, MIT Computer Science and Artificial Intelligence Laboratory (CSAIL), June 2015b.

Phillip Stanley-Marbell and Martin Rinard. Efficiency limits for value-deviation-bounded approximate communication. *IEEE Embedded Systems Letters*, 7(4): 109–112, 2015c.

Phillip Stanley-Marbell and Martin Rinard. Reducing serial i/o power in error-tolerant applications by efficient lossy encoding. In *Proceedings of the 53rd Annual Design Automation Conference*, DAC '16, pages 62:1–62:6, New York, NY, USA, 2016. ACM. .

Phillip Stanley-Marbell, Victoria Caparros, and Ronald Luijten. Pinned to the walls: Impact of packaging and application properties on the memory and power walls. In *Proceedings of the 17th IEEE/ACM International Symposium on Low-power Electronics and Design*, ISLPED '11, pages 51–56, 2011.

Phillip Stanley-Marbell, Virginia Estellers, and Martin Rinard. Crayon: Saving power through shape and color approximation on next-generation displays. In *Proceedings of the Eleventh European Conference on Computer Systems*, EuroSys '16, pages 11:1–11:17, New York, NY, USA, 2016. ACM. .

Eugene W. Stark and Scott A. Smolka. A Complete Axiom System for Finite-State Probabilistic Processes. *Proof, Language and interaction — Essays in honour of Robin Milner*, pages 571–595, 2000.

Mark Stephenson, Jonathan Babb, and Saman Amarasinghe. Bitwidth analysis with application to silicon compilation. In *Proceedings of the ACM SIGPLAN 2000 conference on Programming language design and implementation*, PLDI '00, pages 108–120, New York, NY, USA, 2000. ACM.

Jun Sun, Wanghong Yuan, Mahesh Kallahalla, and Nayeem Islam. Hail: A language for easy and correct device access. In *Proceedings of the 5th ACM International Conference on Embedded Software*, EMSOFT '05, pages 1–9. ACM, 2005.

Karthik Sundaramoorthy, Zach Purser, and Eric Rotenburg. Slipstream processors: improving both performance and fault tolerance. In *Proceedings of the ninth international conference on Architectural support for programming languages and operating systems*, pages 257–268. ACM Press, 2000.

Timmy Sundström, Boris Murmann, and Christer Svensson. Power dissipation bounds for high-speed nyquist analog-to-digital converters. *Circuits and Systems I: Regular Papers, IEEE Transactions on*, 56(3):509–518, 2009.

A. Taber and E. Normand. *IEEE Trans. Nucl. Sci.*, 40:120, 1993.

Kiat Wee Tan, Tadashi Okoshi, Archan Misra, and Rajesh Krishna Balan. Focus: A usable & effective approach to oled display power management. In *Proceedings of the 2013 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, UbiComp '13, pages 573–582, New York, NY, USA, 2013. ACM.

Texas Instruments. *CC256x Bluetooth® and Dual-Mode Controller*, Data Sheet, January 2014a.

Texas Instruments. *HDC1000 Low Power, High Accuracy Digital Humidity Sensor with Temperature Sensor*, Data Sheet, Nov 2014b.

Texas Instruments. *TMP006/B Infrared Thermopile Sensor in Chip-Scale Package*, Data Sheet, November 2014c.

Texas Instruments. *TPS8267x 600-mA, High-Efficiency MicroSIP™ Step-Down Converter*, Data Sheet, October 2014d.

Texas Instruments. *TPS82740x 360nA IQ MicroSIP™ Step Down Converter Module for Low Power Applications*, Data Sheet, June 2014e.

T. N. Theis and P. M. Solomon. In quest of the “next switch”: Prospects for greatly reduced power dissipation in a successor to the silicon field-effect transistor. *Proceedings of the IEEE*, 98(12):2005 –2014, Dec 2010.

Jonathan Ying Fai Tong, David Nagle, and Rob. A. Rutenbar. Reducing power by optimizing the necessary precision/range of floating-point arithmetic. *IEEE Trans. Very Large Scale Integr. Syst.*, 8:273–285, June 2000.

G. V. Varatkar, S. Narayanan, N. R. Shanbhag, and D. L. Jones. Stochastic networked computation. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, PP(99):1–1, October 2009.

Benjamin Vigoda. *Analog logic: Continuous-Time analog circuits for statistical signal processing*. PhD thesis, Massachusetts Institute of Technology, 2003.

Benjamin Vigoda, David Reynolds, Jeffrey Bernstein, Theophane Weber, and Bill Bradley. Low power logic for statistical inference. In *Proceedings of the 16th ACM/IEEE international symposium on Low power electronics and design*, ISLPED ’10, pages 349–354, New York, NY, USA, 2010. ACM.

John von Neumann. Probabilistic logics and the synthesis of reliable organisms from unreliable components. *Automata Studies*, pages 43–98, 1956.

John von Neumann and Ray Kurzweil. *The computer and the brain*. Yale University Press, 2012.

I. Wagner and V. Bertacco. Engineering trust with semantic guardians. In *2007 Design, Automation & Test in Europe Conference & Exhibition*, page 140. IEEE, 2007.

David Walker, Lester Mackey, Jay Ligatti, George A. Reis, and David I. August. Static typing for a faulty lambda calculus. In *Proceedings of the eleventh ACM SIGPLAN international conference on Functional programming*, ICFP ’06, pages 38–49, New York, NY, USA, 2006. ACM.

Gregory K Wallace. The jpeg still picture compression standard. *Communications of the ACM*, 34(4):30–44, 1991.

Ji Wang, Xiao Lin, and Chris North. GreenVis : Energy-Saving Color Schemes for Sequential Data Visualization on OLED Displays. 2012.

Chris Weaver and Todd M. Austin. A fault tolerant approach to microprocessor design. In *Proceedings of the 2001 International Conference on Dependable Systems and Networks (formerly: FTCS)*, pages 411–420. IEEE Computer Society, 2001.

Herbert S Wilf and Albert Nijenhuis. *Combinatorial algorithms: an update*. SIAM, 1989.

Niklaus Wirth. What can we do about the unnecessary diversity of notation for syntactic definitions? *Commun. ACM*, 20(11):822–823, November 1977.

Shirley Wong, Simon Lucas, Alex Panaretos, Luis Velazquez, Robert Young, and Anthony Tang. Robust word recognition dataset. In *ICDAR*, 2003.

Vicky Wong and Mark Horowitz. Soft error resilience of probabilistic inference applications. In *In Proceedings of the Workshop on System Effects of Logic Soft Errors*, 2006.

Gunther Wyszecki and WS Stiles. *Color Science: Concepts and Methods, Quantitative Data and Formulae*. Wiley-Interscience, New York, 2000.

YZ Xu, H. Puchner, A. Chatila, O. Pohland, B. Bruggeman, B. Jin, D. Radaelli, and S. Daniel. Process impact on SRAM Alpha-particle SEU performance. In *2004 IEEE International Reliability Physics Symposium Proceedings, 2004. 42nd Annual*, pages 294–299, 2004.

Mengying Zhao, Yiran Chen, Xiang Chen, and Chun Jason Xue. Online oled dynamic voltage scaling for video streaming applications on mobile devices. *SIGBED Rev.*, 10(2):18–18, July 2013.

Neil Zhao. Full-Featured Pedometer Design Realized with 3-Axis Digital Accelerometer. *Analog Dialogue*, 44(06), June 2010.

James F. Ziegler and William A. Lanford. Effect of cosmic rays on computer memories. *Science*, 206(4420):776–788, 1979.

James F. Ziegler and William A. Lanford. The effect of sea level cosmic rays on electronic devices. *Journal of applied physics*, 52(6):4305–4312, 1981.

James F. Ziegler, Huntington W. Curtis, Hans P. Muhlfeld, Charles J. Montrose, B. Chin, Michael Nicewicz, C. A. Russell, Wen Y. Wang, Leo B. Freeman, P. Hosier, et al. Ibm experiments in soft fails in computer electronics (1978–1994). *IBM journal of research and development*, 40(1):3–18, 1996.