Appearing in *IEEE Symposium on Field-Programmable Custom Computing Machines* (FCCM), May 11–13, 2014

Energy Reduction through Differential Reliability and Lightweight Checking

Edin Kadric, Kunal Mahajan, and André DeHon
Dept. of Electrical and Systems Engineering
University of Pennsylvania
Philadelphia, PA, USA
Email: ekadric@seas.upenn.edu

***Abstract*—As technology feature sizes shrink, aggressive voltage scaling is required to contain power density. However, this also increases the rate of transient upsets—potentially preventing us from scaling down voltage and possibly even requiring voltage increases to maintain reliability. Duplication with checking and triple-modular redundancy are traditional approaches to combat transient errors, but spending 2–3× the energy for redundant computation can diminish or reverse the benefits of voltage scaling. As an alternative, we explore the opportunity to use checking computations that are cheaper than the base computation they are guarding. We identify and evaluate the effectiveness of lightweight checks in a broad set of common FPGA tasks in scientific computing and signal and image processing. We find that the lightweight checks cost less than 14% of the base computation. Using an exponential model for the relationship between voltage and transient upset rate, we are able to show over 80% net energy reduction by aggressive voltage scaling without compromising reliability compared to operation at the nominal voltage.**

I. Introduction

As we shrink technology feature sizes, we reduce the capacitance, C, on wires and gates. This reduces delay and energy, but also reduces the critical charge ($Q = CV$) holding state on the nodes, making it easier for the node to be upset. From an energy standpoint, we would like to reduce voltage to reduce energy, but this, too, decreases the critical charge. Therefore, to preserve the reliability of operations, we may be forced to increase voltage.

Operating power density limits coupled with limited voltage scaling have already put us in an energy-dominated regime. We can now place more transistors on an integrated circuit die than we can afford to switch [1], a phenomenon known as dark silicon [2]. As a result, energy and reliability requirements are at odds, and together, may limit our exploitation of scaled technology.

Can we tolerate higher rates of upsets in our designs to avoid this impasse? That is, if our designs are tolerant to the upset rates that come with reduced voltage operation, we could continue to reduce voltage as required by power density concerns, extracting more computation from our energy-limited integrated circuits. However, the design changes we make must not consume more energy than we save by voltage reduction. For example, triple modular redundancy (TMR) performs computations three times, thereby increasing the energy by a factor of 3. Since energy scales as CV^2, we must achieve voltage reductions of over $\sqrt{3}$ in order to achieve any benefits, and even when there is a benefit, it is limited by the TMR overhead.

As an alternative, we explore the use of application-specific checks that are less expensive than the computation itself—*lightweight checks* (LWC). We exploit the fact that, for many computations, it is asymptotically and absolutely cheaper to check that a proposed answer for a computational task is correct than it is to perform the computation (Sec. II-B). Furthermore, we exploit the fact that not all computations must operate with the same reliability—*differential reliability* (Sec. II-A). When we have operations that are amenable to LWCs, these can run at lower voltages, while the supervisory circuits that assure that the checks are performed operate at higher voltages and hence reliabilities. To the extent we can make the supervisory circuits a small fraction of the actual computation, *we can achieve the reliability of the high-energy reliable circuits with the energy efficiency of the low-energy, error-prone computation*.

Our novel contributions include:

- Identifying a framework for exploiting LWCs (Sec. IV)
- Identifying a set of LWCs for common FPGA kernels (Sec. V) and estimating their overhead (Sec. VII)
- Estimating the energy savings possible by exploiting these LWCs with differential reliability (Sec. VII)

This paper focuses on logic and interconnect voltage, errors, and energy, leaving the treatment of memories for future work. Hence we assume that the memories do not operate at the reduced voltages explored for logic. In a companion paper [3], we explore techniques to minimize memory energy.

II. Opportunities

We start by highlighting the key opportunities we explore.

A. Differential Reliability and Multiple Vdd

Differential reliability observes that we do not need the *same* reliability out of all the components of our design. In particular, we can often use more reliable components to oversee the computation of less reliable components. Error-correction on memories and checksums on packet data transmission are familiar and commonly used forms of differential reliability. In the memory case, the peripheral circuitry for error detection

TABLE I: Taxonomy of Reliability Problems

Challenge	Detect Immediate?	How?	Response	Sec.	This Work
Logic & Latch	Y	Concurrent Check	Rollback, Retry	III-A	focus
Configuration Upset	N	Checksum	Reload	III-C	
	Y	Concurrent Check	Reload, Rollback	III-C	detect
Aging	N	Offline Test	Remap to Avoid	III-E	
	Y	Concurrent Check	Remap, Rollback	III-E	detect
Manufacture	N/A	Offline Test	Map to Avoid	III-D	

and correction is typically of a larger, more reliable feature size than the memory core.

We use operating voltage to control the reliability of a circuit. We assume that we can run portions of the FPGA at different voltages, following a long history of multiple V_{dd} designs in the FPGA literature [4], [5]. Commercial FPGAs use multiple supply voltages for core and periphery and multiple bias voltages for operation [6].

B. Lightweight Checking (LWC)

Many computations have the property that it is cheaper to check the computation than to perform it. Square root and factoring are familiar examples, for which the check is a simple multiplication, requiring less work than identifying the square root or the factors. Furthermore, the NP-complete complexity class specifically identifies a family of computations that have small, polynomial time checks, but have, to date, no known polynomial time approach to computation. In this paper, we build our understanding and show that many important kernels and classes of computations do have checkers that are significantly smaller in practice (Sec. V).

Error correcting codes (ECC) can also be seen as LWCs, since they avoid the cost of replicating stored or transmitted data, instead using a small number of bits to detect errors in a much larger number of bits. ECCs efficiently mitigate high transient upset rates in memories, which is why this paper focuses on the complementary problem of mitigating transient upsets in computations.

III. Background

To clarify the goals of this work and put it into context, this section and Tab. I review common, small-feature-size reliability problems. We also review prior work on fault-tolerance in FPGAs.

A. Upset Phenomena and Voltage Scaling

This paper primarily addresses single-event logic and latch upsets (SEUs), which become increasingly important with scaled technology [7]. These may be caused by ionizing particles that disrupt the voltage on nodes causing wrong values in latches. SEUs may result directly from upsetting the stored state in a latch or from upsetting logic that is then sampled into a latch at a clock edge. They may also be caused by thermal fluctuations or shot noise [8].

B. Prior Work on Soft-Error Upsets and Timing Failures

Space and avionics applications have long had to deal with higher upset rates than ground-based systems, spawning a host of prior work on SEU tolerance in FPGAs. Our LWCs are an optimization over *duplication with compare* [9] since our checkers are small compared to the base computation. TMR has been the typical mitigation mechanism on FPGAs [10], [11]. However, this comes at a high energy overhead (>200%). When the application can accept errors in the output, previous work shows that this can be reduced by applying TMR selectively [12]. In contrast, our solution catches errors as they occur, before they corrupt the output, and are significantly more lightweight than TMR. Unlike TMR, our detection and correction scheme does impact the throughput of results and may not be suitable when there is no timing slack available for recomputation. Still, as we will see from the low rate of recomputation (Tab. II), the impact on aggregate throughput due to recomputation is small.

Lowering voltage can also increase error rate due to timing violations. The Razor latch [13], [14] addresses this issue inexpensively by detecting late changes in signal values but only catches violations in a narrow timing window and does not address the broader classes of SEUs.

C. Configuration Upsets

FPGAs are particularly sensitive to transient events that upset configuration bits. Nonetheless, (1) FPGA vendors already provide checksums and scrubbing logic to detect when configurations need to be reloaded [15], and (2) there is no need to aggressively scale down configuration voltages since they do not switch dynamically during operation and do not contribute to dynamic energy consumption. Keeping the voltage on configuration bits high is a form of differential reliability. A common recovery strategy is checkpoint and rollback [16].

Our LWCs can catch configuration upsets as they occur. In contrast, checksum and scrubbing schemes take millions of cycles to detect upsets, resulting in a large number of erroneous outputs. Our LWC scheme validates **every** output and detects errors immediately. If the error is an SEU (Sec. III-A), the error will not persist and a retry will most likely not see it again. If the error is a configuration upset or a lifetime aging failure (Sec. III-E), the retry will fail as well, indicating the need to reload the configuration or repair the logic [17]–[19].

D. Process Variation

High V_{th} variation in small feature size transistors could also prevent aggressive scaling of component operating voltages. That is, if our voltage scaling were limited by the worst-case V_{th} on a multi-billion transistor 22 nm, or smaller, device, we would have limited room to reduce the voltage. This is **not** the primary concern of this work since prior work [17] shows that it should be possible to avoid high variation transistors in FPGAs and operate down to 150 mV—much lower than the ITRS suggested operating point of 700 mV at 22 nm. Significantly, these variation-avoidance techniques

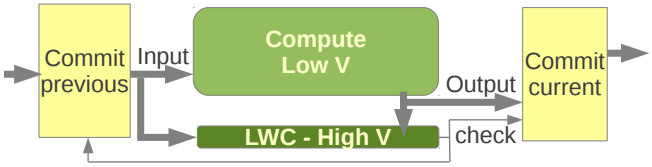


Fig. 1: Differential Reliability Structure

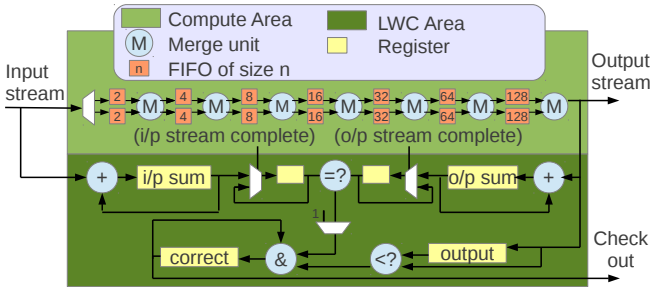


Fig. 2: Sort kernel with its LWC

allow us to operate at the well-defined minimum energy point. Nonetheless, [17] does not address the impact of low-voltage operation on transient faults, which is the primary concern of our work.

This post-fabrication, component-specific mapping allows FPGAs to operate at lower voltages than ASICs and is a key reason why FPGAs may have greater need to tolerate low voltage operation of small-feature size devices than ASICs. By combining the upset tolerance enabled by this work and the variation tolerance in [17], we could close some of the traditional energy gap between FPGAs and ASICs [20].

E. Aging

Small feature size devices are also susceptible to aging faults [21] that can result in permanent rather than transient circuit errors. As noted above (Sec. III-C), the LWCs we describe can also immediately detect those aging faults. If retry and configuration reload do not resolve the error, this is an indication that an aging error has occurred. This can serve as a trigger for repair mechanisms such as [18], [19].

IV. Lightweight Check Example

A. LWCs and Differential Reliability

LWCs are one way to exploit differential reliability. If a check can validate the correctness of a computation, then we can tolerate low reliability in the computation as long as the check reliably identifies when the computation is in error. Specifically, this means we can run the computation at low voltage, V_l, and the check at high voltage, V_h, as shown in Fig. 1. To the extent the check is cheaper than the computation, we have an opportunity to save energy.

Since the computation will have errors at some rate, we checkpoint its inputs and keep them until the LWC validates that the computation was correct and the outputs are themselves checkpointed. When the LWC sees an error, the erroneous output is discarded and recomputed from the checkpointed input. This scheme fits naturally into a streaming compute model (*e.g.*, [22], [23]).

B. Sort Algorithm and Implementation

As a motivating example, consider an $n = 128$ element streaming merge sort [24], where each element is composed of a single-precision floating-point number.

The merge-sort is implemented using $\log(n) = 7$ merge elements with appropriately sized buffers between them, as shown in Fig. 2. A new input is presented to the first unit on every cycle, and an output is produced on every cycle, with a latency of n. The i^{th} merge element performs an in-order merge on two ordered 2^i-element sequences in alternating 2^{i+1}-cycle phases. On every cycle, each merge element consumes the smallest of its inputs and sends it to the top output on phase-1 and the bottom output on phase-2.

C. Checking Logic

To check the sort, we confirm the outputs' order with a simple pairwise comparison (Fig. 2). We also confirm that no element was lost or modified by computing an integer checksum of all the inputs and confirming that it matches that of the outputs.

Performing the sort requires $O(n \log(n))$ work, whereas performing n comparisons and sums requires $O(n)$ work. The check is asymptotically and absolutely easier to perform than the computation, making it a *lightweight* check.

Details on our experiments are in Sec. VII. We find that the LWC's energy overhead is only 10%, that it can detect *all* faults due to single bit flips in the logic, and 99.955% of faults due to 2 simultaneous flips (Tab. II). Differential reliability and LWCs allow us to operate at 150mV to get 85% energy savings with no loss in reliability.

V. Set of Kernels with LWCs

The sort example shows that we can reduce energy by running the main computation on low-energy, error-prone circuitry. The key enabler was the LWC, which raises the question about how often such checks exist. In this section, we identify LWCs for common FPGA operations in the areas of scientific computing and signal and image processing.

A. Do Nothing

Before we review operations with LWCs, we note that many applications, particularly in signal processing, can increase the error rate while still maintaining proper operation. These applications already tolerate some Signal to Noise Ratio (SNR), such as compression, object matching and feature detection. Therefore, we often have a margin to decrease voltage and SNR, yet maintain an acceptable signal, image or sound quality, even with more errors at the output.

For example, consider using Gaussian Mixture Models (GMM) [25] to separate foreground objects from the background. The algorithm itself is inherently noisy. As a result, the downstream processing typically performs morphological operations to tolerate isolated pixels that are misidentified [26]. As long as the errors introduced due to SEUs are small compared to the noise inherent in the algorithm, the downstream operations can tolerate them along with the algorithm

noise. For the sake of evaluation, we assume that one pixel classification error per 1000 pixels processed is tolerable. This would result in an isolated foreground or background pixel that would be removed by the morphological operations.

B. Operations with Checksums

Many signal, image processing, and scientific computing tasks are based on linear weighted sums. As such, it is often possible to identify sums that remain invariant between the input data and the output data, or, at least, change in easily predictable ways. These sums serve as an LWC on the operation. As we have already shown, a sum was a useful part of guarding sort operations (Sec. IV-C).

1) Window Filtering: Window filtering operations compute each output pixel as the weighted sum of a number of neighboring pixels. Gaussian Filtering for edge detection is a common example of a window filter. Since each pixel in the original image contributes to the output image weighted by coefficients in the window mask, the total sum of the final image is the same as the sum of the original image times the weight of the window mask, provided we compute the edge pixels for the output image as well (assuming neighbors with pixel value of 0 when their value is not given).

Except for edging effects, the LWC only requires two additions for each pixel, one for the input sum and one for the output sum, while the window filter operation requires one multiplication and one addition per window coefficient for each pixel. For a 5×5 window filter, this is 25 multiplications and additions per pixel.

2) Matrix Multiplication: The $n \times n$ square matrix multiplication $A \times B = C$ requires n^3 additions and multiplications. As suggested in [27] and implemented for FPGAs in [28], an LWC for this operation is to compute a checksum row for A: $A_{n+1,j} = \sum_{i=1}^{n} A_{i,j}$, a checksum column for B: $B_{i,n+1} = \sum_{j=1}^{n} B_{i,j}$, and include them as part of the multiplication, resulting in a product matrix C with an extra row and column, whose common element, the bottom-right corner, is the sum of all the elements of C, thus the LWC:

$$C_{n+1,n+1} = \sum_{i=1}^{n} \sum_{j=1}^{n} C_{i,j}$$

The LWC requires $3n^2$ additions to compute the checksums and n additions and multiplications for the corner element $C_{n+1,n+1}$, which is $O(n^2)$ complexity. This is less work than computing $A \times B = C$, which has complexity $O(n^3)$.

3) FFT: The Discrete Fourier Transform (DFT) is a matrix-vector computation, and thus has an LWC similar to the one just described. However, the most straight-forward version does not protect against errors in the butterfly multiplications. A variant from [29] addresses this problem by encoding the inputs and outputs to assign each of them a non-trivial weight in the checksum. The result is that we simply need to perform a dot product on the input and output of the FFT and compare them. This means the LWC requires $O(n)$ operations, while the FFT requires $O(n \log(n))$. In fact, the energy overhead of the check using [29] is about $2/\log(n)$.

C. Convergent Algorithms

Iterative convergent algorithms are an important class of kernels in both image processing (*e.g.*, [30]) and scientific computing [31]. They often have the useful property that an LWC is built into the algorithm—the convergence acceptance test. Assuming we protect the acceptance check, it guarantees that no result is produced until correct. In an iterative improvement computation, the algorithm will typically self-correct when errors occur. This means, for correctness, we only need to focus on the convergence test regardless of the iterative improvement computation.

We use the Conjugate Gradient iterative solution (CGrad) to a system of linear equations $A\vec{x} = \vec{b}$ as an example. We check whether the residue of the current iteration is smaller than the one obtained from the previous iteration: $(\vec{r}_{k+1}^T \vec{r}_{k+1} < \vec{r}_k^T \vec{r}_k)$. If it is, then the monotonic convergence property of the conjugate gradient was preserved. Otherwise, we know that an error occurred during the last iteration, so we rollback to the previous state whose estimate was closer to the actual solution. Furthermore, CGrad might converge to a wrong value even if $(\vec{r}_{k+1}^T \vec{r}_{k+1} < \vec{r}_k^T \vec{r}_k)$ is respected at every step because $\vec{r}_k$ is updated based on its previous value $\vec{r}_{k-1}$, instead of using $\vec{r}_k = \vec{b} - A\vec{x}_k$ at every step (making the algorithm more efficient). We detect these cases by computing $A\vec{x}_k$ and confirming that $(A\vec{x}_k = \vec{b})$ after convergence is achieved. Since convergence takes $O(n)$ cycles for an $n \times n$ matrix A, the conjugate gradient algorithm requires $O(n^3)$ work, whereas the check is only an $O(n^2)$ operation.

D. Round-off Errors

When dealing with floating-point data, checking exact equality of the checksums does not work. Instead, to avoid false positives in the LWC output, we must consider a threshold σ that indicates whether two numbers are close enough together: whether they might be different only because of rounding. This threshold should be high enough to avoid false positives, but low enough to catch errors. We consider two floating-point numbers to be equal when their difference is within σ ULPs (Unit of Least Precision).

As suggested in [29] for FFTs, when the LWC catches an error, the recomputation should be done differently to avoid triggering the same false positive. For example, in the FFT case, we use a simple mechanism where the input checksum is computed as a running sum starting from the i^{th} input, where i is the number of times the computation was already performed with the same inputs. In our experiments, this always succeeded in identifying false positives, and it took an average of 15 recomputations to do so (if the error is not a false positive, only 1 recomputation is normally required). False positives being rare events, and the recomputation being lightweight, the overall cost associated with false positives is small as quantified in Sec.VII and Tab. II.

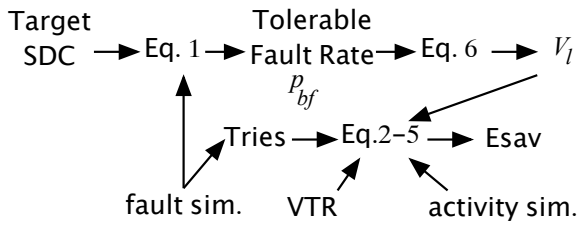


Fig. 3: Calculation Flow for Energy Savings

VI. Evaluation Methodology

To evaluate the effectiveness of differential reliability and LWCs, we need to understand:

- How reliably the check protects the kernel as a function of SEU rate
- How lightweight the check is compared to the base kernel computation
- How the SEU rate rises with voltage reduction

Fig. 3 shows how the pieces come together to estimate energy savings. We describe our kernels and LWCs in Bluespec SystemVerilog [32] and compile them to Verilog. The Verilog modules are then mapped with VTR (Verilog To Routing) [33]. The rest of this section describes the custom components of the Fig. 3 flow.

a) Reliability: To estimate reliability, we systematically inject all combinations of 1, 2, or 3 errors into a mapped, LUT-level simulation of the kernel and compare the outputs with an error-free version of the kernel. Each error is injected for one randomly chosen clock cycle at the net level (at LUT outputs), thus covering logic and routing transient errors. From these simulations, we determine which faults actually propagate to outputs, which are detected by the lightweight check, and which are not detected at the output. To compute the probability of Silent Data Corruption (SDC), an error that is not caught by our LWC, we compute:

$$p_{sdc} = \sum_{i=1}^{\infty} \binom{N}{i} (p_{bf})^i (1-p_{bf})^{N-i} p_{prop}(i) \times [p_{undet}(i)(1-p_{fd}) + (1-p_{undet}(i))\, p_{fd}] \quad (1)$$

Here p_{bf} is the probability of a LUT output being in error, i is the number of gates failing on a cycle, N is the total number of gate evaluations in the kernel, $p_{prop}(i)$ is the probability the error propagates to the output when there are i faults, p_{fd} is the probability of an error in the detector itself, and $p_{undet}(i)$ is the probability that the injection of i faults results in an error at the output that is not detected by the LWC. Since the probability of errors decreases exponentially with i, we approximate Eq. 1 by only performing the sum up to $i = 3$.

b) Energy: For relative energy estimations, we map our designs to 4-LUTs with one LUT per cluster. In FPGAs, it is well-known that interconnect is the dominant contribution to energy [34]. As such, we approximate the dynamic energy consumption of a net i as proportional to its activity α_i and segment count S_{c_i}. This gives the total energy of a circuit:

$$E_{dyn} = \beta C_{seg} V^2 \quad (2)$$

$$\beta = \sum_{nets\ i} \alpha_i S_{c_i} \quad (3)$$

C_{seg} is the capacitance of an interconnect segment. Our custom simulator computes each net's activity factor, α_i, and reads in segment counts, S_{c_i}, from VTR's output route file, allowing us to evaluate energy consumption. Since the ratios of energies do not depend on C_{seg}, the absolute value of C_{seg} is not important to the comparison. This is an abstract model, not for a specific FPGA; we believe that it is representative of FPGA energy. As a sanity check, we correlated the total computation and routing energy that Altera PowerPlay estimates for designs mapped to a Stratix IV to the R4 segment count for those designs and found a correlation coefficient of 98%. We compute β for both the kernel computation (β_{kernel}) and for its associated LWC (β_{lwc}).

The total energy for a protected computation is:

$$E_{total} = Tries \times (E_{kernel} + E_{lwc}) \quad (4)$$

The average number of tries is $\frac{1}{1-p_{rcmp}}$; p_{rcmp} is the probability that a failure is detected, requiring rollback and recomputation, which is also computed by our fault injection simulations. This means the total energy saved when operating the kernel at low voltage, V_l, and the checker at a high voltage, V_h, is:

$$E_{sav} = 1 - \frac{E_{total}}{E_{kernel}(V=V_h)} = 1 - \left[\left(\frac{V_l}{V_h}\right)^2 + \frac{\beta_{lwc}}{\beta_{comp}}\right]\left(\frac{1}{1-p_{rcmp}}\right) \quad (5)$$

As V_l tends to zero, E_{sav} is driven toward the β ratio until p_{rcmp} becomes large.

c) Reliability and Voltage: Both [35] and [36] suggest an exponential relationship between Soft Error Rate (SER) and voltage due to the minimum charge, Q_{crit}, that needs to be deposited by a particle strike to upset a node:

$$p_{bf_l}(V) = p_{bf_h} 10^{c_0(V_h - V_l)} \quad (6)$$

p_{bf_l} is the bit flip rate of a node at voltage V_l, p_{bf_h} is the bit flip rate at nominal voltage ($V_h = V_{dd}$), and c_0 is a technology-dependent constant. To date, these values have largely been determined by physical fault injection experiments. The exact value of c_0 does not affect the core ideas and basic trends presented in this paper, but will impact the absolute benefits. In this paper, we show our greatest benefits for any $c_0 \leq 18$. For reference, we can estimate $c_0 = 3$ at 90 nm from [37].

[38] predicts and [39] experimentally confirms a FIT rate (Failures In Time = number of failures in one Billion hours of operation) (p_{bf_h}) of about 4.4×10^{-5} for an inverter running at 1 GHz in the 90 nm technology. Furthermore, Fig. 5 in [38] suggests an order of magnitude increase in FIT rate when scaling feature size by a factor of 2, so about 4.4×10^{-3} at 22 nm. This means the probability of a bit flip, p_{bf_h}, on a net running at the nominal voltage (ITRS suggested V_{dd}), is on the order of $p_{bf_h} = \frac{4.4\times10^{-3}}{1\,\text{Billion hours}\times 1GHz} \approx 10^{-24}$. Nonetheless, the exact value of p_{bf_h} does not have a strong impact on the results. Our results on energy savings in Sec. VII are unchanged for any $p_{bf_h} \leq 10^{-19}$.

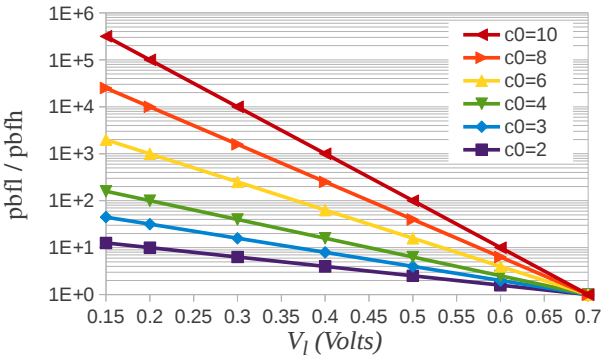


Fig. 4: Bit flip rate of logic at low voltage V_l to bit flip rate of logic at nominal voltage $V_h = 0.7$

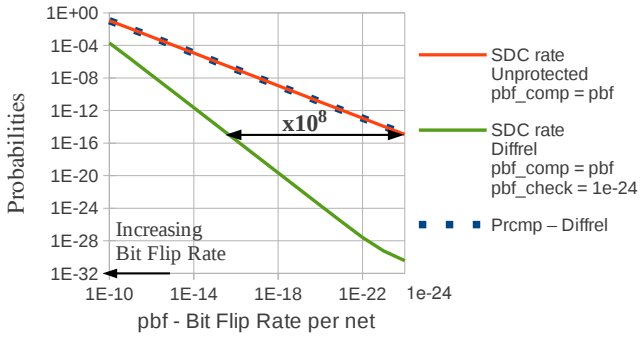


Fig. 5: Effect of changing the bit flip rate per net (p_{bf}) on overall reliability and rollback rate (p_{rcmp}) for WinF

VII. Results

A. Characteristics of the Kernels

We evaluate the impact of the LWCs described in Sec. V. Their characteristics are shown in Tab. II. Input data to the kernels is generated uniformly at random, and it is appropriately scaled to avoid saturation to infinity. Except for CGrad, the work performed per task is data independent.

How lightweight are the checks? Tab. II shows the LWC overheads for each kernel, the β_{ratio}, with values ranging between 0–14%. The β_{ratio} is a lower bound on the energy used by the LWC scheme—a low β_{ratio} ensures there is enough room for energy gains when voltage is scaled (Eq. 5). Tab. II also shows the raw reliability statistics for the kernels and LWCs including their rate of detection (p_{undet}) and rate of false positives (p_{f+t}).

B. Effects of the Different Parameters

1) Error Rate Scaling: What range of SERs do we expect to see for p_{bf}? Fig. 4 plots the exponential dependency of the bit flip rate p_{bf_l} on voltage according to Eq. 6 for different values of c_0. This shows a one to six order of magnitude increase in fault rate across our operational voltage range.

2) Error Rate Impact: What range of SER can our LWC mitigate? Fig. 5 shows how increasing the bit flip rate per net increases the overall error rate seen at the output of the WinF kernel—its SDC rate (similar results are obtained for the other kernels). Note that we show increasing bit flip rates towards the left since high error rates correlate with smaller voltages, which we show on the left of other graphs. This is shown when no LWC is used (Unprotected), and when the LWC is kept at a fixed bit flip rate of 10^{-24} (Diffrel). Also shown is the recomputation rate of Diffrel, which is very close to the SDC rate for the Unprotected case since most of the errors are caught and recomputed. Diffrel clearly shows the improvement in reliability over the Unprotected case. The line showing a $10^8\times$ difference in bit flip rate suggests that the Protected case could run in an environment with a bit flip rate 10^8 times higher, yet still provide the same overall reliability (SDC rate) as the Unprotected case with a bit flip rate of 10^{-24}. This is less than the $10^6\times$ increase in bit flip rate from Fig. 4.

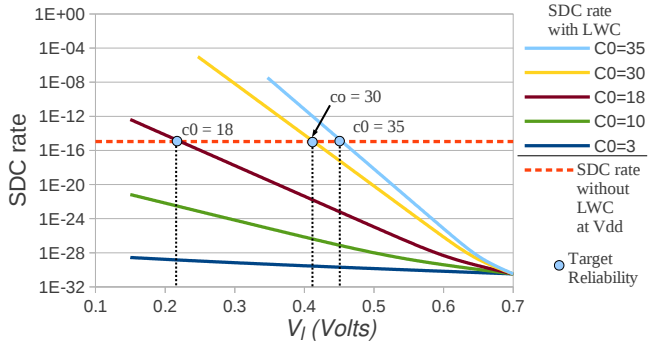
3) Energy Savings: How much energy do we actually save? Fig. 6a shows the exponential evolution of p_{sdc}, the error rate at the output of WinF, as a function of c_0, as voltage is varied. It highlights the point at which p_{sdc} crosses the SDC rate target obtained from running the unprotected kernel at V_{dd}. This determines the lowest operating voltage at which we can meet the reliability target. Fig. 6b shows how the associated voltage points turn into net energy savings. The main curve (c_0=2–15) stops at the 150 mV point (Sec. III-D), where most of the possible savings are achieved. As c_0 is increased, the same voltage point results into a higher upset rate, translating to a higher p_{rcmp}. Yet, as long as the upset rate is low enough to keep p_{rcmp} low (*e.g.*, less than 1%), E_{sav} is determined by the voltage and the β_{ratio} (*i.e.*, $E_{sav} \simeq 1 - \left[\left(\frac{V_l}{V_h}\right)^2 + \frac{\beta_{check}}{\beta_{comp}}\right]$ from Eq. 5). This is independent of c_0, which is why the curves shown in Fig. 6b overlap for every c_0 parameter when the voltages are high. However, each c_0 has a different point at which the energy savings curve reaches a maximum, then starts degrading, due to p_{rcmp} becoming dominant. Highlighted dots on the curves show the voltage points at which the LWC mitigated SDC rate, p_{sdc}, drops to the full voltage SDC rate (same points as shown in Fig. 6a), indicating the maximum energy savings achievable without degrading overall reliability compared to a design running at V_{dd} without LWCs.

In addition to high energy savings, low p_{rcmp} also ensures that the throughput overhead of the checking and recomputation operations is low. This is especially important in embedded applications with a limited tolerance for recomputation in order to support real-time processing.

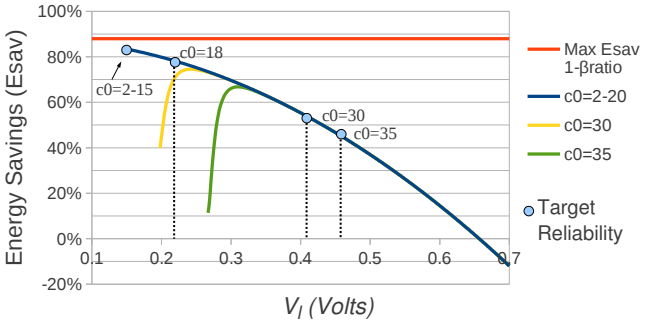
4) Sensitivity To c_0: How much do the results change as c_0 increases? Fig. 7 plots the optimum E_{sav} points (highlighted dots in Fig. 6) versus c_0 for all the kernels. This shows how the energy savings we can achieve vary with technology characteristics. While we are uncertain of the appropriate value of c_0 at 22 nm, we do not expect it to be an order of magnitude higher than $c_0 = 3$ that we saw for 90 nm. We include data at larger c_0 only to illustrate the evolution in behavior. *Fig. 7 shows net energy savings in excess of 80% for all kernels up to $c_0 = 15$.* The LWCs all achieve higher reliability (lower

TABLE II: Characteristics of the Different Kernels

Kernel	Properties	Data	Depth	Nets Comp	Nets LWC	Segment Count Comp	Segment Count LWC	Average α Comp	Average α LWC	β (energy) Comp	β (energy) LWC	β (energy) Ratio	p_{undet} (%) N_e=1	p_{undet} (%) N_e=2	p_{undet} (%) N_e=3	($c_0 = 3$) p_{f+_t}	($c_0 = 3$) p_{rcmp}
Sort	128 inputs	Float	128	6400	904	66183	5263	11.7%	15.4%	7748	811	**10%**	0	0.045	0.075	0	2.2e-8
CGrad	$A_{16\times16}$	Float	376	15850	4971	189888	59814	7.5%	1%	14200	588	**4.1%**	0	0	0	0	2.1e-3
FFT	32K-point	Double	240K	57254	63108	891492	916320	7.9%	1.1%	70122	10060	**14%**	0	0.016	0.022	3.8e-8	5.4e-7
WinF	$1K \times 1K$	FixP32	1024K	1103	278	6545	1341	24.4%	13.9%	1594	186	**12%**	0	3.31	0.46	0	2.6e-7
GMM	3 models	FixP32	1	6057	0	68109	0	22.6%	0%	15419	0	**0%**	-	-	-	0	0
MatMul-i	16×16	Int32	4K	1512	2285	14646	18629	21.2%	1.99%	3101	147	**4.7%**	0	1.38	0.29	0	2.7e-8
MatMul-s	16×16	Float	4K	4731	9486	60513	118277	7.2%	0.3%	4375	396	**9.1%**	0	0.071	0.474	5.9e-10	2.0e-8



(a) Error Rate Versus Voltage for Different c_0 Values for WinF



(b) Energy Savings Versus Voltage for Different c_0 Values for WinF

Fig. 6: Effects of Voltage Scaling for Different c_0 Values

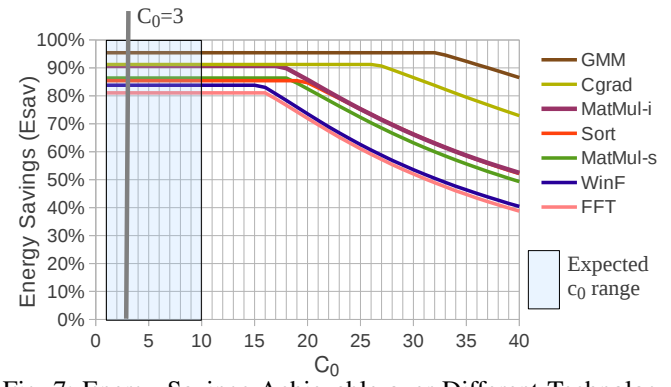


Fig. 7: Energy Savings Achievable over Different Technology Points (Different c_0)

SDC) than the target at low c_0 values. Nonetheless, they have different levels of protection, so we see them begin to drop from their maximum savings at different c_0 values. Those with the highest probability of undetected errors (*e.g.* WinF) and false positive rates (*e.g.*, FFT) drop first (See Tab. II), while those with no false positives and no undetected errors up to 3 errors injected (GMM and CGgrad) tolerate the highest c_0 values before their savings begins to drop.

VIII. Open Questions

As noted at the outset, this paper focused only on logic and interconnect energy. The full design also includes storage energy that can be reduced using small distributed memories as shown in [3] to prevent memory energy from dominating the total energy as we scale voltage down for the logic and interconnect. Applying the differential reliability and LWC technique shown here to the optimized kernels in [3], we get total savings, including memory, of 90% for GMM, 76% for WinF, and 78% for FFT (Sec.VII.D in [3]).

Furthermore, a full accounting must include any extra energy required to store and recover data that would not be incurred in a fault-free computation. Whether the data needs to be stored in the baseline case is highly dependent upon how the kernel is used in a larger computation.

When a kernel becomes large, a high N could make the probability of seeing an error, $1-(1-p_{bf_l})^N$, too high, to the point where it could drive the recomputation rate up and E_{sav} down. In this case, it may be beneficial to decompose the kernel into smaller subkernels with lower error probabilities for each: we compute checksums on smaller computations and rollback less often. Identifying the kernel sizes at which decomposition is necessary is left as future work.

Finally, we also leave as future work the exploration of the freedom we have in trading some of the achievable energy benefits for an **increase** in reliability over unprotected designs running at V_{dd}. This is useful when the SDC rate at nominal voltage is unacceptably high for a given application.

IX. Conclusions

Energy consumption is becoming the dominant concern in a large variety of computing systems at the same time that operational reliability is preventing aggressive voltage scaling. We show how FPGAs can use differential reliability, alongside component-specific mapping and configuration scrubbing, to reduce operating voltage and save over 80% energy in some common FPGA tasks without decreasing reliability, even when the relation between upset rate and voltage decrease is exponential. This provides an opportunity to narrow the traditional energy gap between FPGAs and ASICs. A key enabler was the identification of lightweight checks for a set of important FPGA kernels whose low logic overhead of 0–14% enables high energy savings.

X. Acknowledgments

This research was funded in part by DARPA/CMO contract HR0011-13-C-0005. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

References

[1] B. Nikolic, "Design in the power-limited scaling regime," *IEEE Trans. Electron Devices*, vol. 55, no. 1, pp. 71–83, January 2008. I

[2] H. Esmaeilzadeh, E. Blem, R. S. Amant, K. Sankaralingam, and D. Burger, "Dark silicon and the end of multicore scaling," in *ISCA*, 2011, pp. 365–376. [Online]. Available: http://doi.acm.org/10.1145/2000064.2000108 I

[3] E. Kadric, K. Mahajan, and A. DeHon, "Kung fu data energy—minimizing communication energy in FPGA computations," in *FCCM*, 2014. I, VIII

[4] Y. Hu, Y. Lin, L. He, and T. Tuan, "Physical synthesis for FPGA interconnect power reduction by dual-Vdd budgeting and retiming," *ACM Tr. Des. Auto. of Elec. Sys.*, vol. 13, no. 2, pp. 1–29, 2008. II-A

[5] J. Ryan and B. Calhoun, "A sub-threshold FPGA with low-swing dual-Vdd interconnect in 90nm CMOS," in *CICC*, September 2010, pp. 1–4. II-A

[6] D. Lewis, E. Ahmed, D. Cashman, T. Vanderhoek, C. Lane, A. Lee, and P. Pan, "Architectural enhancements in Stratix-III and Stratix-IV," in *FPGA*, 2009, pp. 33–42. [Online]. Available: http://doi.acm.org/10.1145/1508128.1508135 II-A

[7] V. Chandra and V. R. Aitken, "Impact of technology and voltage scaling on the soft error susceptibility in nanoscale CMOS," in *DFT*, ser. DFT '08, 2008, pp. 114–122. [Online]. Available: http://dx.doi.org/10.1109/DFT.2008.50 III-A

[8] J. Kim and L. Kish, "Error rate in current-controlled logic processors with shot noise," *Fluct. and Noise Let.*, vol. 4, no. 1, pp. 83–86, 2004. III-A

[9] J. Johnson, W. Howes, M. Wirthlin, D. McMurtrey, M. Caffrey, P. Graham, and K. Morgan, "Using duplication with compare for on-line error detection in FPGA-based designs," in *Proc. of IEEE Aerospace Conf.*, 2008, pp. 1–11. III-B

[10] C. Carmichael, *Triple Module Redundancy Design Techniques for Virtex FPGAs*, Xilinx, Inc., 2100 Logic Drive, San Jose, CA 95124, 2006, xAPP 197 <http://www.xilinx.com/bvdocs/appnotes/xapp197.pdf>. III-B

[11] N. Rollins, M. Wirthlin, P. Graham, and M. Caffrey, "Evaluating TMR techniques in the presence of single event upsets," in *Proc. MAPLD*, 2003. III-B

[12] B. Pratt, M. Caffrey, P. Graham, K. Morgan, and M. Wirthlin, "Improving FPGA design robustness with partial TMR," in *Proc. IEEE Intl. Rel. Phys. Symp.*, 2006, pp. 226–232. III-B

[13] T. Austin, D. Blaauw, T. Mudge, and K. Flautner, "Making typical silicon matter with Razor," *IEEE Computer*, vol. 37, no. 3, pp. 57–65, March 2004. III-B

[14] A. Brant, A. Abdelhadi, D. Sim, S. L. Tang, M. Yue, and G. Lemieux, "Safe overclocking of tightly coupled CGRAs and processor arrays using razor," in *FCCM*, 2013, pp. 37–44. III-B

[15] C. Carmichael and C.-W. Tseng, *Correcting Single-Event Upsets in Virtex-4 Configuration Memory*, Xilinx, Inc., 2100 Logic Drive, San Jose, CA 95124, 2009, xAPP 1008. [Online]. Available: http://www.xilinx.com/support/documentation/application_notes/xapp1088.pdf III-C

[16] G.-H. Asadi and M. B. Tahoori, "Soft error mitigation for SRAM-based FPGAs," in *Proc. VLSI Test Symp.*, 2005, pp. 207–212. III-C

[17] N. Mehta, R. Rubin, and A. DeHon, "Limit Study of Energy & Delay Benefits of Component-Specific Routing," in *FPGA*, 2012, pp. 97–106. III-C, III-D

[18] V. Lakamraju and R. Tessier, "Tolerating operational faults in cluster-based FPGAs," in *FPGA*, 2000, pp. 187–194. III-C, III-E

[19] R. Rubin and A. DeHon, "Choose-Your-Own-Adventure Routing: Lightweight Load-Time Defect Avoidance," *Transactions on Reconfigurable Technology and Systems*, vol. 4, no. 4, December 2011. III-C, III-E

[20] I. Kuon and J. Rose, "Measuring the gap between FPGAs and ASICs," *IEEE Trans. Computer-Aided Design*, vol. 26, no. 2, pp. 203–215, February 2007. III-D

[21] E. A. Stott, J. S. J. Wong, P. Pete Sedcole, and P. Y. K. Cheung, "Degradation in FPGAs: measurement and modelling," in *FPGA*, 2010, p. 229. III-E

[22] A. DeHon, Y. Markovsky, E. Caspi, M. Chu, R. Huang, S. Perissakis, L. Pozzi, J. Yeh, and J. Wawrzynek, "Stream computations organized for reconfigurable execution," *J. Microproc. and Microsys.*, vol. 30, no. 6, pp. 334–354, September 2006. IV-A

[23] M. Butts, A. Jones, and P. Wasson, "A structural object programming model, architecture, chip and tools for reconfigurable computing," in *FCCM*, 2007, pp. 55–64. IV-A

[24] D. Koch and J. Torresen, "FPGASort: A high performance sorting architecture exploiting run-time reconfiguration on FPGAs for large problem sorting," in *FPGA*, 2011, pp. 45–54. IV-B

[25] M. Genovese and E. Napoli, "ASIC and FPGA implementation of the gaussian mixture model algorithm for real-time segmentation of high definition video," *IEEE Trans. VLSI Syst.*, vol. 22, no. 3, pp. 537–547, March 2014. V-A

[26] C. Stauffer and W. E. L. Grimson, "Adaptive background mixture models for real-time tracking," in *Computer Vision and Pattern Recognition, 1999. IEEE Computer Society Conference on.*, vol. 2. Los Alamitos, CA, USA: IEEE, Aug. 1999, pp. 246–252 Vol. 2. [Online]. Available: http://dx.doi.org/10.1109/cvpr.1999.784637 V-A

[27] K.-H. Huang and J. Abraham, "Algorithm-based fault tolerance for matrix operations," *IEEE Trans. Comput.*, vol. C-33, no. 6, pp. 518–528, 1984. V-B2

[28] A. Jacobs, G. Cieslewski, and A. D. George, "Overhead and reliability analysis of algorithm-based fault tolerance in FPGA systems," in *FPL*, 2012. V-B2

[29] S.-J. Wang and N. K. Jha, "Algorithm-based fault tolerance for FFT networks," *IEEE Trans. Comput.*, vol. 43, no. 7, pp. 849–854, July 1994. V-B3, V-D

[30] S. Baker and I. Matthews, "Lucas-kanade 20 years on: A unifying framework," *International Journal of Computer Vision*, vol. 56, no. 3, pp. 221–255, Febuary 2004. V-C

[31] Y. Saad, *Iterative Methods for Sparse, Linear Systems*, 2nd ed. SIAM, 2003. V-C

[32] Bluespec, Inc., "Bluespec SystemVerilog 2012.01.A." [Online]. Available: http://www.bluespec.com VI

[33] J. Rose, J. Luu, C. W. Yu, O. Densmore, J. Goeders, A. Somerville, K. B. Kent, P. Jamieson, and J. Anderson, "The VTR project: architecture and CAD for FPGAs from verilog to routing," in *FPGA*. New York, NY, USA: ACM, 2012, pp. 77–86. VI

[34] T. Tuan, A. Rahman, S. Das, S. Trimberger, and S. Kao, "A 90-nm Low-Power FPGA for Battery-Powered applications," *IEEE Trans. Computer-Aided Design*, vol. 26, no. 2, pp. 296–300, 2007. VI-0b

[35] D. Zhu, R. Melhem, and D. Mossé, "The effects of energy management on reliability in real-time embedded systems," in *ICCAD*, 2004, pp. 35–40. VI-0c

[36] F. Firouzi, M. E. Salehi, F. Wang, and S. M. Fakhraie, "An accurate model for soft error rate estimation considering dynamic voltage and frequency scaling effects," *Micro. Rel.*, vol. 51, no. 2, pp. 460–467, 2011. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0026271410004804 VI-0c

[37] T. Heijmen, P. Roche, G. Gasiot, K. Forbes, and D. Giot, "A comprehensive study on the soft-error rate of flip-flops from 90-nm production libraries," *IEEE Trans. Device Mat. Rel.*, vol. 7, no. 1, pp. 84–96, 2007. VI-0c

[38] P. Shivakumar, M. Kistler, S. Keckler, D. Burger, and L. Alvisi, "Modeling the effect of technology trends on the soft error rate of combinational logic," in *Proc. Intl. Conf. Dependable Sys. and Nets*, 2002, pp. 389–398. VI-0c

[39] B. Narasimham, M. Gadlage, B. Bhuva, R. Schrimpf, L. Massengill, W. Holman, A. Witulski, R. Reed, R. Weller, and X. Zhu, "Characterization of neutron- and alpha-particle-induced transients leading to soft errors in 90-nm CMOS technology," *IEEE Trans. Device Mat. Rel.*, vol. 9, no. 2, pp. 325–333, 2009. VI-0c