Hindawi Publishing Corporation
Journal of Electrical and Computer Engineering
Volume 2012, Article ID 593532, 12 pages
doi:10.1155/2012/593532

Research Article

Selectively Fortifying Reconfigurable Computing Device to Achieve Higher Error Resilience

Mingjie Lin,[1] Yu Bai,[1] and John Wawrzynek[2]

[1] *Department of Electrical Engineering and Comouter Science, University of Central Florida, Orlando, 32816 FL, USA*
[2] *Department of Electrical Engineering and Comouter Science, University of California at Berkeley, Berkeley, 94720 CA, USA*

Correspondence should be addressed to Mingjie Lin, mingjie@eecs.ucf.edu

Received 11 September 2011; Revised 9 January 2012; Accepted 11 January 2012

Academic Editor: Deming Chen

With the advent of 10 nm CMOS devices and "exotic" nanodevices, the location and occurrence time of hardware defects and design faults become increasingly unpredictable, therefore posing severe challenges to existing techniques for error-resilient computing because most of them statically assign hardware redundancy and do not account for the error tolerance inherently existing in many mission-critical applications. This work proposes a novel approach to selectively fortifying a target reconfigurable computing device in order to achieve hardware-efficient error resilience for a specific target application. We intend to demonstrate that such error resilience can be significantly improved with effective hardware support. The major contributions of this work include (1) the development of a complete methodology to perform sensitivity and criticality analysis of hardware redundancy, (2) a novel problem formulation and an efficient heuristic methodology to selectively allocate hardware redundancy among a target design's key components in order to maximize its overall error resilience, and (3) an academic prototype of SFC computing device that illustrates a 4 times improvement of error resilience for a H.264 encoder implemented with an FPGA device.

1. Introduction

With the advent of 10 nm CMOS devices and "exotic" nanodevices [1–3], error resilience is becoming a major concern for many mission- and life-critical computing systems. Unfortunately, as these systems grow in both design size and implementation complexity, device reliability severely diminishes due to escalating thermal profiles, process-level variability, and harsh environments (such as outer space, high-altitude flight, nuclear reactors, or particle accelerators). Unlike design faults, device failures often are *spatially probabilistic* (the locations of hardware defects and design faults within a chip are unknown) and *temporally unpredictable* [4, 5] (the occurrence time of hardware defects are hard to foretell).

1.1. Related Work. Traditionally, either special circuit technique or conservative design is employed to ensure correct operation and to achieve high error resilience. Existing circuit techniques include guard banding, conservative voltage scaling, and even radiation hardening [6, 7]. For example, qualified versions of SRAM-based FPGAs, such as Xilinx's QPro [8], are commercially available for mitigating SEUs at the circuit level. More recently, hardware designers started to employ information redundancy, hardware redundancy, time redundancy, or a combination of these techniques in order to circumvent hardware defects at the device, circuit, and architectural levels. Specifically, the use of redundant hardware, data integrity checking, and data redundancy across multiple devices is gaining considerable popularity [9–11]. In fact, several engineers and researchers [12, 13] have demonstrated that logic-level triple-modular redundancy (TMR) with scrubbing of the FPGAs programming (or configuration) data effectively mitigates the results of radiation-induced single-bit upsets (SBUs). However, most of these redundancy techniques detect and recover from errors through expensive redundancy statically, that is, the allocation of hardware redundancy based on previously known or estimated error information, usually treating all

components within a system indiscriminately, and therefore incur huge area and performance penalties.

Recognizing TMR is costly; a recent study from BYU [14] proposed applying partial TMR method on the most critical sections of the design while sacrificing some reliability of the overall system. Specially, they introduced an automated software tool that uses the partial TMR method to apply TMR incrementally at a very fine level until the available resources are utilized, thus maximizing reliability gain for the specified area cost. Although with apparent conceptual similarity to this work, there are several important distinction between studies [14–16] and this one. First, while the study [14] focuses primarily on TMR or partial one, we consider more general case of selecting among n-nary modular redundancy (nMR) choices, where $n = 3, 5, 7, \ldots$. This generalization turned out to be important. In Section 7, we will show that in our target H.264 application, if more than TMR, such as 7-MR, is applied to certain components, the overall error resilience can be significantly improved. In addition, we provide a stochastic formulation of maximizing a system's error resilience when hardware failures are *spatially probabilistic* and *temporally unpredictable*. As a result, we can in principle obtain a mathematically provable optimal solution to maximizing the system's overall error resilience while being constrained by a total available hardware redundancy. Finally, while the techniques studied in [14–16] are mostly specific to FPGA fabric, our proposed methodology can be readily introduced into logic synthesis for an application-specific integrated circuit (ASIC) design.

Identifying the most critical circuit structures, thus trading hardware cost for SEU immunity, is not a new idea. In fact, Samudrala et al. proposed the selective triple modular redundancy (STMR) method which uses signal probabilities to find the SEU-sensitive subcircuits of a design [15]. Morgan et al. have proposed a partial mitigation method based on the concept of persistence [16]. We approach the criticality analysis differently in this study. Instead of being logic circuit or FPGA fabric-specific, we analyze individual system criticality by computing output sensitivity subjected to various amount of input perturbation probabilistically. Therefore, we are able to not only distinguish critical and noncritical components, but also quantitatively assigning different criticality values, which leads to a more cost-effective allocation of hardware redundancy.

This paper proposes a selectively fortified computing (SFC) approach to providing error resilience that preserves the delivery of expected performance and accurate results, despite of the presence of faulty components, in a robust and efficient way. The key idea of SFC is *to judiciously allocate hardware redundancy according to each key component's criticality towards target device's overall error resilience*. It leverages two emerging trends in modern computing. First, while ASIC design and manufacturing costs are soaring today with each new technology node, the computing power and logic capacity of modern FPGAs steadily advance. As such, we anticipate that FPGA-like reconfigurable fabric, due to its inherent regularity and built-in hardware redundancy, will become increasingly attractive for robust computing. Moreover, we believe that modern FPGA devices, bigger and more powerful, will become increasingly more suitable hardware platforms to apply the SFC concept and methodology for the improvements of fault-tolerance and computing robustness. Second, many newly emerging applications, such as data mining, market analysis, cognitive systems, and computational biology, are expected to dominate modern computing demands [17]. Unlike conventional computing applications, they typically process massive amounts of data and build mathematical models in order to answer real-world questions and to facilitate analyzing complex system, therefore tolerant to imprecision and approximation. In other words, for these applications, computation results need not always be perfect as long as the accuracy of the computation is "acceptable" to human users [18].

The rest of the paper is organized as follows. Section 2 overviews the proposed SFC framework and Section 3 discusses its target applications. We then delve into more detailed descriptions of criticality analysis and optimally allocating hardware redundancy with constrained hardware allowance. Subsequently, Section 6 describes in detail our H.264 decoder prototype and numerous design decisions. Finally, we present and analyze the error resilience results from a SFC design against its baseline without hardware redundancy in order to demonstrate its effectiveness, with Section 7 concluding the paper.

2. SFC Framework Overview

There are two conceptual steps depicted in Figure 1 to achieve hardware-efficient reliable computing by the approach of selectively fortified computing (SFC): criticality probing and hardware redundancy allocation. Because there is a fundamental tradeoff between hardware cost and computing reliability, it is infeasible to construct a computing system that can tolerate all the faults of each of its elements. Therefore, the most critical hardware components in a target device need to be identified and their associated computation should be protected as a priority. In SFC, we develop methods to identify sensitive elements of the system whose failures might cause the most critical system failures, prioritize them based on their criticality to user perception and their associated hardware cost, and allocate hardware redundancy efficiently and accordingly.

3. Target Applications

Many mission-critical applications, such as multimedia processing, wireless communications, networking, and recognition, mining, and synthesis, possess a certain degree of inherent resilience, that is, when facing device or component failure, their overall performance degrades gracefully [19–22]. Such resilience is due to several factors. First, the input data to these algorithms are often quite noisy and nevertheless these applications are designed to handle them. Moreover, the input data are typically large in quantity and frequently possess significant redundancy. Second, the algorithms underlying these applications are often statistical or probabilistic in nature. Finally, the output data of these

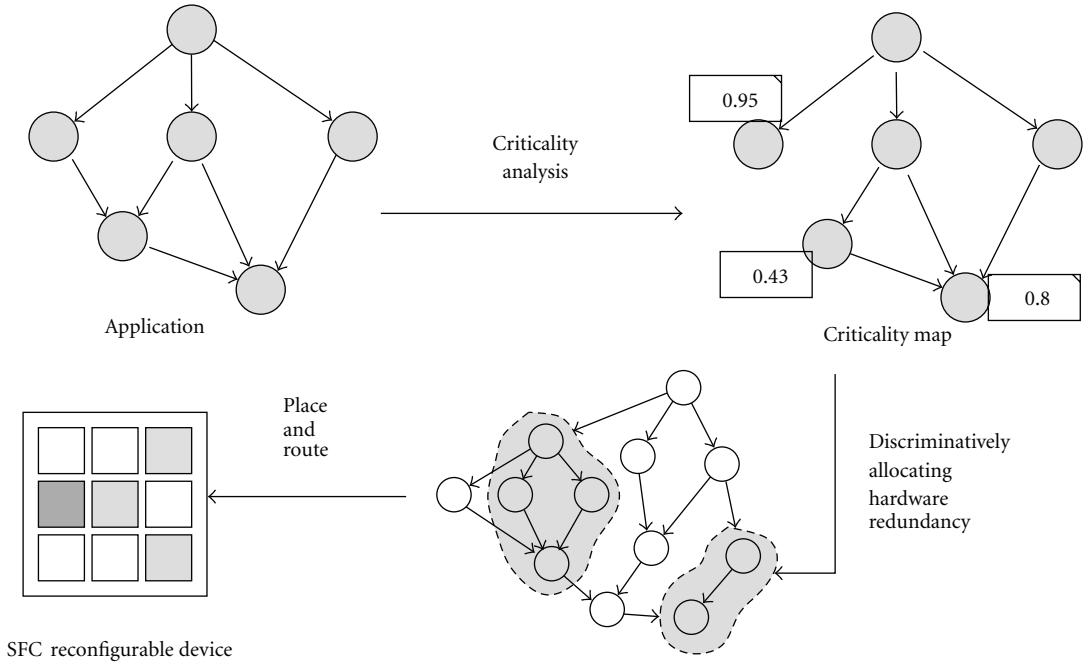

Figure 1: Conceptual flow of SFC (selectively fortified computing) methodology.

applications are mostly for human consumption, therefore some imperfections can be tolerated due to limited human perception. We believe that these applications are ideal SFC candidates because their overall system error resilience can potentially be significantly improved by selectively fortifying their key components. In this study, we illustrate our proposed SFC strategy by focusing on a highly efficient 720 p HD H.264 encoder [23]. We choose H.264 encoder because it is widely used in many mission-critical instruments. Moreover, H.264 contains a variety of computational motifs, from highly data parallel algorithms (motion estimation) to control intensive ones (CABAC) as depicted in Figure 2.

Our baseline FPGA implementation consists of five major functional components that account for more than 99% of the total execution time. Among these functional components, IME (integer motion estimation) finds the closest match for an image block from a previous reference image and computes a vector to represent the observed motion. While it is one of the most compute intensive parts of the encoder, the basic algorithm lends itself well to data parallel architectures. The next step, FME (fractional motion estimation), refines the initial match from integer motion estimation and finds a match at quarter-pixel resolution. FME is also data parallel, but it has some sequential dependencies and a more complex computation kernel that makes it more challenging to parallelize. IP (intraprediction) then uses previously encoded neighboring image blocks within the current image to form a prediction for the current image block. Next, in DCT/Quant (transform and quantization), the difference between a current and predicted image block is transformed and quantized to generate quantized coefficients, which then go through the inverse quantization and inverse transform to generate the reconstructed pixels. Finally, CABAC (context adaptive binary arithmetic coding) is used to entropy-encode the coefficients and other elements of the bit-stream. Unlike the previous algorithms, CABAC is sequential and control dominated. While it takes only a small fraction of the execution (1%), CABAC often becomes the bottleneck in parallel systems due to its sequential nature.

To appreciate the limited resilience characteristics of the HD H.264 encoder, we first implement all of five components using a standard HDL synthesis flow with Xilinx ISE Design Suite 13.0. Subsequently, errors are injected to four key components—IME (integer motion estimation), FME (fractional motion estimation), DCT/Quant (transform and quantization), and CABAC (context adaptive binary arithmetic coding) separately through VPI interfaces (see Section 5.2). These four components are quite diverse and representative in their computing patterns. More importantly, they together consume more than 90% of total execution load and more than 75% of total energy consumption [23]. For each of our target components X (I, II, III, and IV in Figure 2), we conduct 10000 fault injection experiments for each different error rate r according to the procedure outlined in Section 5.2. At a different redundancy ratio w_i, we repeat this procedure and measure the output deviation between the resulting image and its reference and quantify the result error as in Section 5.3. More details of our measurement procedure can be found in Section 5.

Before discussing the results presented in Figures 3 and 4, we formally define several key parameters. Some of them will be used in subsequent sections. Let h_i denote the hardware cost of implementing the component i in our target design. We define redundancy ratio $w_i = (h_i - h_{i,0})/(h_{i,0})$, where $h_{i,0}$ denotes the hardware cost of the component i without any redundancy.

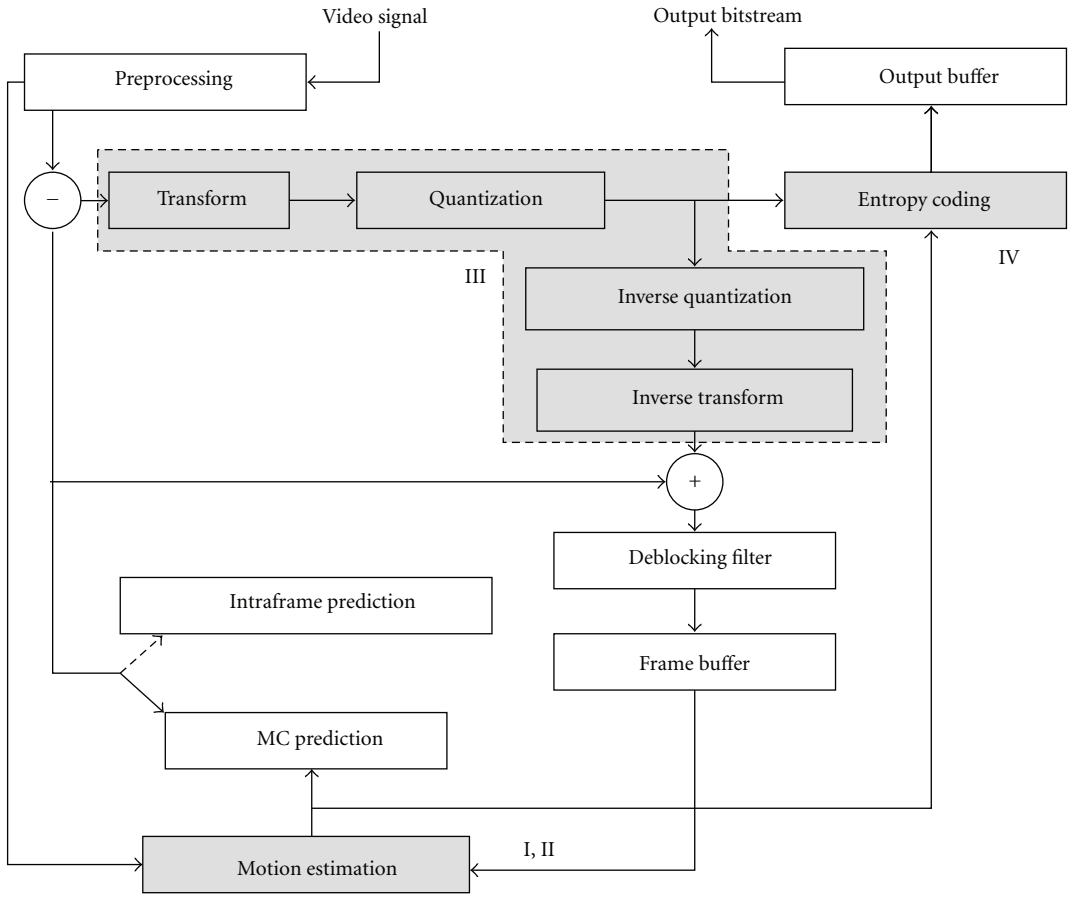

Figure 2: Block diagram of a standard H.264 encoder.

Figure 3 plots our measurement results for IME (integer motion estimation). Different curves represent output responses for different hardware redundancy ratios. Intuitively, the higher the value of w_i is, the more computing fortification the target device will possess. Roughly speaking, $w_i = 2$ is equivalent of a TMR (triple modular redundancy) unit. The values of w_i can be fractional because of partial redundancy as discussed in Section 6.1. In our experiments, error injection rate is measured as in error/gate/clock-cycle. We defer all discussions of error modeling and injection procedure to Section 5.2. Note in Figure 3 that errors injected at rate below 1×10^{-4} error/gate/clock-cycle do not impact the output quality by more than 1%, revealing the inherent resilience of such applications to rare data errors. However, error rates beyond 2.0×10^{-4} error/gate/clock-cycle impact image outputs significantly. We call this error rate as critical point. In our set of experiments, we injected errors by randomly selecting one of considered subcomponents and flipping one bit at a randomly chosen location. Our IME unit becomes totally dysfunctional after an average of 67 error injections when the error injection rate was as low as 1.13×10^{-4} error/gate/clock-cycle. These results confirm that, without proper system design, we cannot expect IME to produce useful results on unreliable hardware with relatively high error rate. Interestingly, as we increase hardware redundancy ratio, the critical point noticeably shifts towards high error rate. For our particular IME unit instance, diminish return of hardware redundancy quickly appears as w_i goes beyond 4, which is manifested by the flattening in its curve. Finally, at each w_i, the middle portion of each curve in Figure 3 exhibits a strong linearity, thus can be accurately approximated as a straight line.

In Figure 4, we fix error inject rate at 3.0×10^{-4} error/gate/clock-cycle and plot our measurement results for all four key components. The values of w_i can be fractional because of partial redundancy as discussed in Section 6.1. In general, different component responds quite differently to the same error rate. As w_i increases, the output error decreases consistently except for the component IV: CABAC. The relative small slopes of all curves in the middle portion again display early occurrence of diminished return of hardware redundancy. The most important finding of this experiment is to discover that DCT is far more sensitive to hardware redundancy than other components, while CABAC show exactly the opposite. Intuitively, this means that if facing hardware cost constrains, DCT should be assigned with a higher priority of hardware fortification and will have a larger benefit in improving the overall error resilience of the whole system. Finally, these results show that without

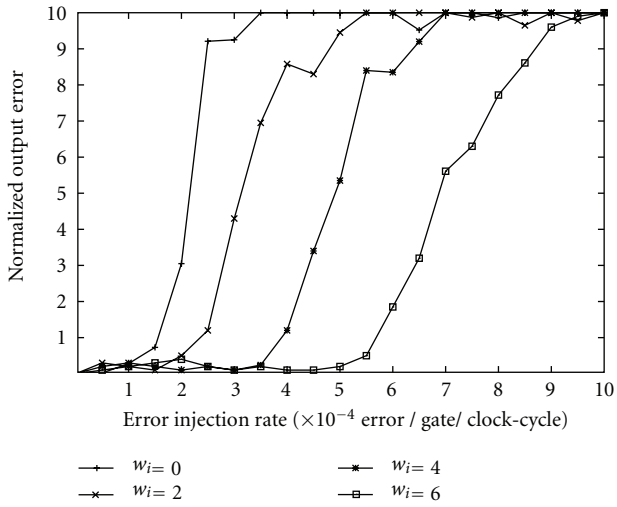

Figure 3: Output error versus error injection rate at different w_i for IME unit.

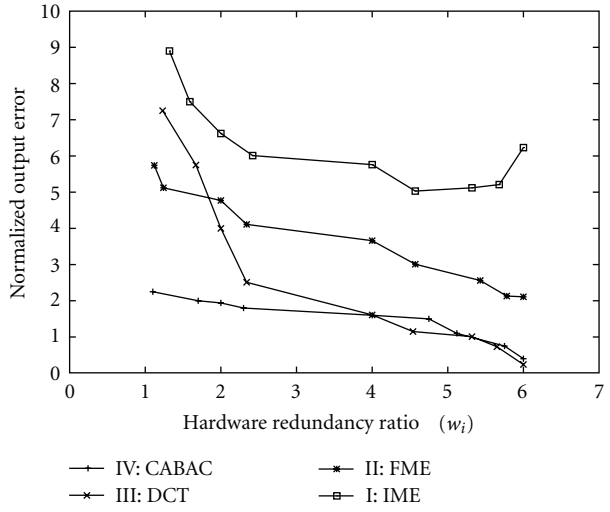

Figure 4: Output error versus hardware redundancy ratio w_i for different units.

computing fortification, the inherent error resilience for these key components, although exists, is quite limited.

4. Stochastic Formulation of Error Resilience

For a target design $\mathfrak{G}$ consisting of K functional components, each component C_i will be assigned a criticality value φ_i as defined in Section 5. Intuitively, a higher criticality value for a particular component C_i means more importance in its effect towards the overall correctness of final results. Moreover, we should allocate relatively more hardware resources to components with higher criticality in order to maximize the overall system's error resilience when constrained by total hardware usage.

Assume each component C_i consumes A_i units of hardware in the target application's final FPGA implementation. The unit used here can be either gate count or physical chip area. Furthermore, assuming all hardware defects are distributed in an i.i.d. manner, that is, all hardware defects are independent and identically distributed, all following a uniform distribution. Therefore, for any randomly occurring device error, the probability for one specific component C_i to be affected is $A_i/\sum_{j=1}^{K} A_j = g_i$. Note that in our proposed DFHS methodology, the random distribution of hardware effects can be in other form and does not limit the applicability of our proposed approach. In other words, if the randomly distributed hardware failures are more concentrated (i.e., occurring with higher probability) in certain component i, we can accordingly increase C_i to accommodate such changes.

Finally, we define the error probability of each component C_i to be $e_i = F(w_i, n_i)$, a function of two variables: w_i and n_i. w_i denotes the hardware redundancy ratio defined in Section 3, which quantifies how much hardware redundancy is built in the component C_i. Associated with each of the n functional components, there exist several choices of design alternatives having different hardware redundancy ratios w (as defined in Section 3). Typically, higher w_i values intuitively indicate more error resilience for a particular component at the price of high hardware usage. n_i denotes the number of hardware defects occurring in the component under consideration.

Given all the above definitions, further assuming error propagation within this target system is multiplicative, the objective of maximizing the overall error resilience of the target system $\mathfrak{G}$, under the attack of N randomly located hardware defects, can be formulated as minimizing the overall product of error probability and criticality among all its components:

$$E(\vec{w}, N) = \sum_{i=1}^{K} \varphi_i g_i e_i = \sum_{i=1}^{K} \varphi_i \left(\sum_{j=1}^{N} \left(\frac{A_i}{\sum_{k=1}^{K} A_k} \right)^j F(w_i, j) \right), \quad (1)$$

where $\vec{w}$ denotes $w_1, w_2, \ldots, w_K$. Clearly, constrained by a total hardware resource H, the problem is to determine which hardware redundancy ratio to select for each key component in order to achieve the greatest error resilience while keeping the total hardware cost within the allowable amount, taking into consideration each component's distinct criticality. This formulation of the problem leads to minimizing the total system error probability E:

$$\arg \min_{w_i, i \in [1,K]} E(\vec{w}, N), \qquad \text{s.t.} \sum_{i=1}^{n} A_i \leq H. \quad (2)$$

Interestingly, because both g_i and e_i depend on w_i, (1) is fundamentally a nonlinear multivariate optimization problem. More discussion on solving this optimization problem can be found in Section 6.

5. Criticality Analysis

Criticality analysis (CA) provides relative measures of significance of the effects of individual components on the

overall correctness of system operation. In essence, it is a tool that ranks the significance of each potential failure for each component in the system's design based on a failure rate and a severity ranking. Intuitively, because some parts within an application may be more critical than others, more computing resources should be allocated to these more critical components, that is, stronger computing "fortification," in order to achieve higher overall error resilience. In the framework of SFC, because the target computing device to be fortified is synthesized with a standard HDL flow and then realized (placed and routed) with a reconfigurable device such as an FPGA, criticality analysis is the key step to optimally allocate hardware redundancy within the target implementation.

Criticality analysis has been extensively studied within software domain [24–26], but quite rare in error-resilient computing device research. Our proposed approach to quantifying criticality directly benefits from mature reliability engineering techniques and scenario-based software architecture analysis. The key insight underlying our approach is that criticality analysis can be recast as a combined problem of uncertainty analysis and sensitivity analysis. Sensitivity analysis involves determining the contribution of individual input factors to uncertainty in model predictions. The most commonly used approach when doing a sensitivity analysis on spatial models is using Monte Carlo simulation. There are a number of techniques for calculating sensitivity indices from the Monte Carlo simulations, some more effective or efficient than others (see Figure 5).

Figure 6 depicts our procedure of criticality analysis. For each target system implemented with HDL, we first dissect it into multiple components. Often based on domain-specific knowledge of designer, a set of key components is selected and undergoes our criticality analysis. Fault modeling and injection mechanism are discussed in Section 5.2, and how to quantify the output errors is described in Section 5.3. To facilitate later discussion, we now define two metrics: sensitivity ψ_i and criticality φ_i for a specific component i at error rate e, the sensitivity value $\psi_i = \Delta y_i(e)/\Delta w_i$. Moreover, we define the criticality value φ_i as

$$\varphi_i = \mathbf{E}\left[\frac{(1/\psi_i)}{\sum_j \left(1/\psi_j\right)(e)}\right], \tag{3}$$

where $\mathbf{E}[\cdot]$ denotes the expected averaging over a wide range of error injection rates. In our case, we picked an i.i.d. uniform distribution for different error rates. ψ_i can be further approximated as a linear equation $\psi_i(e) = a_i(e) + b_i(e)w_i$. Both a_i and b_i values can be obtained through performing least square data fitting on our empirical data exemplified in Figure 4.

5.1. System Component Dissection. For a given target hardware implementation, before performing sensitivity and criticality analysis, the first step is to decompose the overall system into components. In this study, such system dissection typically follows naturally with individual module boundary. In most instances, block diagrams of logic implementation clearly shows the boundary between different system modules.

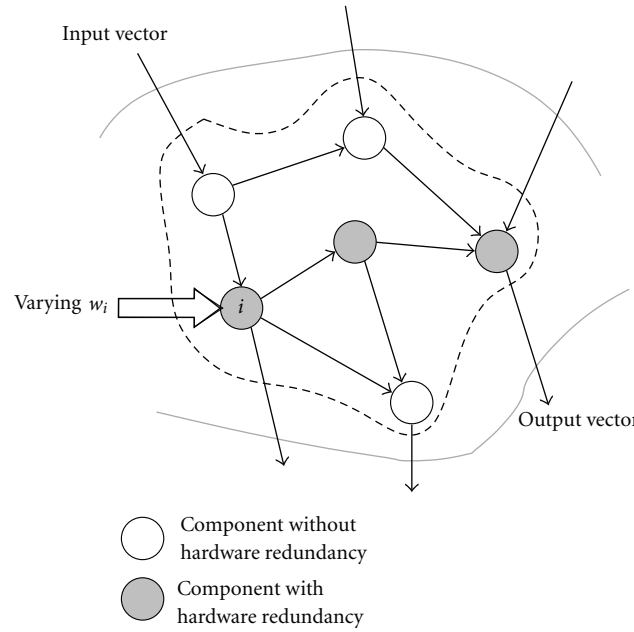

Figure 5: Conceptual diagram of sensitivity and criticality analysis.

We now use IME as an example to illustrate our FPGA implementation in detail. More importantly, we demonstrate how we perform component dissection in this SFC study. IME computes the motion vector predictor (MVP) of current macroblock (MB). There are many kinds of algorithms for block-based IME. The most accurate strategy is the full search (FS) algorithm. By exhaustively comparing all reference blocks in the search window, FS gives the most accurate motion vector which causes minimum sum of absolute differences (SAD) or sum of square difference (SSD). Because motion estimation in H.264/AVC supports variable block sizes and multiple reference frames, high computational complexity and huge data traffic become main difficulties in VLSI implementation [23]. Therefore, current VLSI designs usually adopt parallel architecture to increase the total throughput and solve high computational complexity. We implemented our IME unit in Verilog HDL language by closely following the block diagram in Figure 7. Among all units, PE array and four-input comparators perform computation on input pixel values and are most critical to the accuracy of final outputs. We totally have 41 four-input comparators and 138 PE units. Our approach allows either fortifying some or all of these subunits.

5.2. Fault Modeling and Injection. Much of our fault modeling and error injection mechanism is based on prior studies [27–29]. Our fault injection and evaluation infrastructure exploit Verilog programming interface (VPI) in order to quickly and flexibly inject emulated hardware defects into Verilog-based digital design in real time. Unlike many traditional fault injection methods, which either need to modify the HDL code or utilize simulator commands for

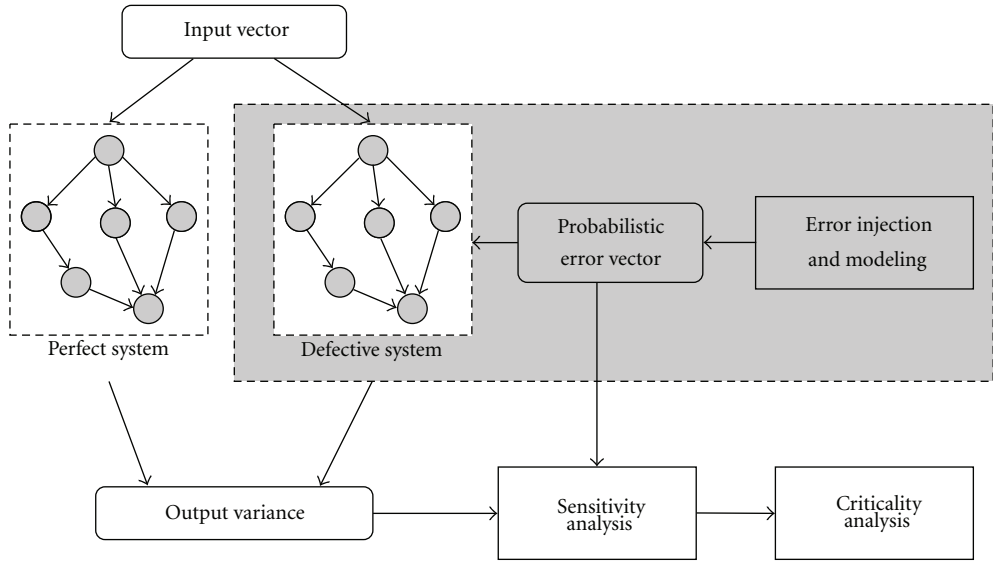

Figure 6: Flow chart of criticality analysis.

fault injection, our VPI-based approach is minimally invasive to the target design. Due to standardization of the VPI, the framework is independent from the used simulator. Additionally, it does not require recompilation for different fault injection experiments like techniques modifying the Verilog code for fault injection.

As shown in Figure 8, at the beginning of the simulation, the simulator calls the system with the command `$HDL_InjectError`, which generates one fault injection callback that is subsequently registered according to the specified arguments fault type and place. Each of such callbacks is specified by its time instant and duration and will be scheduled to execute at the injection instant and the end of injection. Then, when the inject instant reaches, the according callback will be executed by the simulator. This callback sets the conditions when the value modification on the specified place will occur. During the injection, the value of the injection place is modified by using VPI to control the Verilog simulator. At the end of fault, the injection callback is executed again to restore the normal value. One advantage of using VPI-based fault injection is that it can be applied to all VPI compliant Verilog simulators.

In order to make fault injection and evaluation feasible, we limit the fault models used in this work to single bit faults. A variety of fault models are defined in order to represent real physical faults that occur in integrated circuits (ICs). We totally consider four kinds of hardware fault. In this work, the bit flip representing the inverted value at the instant ts is differentiated from the toggling bit flip that toggles with original value. In principle, fault types can be readily extended with additional fault types.

5.3. Evaluating Fault Impact. The criticality analysis in SFC requires accurately quantifying the impact of injected hardware faults by comparing the result images with the reference ones. Unfortunately, computing the overall differences of pixel values will not detect and record the visual differences critical to human perception [30]. In this study, we feel other instances of comparing. In particular, the methodology presented here is based on a stochastic Petri-net (SPN) graph We instead calculate the Hausdorff distance to extract and record local and global features from both images, compare them, and define the percentage of similarity.

The Hausdorff distance measures the extent to which each point of a "model" set lies near some point of an "image" set and vice versa. Thus, this distance can be used to determine the degree of resemblance between two objects that are superimposed on one another. In this paper, we provide efficient algorithms for between all possible relative positions of a binary image and a model.

Given two finite point sets $A = a_1, a_2, \ldots, a_p$ and $B = b_1, b_2, \ldots, b_q$, the Hausdorff distance is defined as

$$H(A,B) = \max(h(A,B), h(B,A)), \tag{4}$$

where

$$H(A,B) = \max_{a \in A} \min_{b \in B} \|a - b\| \tag{5}$$

and $\|\cdot\|$ is some underlying norm on the points of A and B (e.g., the $L2$ or Euclidean norm). The Hausdorff distance $H(A,B)$ is the maximum of $h(A,B)$ and $h(B,A)$. Thus, it measures the degree of mismatch between two sets by measuring the distance of the point of A that is farthest from any point of B and vice versa. Intuitively, if the Hausdorff distance is d, then every point of A must be within a distance d of some point of B and vice versa. Thus, the notion of resemblance encoded by this distance is that each member of A be near some member of B and vice versa. Unlike most methods of comparing shapes, there is no explicit pairing of points of A with points of B (e.g., many points of A may be close to the same point of B). The function $H(A,B)$ can be trivially computed in time $\mathcal{O}(pq)$ for two-point sets

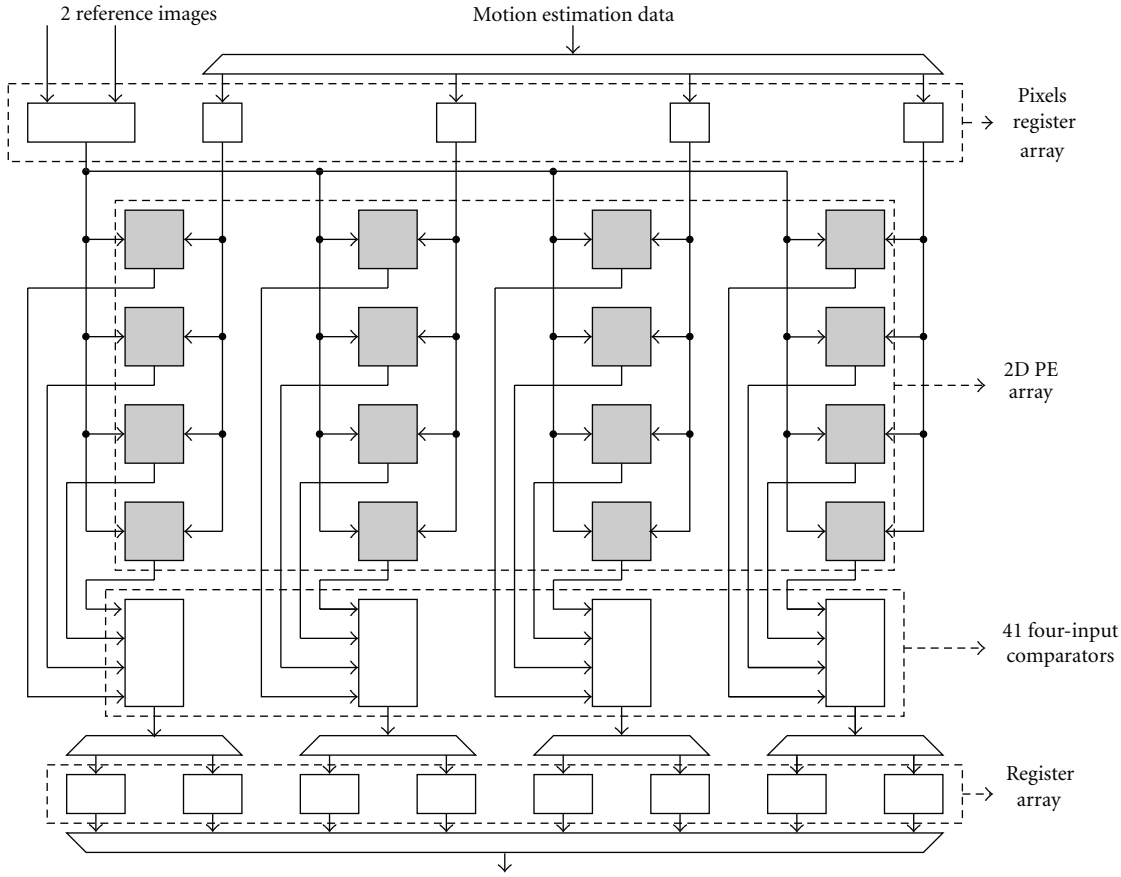

Figure 7: Block diagram of an integer motion estimator design with a macroblock parallel architecture and sixteen 2D PE arrays [23].

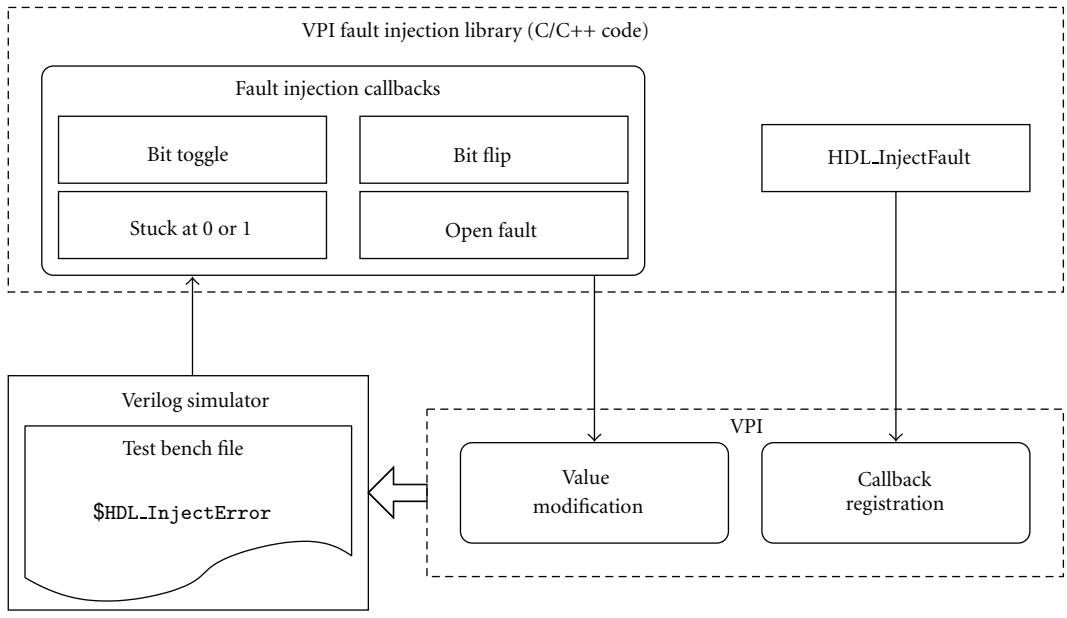

Figure 8: Block diagram of VPI simulation and fault injection flow [29].

of size p and q, respectively, and this can be improved to $\mathcal{O}((p+q)\log(p+q))$ [30].

6. Optimally Allocating Hardware Redundancy

Assuming that our target system consists of n functional components and error propagation within the target system is multiplicative. Associated with each of the n functional components, there exist several choices of design alternatives having different hardware redundancy ratio w (as defined in Section 3). Let e_i represent the ith component with a specified inherent error tolerance, and let $R_i(w)$ denote the derived error tolerance function of the ith component when the hardware redundancy ratio of e_i is w. Given the overall hardware cost limit of H, the problem is to determine which hardware redundancy ratio to select for each key component in order to achieve the greatest error resilience while keeping the total hardware cost within the allowable amount. This formulation of the problem leads to the maximization of system reliability R given by the product of unit error resilience

$$\arg \max_{w_i, i\in[1,n]} R = \prod_{i=1}^{n} R_i(w_i), \qquad \text{s.t.} \sum_{i=1}^{n} h_i \le H. \tag{6}$$

We now discuss the procedure to obtain both $R_i(w)$ and h_i and how to solve the above multivariate optimization problem.

6.1. Modeling Reliability and Its Hardware Cost. We consider two kinds of hardware redundancy: fractional redundancy and n-nary modular redundancy. First, for multiple subcomponents within one unit, we allow a fraction of these subcomponents to have hardware redundancy. For example, in Figure 7, a typical IME unit contains 138 PE units, each of which independently performs sum of absolute differences (SAD) and sum of square difference (SSD) operations. Due to stringent hardware costs, we normally cannot afford to build hardware redundancy into each of these PE units. Instead, our SFC framework allows part of these units to be fortified. In this study, the selection of these units is done randomly for a given fixed percentage. Second, our SFC system allows classic n-nary modular redundancy for chosen components. An n-tuple modular system is a natural generalization of the TMR (triple-modular redundancy) concepts [31], which utilizes $2n+1$ units together with a voter circuitry and its reliability is $R(NMR) = \sum_{i=0}^{n} \binom{N}{i} (1-R)^i R^{N-i}$ where the combinatorial notation $\binom{N}{i} = N!/((N-i)!i!)$.

Despite being conceptually simple, analytically computing the cost of hardware redundancy and its impact on the overall system's error resilience proves to be extremely difficult and inaccurate [31]. In this study, we opted to obtain both values empirically. As shown in Figure 3, $\varphi_i(w)$ can be approximated as $a_i + b_i w_i$, where a_i and b_i are the regression coefficients derived from experimental data. Furthermore, the hardware cost of implementing fortified component i is $(1+w_i)h_{i,0}$ according to w_i's definition.

6.2. Solving Allocation Problem. In this section, we determine the optimal allocation for a generic hardware design subject to a set of hardware redundancy with various costs, for example, solving the optimization problem defined in (6). Note $\arg\max_{w_i, i\in[1,n]} R$ equals $\arg\max_{w_i, i\in[1,n]} \ln R = \arg\max_{w_i, i\in[1,n]} \sum_{i=1}^{n} \ln R_i(w_i)$, where $\ln(a_i + b_i w_i)$ can be extended using Taylor series as

$$\begin{aligned}(a_{i,0} + b_{i,0}w_{i,0}) + \frac{d\ln(a_i + b_i w_i)}{dw_i}\Big|_{w_i = w_{i,0}} \cdot (w_i - w_{i,0}) + \cdots \\ \approx \alpha_i w_i + \beta_i,\end{aligned} \tag{7}$$

where $\alpha_i = (b_{i,0})/(a_{i,0} + b_{i,0}w_{i,0})$ and $\beta_i = ((a_{i,0} + b_{i,0}w_{i,0})^2 - b_{i,0}w_{i,0})/(a_{i,0} + b_{i,0}w_{i,0})$. In practice, $w_{i,0}$ should be chosen to be close to the optimal solution w_i^* of (6) in order to approximate more accurately. Therefore, (6) becomes

$$\arg \max_{w_i, i\in[1,n]} \sum_{i=1}^{n} \alpha_i w_i + \beta_i, \qquad \text{s.t.} \sum_{i=1}^{n} h_{i,0}(w_i + 1) \le H. \tag{8}$$

Equation (8) is a classic linear programming problem with bounded constraints and therefore can be readily solved by the simplex algorithm [32] by constructing a feasible solution at a vertex of the polytope and then walking along a path on the edges of the polytope to vertices with nondecreasing values of the objective function until an optimum is reached. Although in theory, the simplex algorithm can perform poorly for the worse case, in practice, the simplex algorithm is quite efficient and can be guaranteed to find the global optimum rather quickly [32]. In this study, we implemented a revised simplex method according to the algorithm outlined on page 101 of [33]. For all the simplex runs we have performed, they all complete successfully within 20,000 iterations.

7. Hardware Prototyping and Performance Comparison

Our prototype of H.264 encoder is based on the parallel architecture suggested by [23] and implemented with a Virtex-5 FPGA (XCV5LX155T-2) device. The functionality of this prototype is verified against a software reference design provided with MediaBench II Benchmark [34]. Table 1 lists the hardware usage of various components, in which IME and FME consume more than half of the total hardware usage and exhibits a clear data parallel computing pattern while CABAC is much smaller in gate usage but is totally control dominated.

For input image data, we used the base video input data set that has a 4 CIF video resolution (704×576) with a moderate degree of motion and is compressed to a bitrate providing medium/average video quality using a search window of ± 16 pixels for motion estimation [34]. As shown in Figure 9, we compare error resilience results between our baseline design with zero hardware redundancy, equal criticality solution, and the SFC design with optimally allocated redundancy. All three designs are subjected to

Table 1: Hardware usage of a H.264 encoder with a Virtex-5 FPGA (XCV5LX155T-2) device.

Unit	Logic elements	Memory bits	%	φ_i	w_i
IME	58960	4500	21.2	0.13	2.13
FME	79842	7600	29.0	0.10	1.89
IP	47823	3600	17.1	0.23	3.42
DCT	8966	750	3.2	0.43	5.78
CABAC	62451	2100	21.5	0.11	2.11
Other	10210	410	8	—	—

a wide range of error injection rates. To make a fair comparison, both the equal criticality solution and the SFC design are only allowed 30% additional hardware for design redundancy. For our baseline design, the error resilience is quite limited. As soon as the error injection rate exceeds 0.5×10^{-4} error/gate/clock-cycle, the normalized output error immediately jumps to more than 1, which roughly means more than 10% of output image data is corrupted and unrecognizable. We obtained the equal criticality solution by equalizing the ratio between hardware usage and criticality among all key components, that is, for components $i = 1, \ldots, n$, equalizing h_i/φ_i. Intuitively, this allocation scheme means assigning more hardware redundancy to more critical components, which impact more onto the system's overall error resilience. From Figure 9, we can see clearly that this criticality equalizing approach, despite conceptually simple, yields significant improvements. In fact, on average, the equal criticality solution improves the average error resilience by almost 1.5 times. Finally, after applying the method outlined in Section 6.2, the SFC solution produces even further improvements in error resilience. Now, the H.264 encoder can perform almost undisturbed even the error injection rate reaches beyond 4×10^{-4} error/gate/clock-cycle, an almost 4 and 2.5 times improvement over the baseline design, respectively.

As shown in Figure 9, for different number of hardware faults N and a total amount of available hardware redundancy W, the optimal hardware redundancy allocation is different. As a user of DFHS, when we make our FPGA design, we do not know exactly how many hardware errors will be encounter, therefore how do we make our choice of $\vec{w}$ to maximize its error resilience? Figure 10 presents three different allowable total hardware redundancy $W = 50\%$, 75%, and 85%. For each W, we plot two curves showing the total error resilience E as defined in (1) versus different numbers of hardware errors N. The lower curve is the result of optimal $\vec{w}$, whereas the upper curve is the worst E for each different N when choosing any of optimal $\vec{w}$ solutions at different Ns. This figure is useful because, for any specified maximumly allowed error resilience, the user can use Figure 10 to decide how much total hardware redundancy should be used and how many hardware failures the resulted design can tolerate. For example, if the maximumly allowed E is 0.12, using 50% total hardware redundancy and any optimal solution $\vec{w}$ when considering N ranges from 1 to 20,

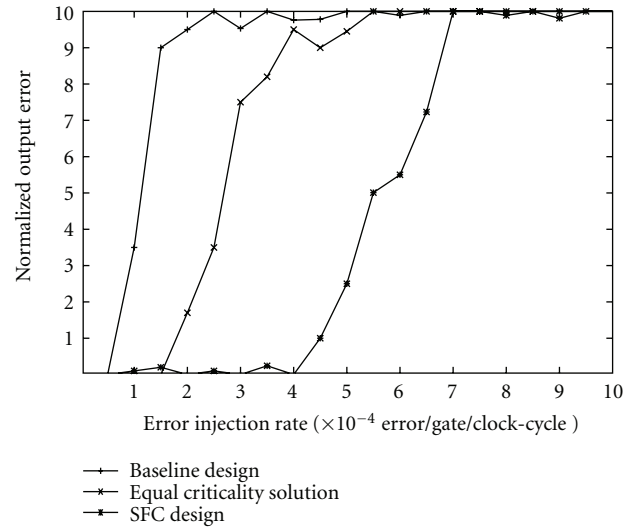

Figure 9: Error resilience comparison between baseline, equal criticality solution, and SFC design.

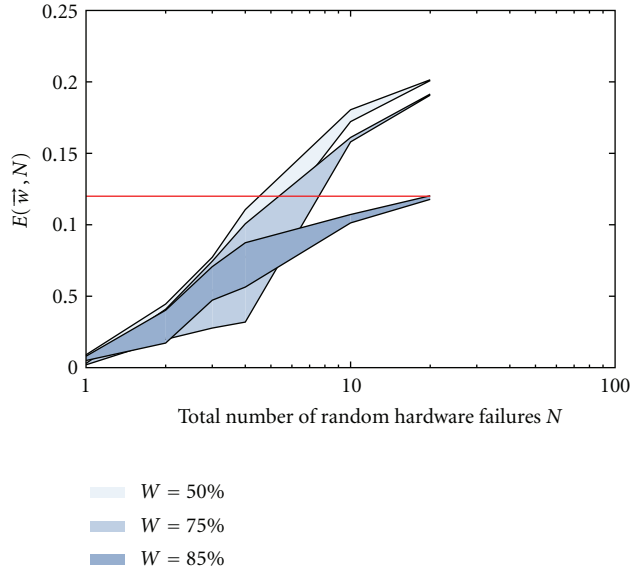

Figure 10: Design space exploration of DFHS for different allowed amount of hardware redundancy.

the resulted design can tolerate no more than 4 hardware failures. If the user desires to tolerate no more than 20 hardware failures ($E \leq 0.12$), the design needs to use at least 85% of total hardware redundancy.

8. Conclusion

This study introduces the concept of selectively fortified computing (SFC) for many mission-critical applications with limited inherent error resilience. The SFC methodology deviates significantly from the conventional approaches that heavily rely on static temporal and/or spatial redundancy and sophisticated error prediction or estimation techniques.

Instead of treating hardware or software redundancy as static objects and assuming prior knowledge of defects, the SFC approach focuses on providing much more hardware-efficient reliable computing through efficiently and strategically distributing hardware redundancy to maximize the overall achievable reliability.

To validate the feasibility to implement an error-resilient reconfigurable computing system with SFC, we have implemented a 720P H.264/AVC encoder prototype with an Virtex 5 FPGA device that operates at very high error rates. The goal was to show that even under such harsh error environment, our SFC system can still maintain high error tolerance without incurring excessive hardware redundancy. To achieve such robustness, various algorithms are developed to perform hardware sensitivity and criticality analysis, followed by an efficient heuristic methodology to optimally allocate hardware redundancy. Our experimental results from a 720P H.264/AVC encoder prototype implemented with an Virtex 5 device has clearly demonstrated the effectiveness of SFC operating under a wide range of error rates. Specifically, this H.264 prototype can tolerate the error rates as high as 18,000 errors/sec/RRC, which are emulated by hardware defects uniformly distributed through the whole computing device. Such high error rates extend far beyond radiation-induced soft error rates and may be caused by highly frequent erratic intermittent errors, process variations, voltage droops, and Vccmin. Compared with the unmodified reference design, the SFC prototype maintains 90% or better accuracy of output results and achieves more than four times of error resilience when compared with a baseline design. In principle, SFC is not restricted to video processing applications and can be applied to other general-purpose applications that are less resilient to errors.

Acknowledgments

This work was partially supported by DARPA System Center (Grant no. N66001-04-1-8916). Partial results of this study were published in the HASE2011 conference.

References

[1] P. Bose, "Designing reliable systems with unreliable components," *IEEE Micro*, vol. 26, no. 5, pp. 5–6, 2006.

[2] S. Borkar, "Designing reliable systems from unreliable components: the challenges of transistor variability and degradation," *IEEE Micro*, vol. 25, no. 6, pp. 10–16, 2005.

[3] W. Robinett, G. S. Snider, P. J. Kuekes, and R. S. Williams, "Computing with a trillion crummy components," *Communications of the ACM*, vol. 50, no. 9, pp. 35–39, 2007.

[4] S. L. Jeng, J. C. Lu, and K. Wang, "A review of reliability research on nanotechnology," *IEEE Transactions on Reliability*, vol. 56, no. 3, pp. 401–410, 2007.

[5] S. R. Nassif, N. Mehta, and Y. Cao, "A resilience roadmap," in *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE 10)*, pp. 1011–1016, March 2010.

[6] N Haddad, R. Brown, T. Cronauer, and H. Phan, "Radiation hardened cots-based 32-bit microprocessor," in *Proceedings of the 5th European Conference on Radiation and Its Effects on Components and Systems (RADECS '99)*, pp. 593–597, Fontevraud , France, 1999.

[7] W. Heidergott, "SEU tolerant device, circuit and processor design," in *Proceedings of the 42nd Design Automation Conference (DAC '05)*, pp. 5–10, ACM, New York, NY, USA, June 2005.

[8] "QPro Virtex-II Pro 1.5V Platform FPGAs," http://www.xilinx.com/support/documentation/defenseqpro.htm/.

[9] M. A. Gomaa, C. Scarbrough, T. N. Vijaykumar, and I. Pomeranz, "Transient-Fault Recovery for Chip Multiprocessors," *IEEE Micro*, vol. 23, no. 6, pp. 76–83, 2003.

[10] J. Gaisler, "A portable and fault-tolerant microprocessor based on the SPARC V8 architecture," in *Proceedings of the International Conference on Dependable Systems and Networks (DNS '02)*, pp. 409–415, June 2002.

[11] H. Quinn, P. Graham, J. Krone, M. Caffrey, and S. Rezgui, "Radiation-induced multi-bit upsets in SRAM-based FPGAs," *IEEE Transactions on Nuclear Science*, vol. 52, no. 6, pp. 2455–2461, 2005.

[12] N. Rollins, M. Wirthlin, M. Caffrey, and P. Graham, "Evaluating TMR techniques in the presence of single event upsets," in *Proceedings of the 6th Annual International Conference on Military and Aerospace Programmable Logic Devices*, pp. 63–70, September 2003.

[13] G. M. Swift, S. Rezgui, J. George et al., "Dynamic testing of xilinx virtex-II field programmable gate array (FPGA) input/output blocks (IOBs)," *IEEE Transactions on Nuclear Science*, vol. 51, no. 6, pp. 3469–3474, 2004.

[14] B. Pratt, M. Caffrey, J. F. Carroll, P. Graham, K. Morgan, and M. Wirthlin, "Fine-grain SEU mitigation for FPGAs using partial TMR," *IEEE Transactions on Nuclear Science*, vol. 55, no. 4, Article ID 4636895, pp. 2274–2280, 2008.

[15] P. K. Samudrala, J. Ramos, and S. Katkoori, "Selective triple modular redundancy (STMR) based single-event upset (SEU) tolerant synthesis for FPGAs," *IEEE Transactions on Nuclear Science*, vol. 51, no. 5, pp. 2957–2969, 2004.

[16] K. Morgan, M. Caffrey, P. Graham, E. Johnson, B. Pratt, and M. Wirthlin, "SEU-induced persistent error propagation in FPGAs," *IEEE Transactions on Nuclear Science*, vol. 52, no. 6, pp. 2438–2445, 2005.

[17] S. Narayanan, G. V. Varatkar, D. L. Jones, and N. R. Shanbhag, "Computation as estimation: estimation-theoretic IC design improves robustness and reduces power consumption," in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP '08)*, pp. 1421–1424, April 2008.

[18] M. A. Breuer, "Multi-media applications and imprecise computation," in *Proceedings of the 8th Euromicro Conference on Digital System Design (DSD '05)*, pp. 2–7, September 2005.

[19] R. Hegde and N. R. Shanbhag, "Energy-efficient signal processing via algorithmic noise-tolerance," in *Proceedings of the International Conference on Low Power Electronics and Design (ISLPED '99)*, pp. 30–35, ACM, New York, NY, USA, August 1999.

[20] D. Mohapatra, G. Karakonstantis, and K. Roy, "Significance driven computation: a voltage-scalable, variation-aware, quality-tuning motion estimator," in *Proceedings of the 14th ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED '09)*, pp. 195–200, ACM, New York, NY, USA, August 2009.

[21] R. Nair, "Models for energy-efficient approximate computing," in *Proceedings of the 16th ACM/IEEE International*

Symposium on Low Power Electronics and Design (ISLPED '10), pp. 359–360, ACM, New York, NY, USA, 2010.

[22] L. Leem, H. Cho, J. Bau, Q. A. Jacobson, and S. Mitra, "Ersa: error resilient system architecture for probabilistic applications," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE '10)*, pp. 1560–1565, European Design and Automation Association, Leuven, Belgium, 2010.

[23] Y.-L. S. Lin, C.-Y. Kao, H.-C. Kuo, and J.-W. Chen, *VLSI Design for Video Coding: H.264/AVC Encoding from Standard Specification to Chip*, Springer, 1st edition, 2010.

[24] P. G. Bishop, R. E. Bloomfield, T. Clement, and S. Guerra, "Software criticality analysis of cots/soup," in *Proceedings of the 21st International Conference on Computer Safety, Reliability and Security (SAFECOMP '02)*, pp. 198–211, Springer, London, UK, 2002.

[25] C. Ebert, "Fuzzy classification for software criticality analysis," *Expert Systems with Applications*, vol. 11, no. 3, pp. 323–342, 1996.

[26] P. Anderson, T. Reps, and T. Teitelbaum, "Design and implementation of a fine-grained software inspection tool," *IEEE Transactions on Software Engineering*, vol. 29, no. 8, pp. 721–733, 2003.

[27] R. Leveugle, D. Cimonnet, and A. Ammari, "System-level dependability analysis with RT-level fault injection accuracy," in *Proceedings of the 19th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT '04)*, pp. 451–458, October 2004.

[28] J. Arlat, M. Aguera, L. Amat et al., "Fault injection for dependability validation: a methodology and some applications," *IEEE Transactions on Software Engineering*, vol. 16, no. 2, pp. 166–182, 1990.

[29] D. Kammler, J. Guan, G. Ascheid, R. Leupers, and H. Meyr, "A fast and flexible platform for fault injection and evaluation in Verilog-based simulations," in *Proceedings of the 3rd IEEE International Conference on Secure Software Integration Reliability Improvement (SSIRI '09)*, pp. 309–314, July 2009.

[30] N. G. Bourbakis, "Emulating human visual perception for measuring difference in images using an SPN graph approach," *IEEE Transactions on Systems, Man, and Cybernetics B*, vol. 32, no. 2, pp. 191–201, 2002.

[31] F. P. Mathur and A. Avižienis, "Reliability analysis and architecture of a hybrid-redundant digital system: generalized triple modular redundancy with self-repair," in *Proceedings of the Spring Joint Computer Conference (AFIPS '70)*, pp. 375–383, ACM, New York, NY, USA, May 1970.

[32] G. B. Dantzig and M. N. Thapa, *Linear Programming 1: Introduction*, Springer, Secaucus, NJ, USA, 1997.

[33] R. Darst, *Introduction to Linear Programming: Applications and Extensions*, Pure and Applied Mathematics, M. Dekker, 1991.

[34] J. E. Fritts, F. W. Steiling, J. A. Tucek, and W. Wolf, "MediaBench II video: expediting the next generation of video systems research," *Microprocessors and Microsystems*, vol. 33, no. 4, pp. 301–318, 2009.