Error Modeling for Image Processing Filters accelerated onto SRAM-based FPGAs

Cristiana Bolchini, Luca Cassano, Andrea Mazzeo, Antonio Miele
Dip. Elettronica, Informazione e Bioingegneria – Politecnico di Milano – Italy
{first_name.last_name}@polimi.it

***Abstract*—Image processing is today employed in a variety of application fields, including safety- and mission-critical ones. In these scenarios it is vital to carefully analyse the reliability of the designed system before deployment and, if necessary, to adopt specific hardening techniques. Two are the techniques generally employed: circuit-level fault injection and application-level functional error simulation. In this paper we present a set of functional error models specific for a number of convolution-based filters that are the basic building blocks for a wide range of image processing applications. The presented error models, derived through a number of circuit-level fault injection experiments, may be integrated into application-level functional error simulators, bridging the gap between the two strategies. The presented error models are the first step towards combining the accuracy of fault injection and the flexibility of error simulation into a widely adopted reliability analysis tool.**

***Index Terms*—Error Modeling, Error Simulation, Fault Injection, Fault Tolerance, Image Processing, Reliability Analysis**

I. Introduction

Image processing is today employed in a variety of safety- and mission-critical scenarios, where digital systems have to meet reliability requirements [1]. As an example let us consider a satellite hosting payload applications accelerated onto an FPGA; such a system operates in a harsh environment, where radiations may induce faults, e.g., *Single Event Upset (SEU)*, in the circuitry that may lead to a failure of the running application [2]. This quest for reliability is exacerbated by the recent trend/near-future interest in integrating autonomous driving mechanisms in automotive or unmanned flying vehicles [3]. Indeed, such systems rely on the perception functionality; it processes images of the surrounding environment taken from cameras to extract features that are then transmitted to the control application to take driving decisions [4].

Image processing applications are generally highly data and compute intensive tasks. Therefore, classical redundancy-based fault detection and mitigation techniques, such as Duplication with Comparison (DWC) or Triple Modular Redundancy (TMR), may not be affordable due to the excessive overhead in terms of area occupation, performance degradation or power consumption increase [5]. On the other hand, these applications may expose an intrinsic degree of fault tolerance due to several reasons: i) they may deal with noisy inputs (e.g., sensors), ii) their outputs may be probabilistic estimates, or iii) produced images may be used by a human, whose perceptual limitations provide resiliency to a certain level of inexactness [6]. It is therefore possible not to analyse the effect of the faults at the circuit-level, but at the application-level, by observing how faults (and their effect on the produced output) affect the ability of the end-user application to carry out its task [7]. This information may be exploited to specifically tailor the hardening process and to reduce the implementation costs. To do so, it is necessary first to accurately assess the intrinsic resiliency of the application and then to finely tune the hardening mechanisms so that the application-level reliability requirements are met at a limited cost [7].

To pursue such strategy, it is necessary to analyze the effects of the faults occurring in the underlying hardware platform on the behaviour of the application itself. The current best practice, based on circuit-level fault injection, allows for an accurate reliability analysis, but it requires the overall system to be fully implemented and deployed on an FPGA device [8]–[11]. This represents a relevant limitation due to the time to design a hardware component, thus not allowing to provide a fast and early feedback to the hardening process before the final implementation and deployment on the FPGA device. Moreover, assessing the resiliency of the application by analysing the results produced through fault injection may be a complex and time-demanding task. When injecting a fault, based on the inputs it may produce no error or it can be masked, or again it may produce an intermediate incorrect data that is later absorbed by the application. On the other hand, error simulation, performed by injecting corrupted data in the application, allows to perform an early and low cost reliability analysis [12], [13]. Therefore, it is possible to perform a reliability analysis at functional level, thus offering valuable information to the hardening process without the need of time-expensive iterations between the design and implementation phases. Moreover, in every experiment the application will be fed with actually corrupted data, not incurring in fault activation and masking issues. The underlying hypothesis is that the corrupted data artificially created well represents the effects of a real possible fault, i.e., that the *image error model* is accurate. As a matter of fact, the adoption of application-level error simulation tools depends on the availability of accurate and validated error models.

The main contribution of this paper is the definition of functional error models for the basic operations of the image processing field, to be exploited in error simulators. These are the first steps for the definition of a functional framework for the reliability analysis of complex image processing applications in the early phases of the design flow. Image error

models have been defined by observing the effects of faults injected at the circuit-level in the hardware implementations of the considered image processing filters. The considered scenario consists of a set of convolutional filters, widely-used basic image processing operators, implemented onto SRAM-based FPGA; the adopted fault model is the SEU. The result of the proposed analysis is an experimentally validated set of 13 error models. The obtained models are generic enough to be integrated in an error simulation engine specific for image processing applications; we show how the error models are independent of the specific filter, of the input image and of the injection time. Preliminary results shows a 30x improvement of the analysis time w.r.t. the classical fault injection.

The rest of the paper is organized as follows. Next section discusses the related work and Section III presents the considered image filters. Then, Section IV describes the framework and the setup used the experimental campaign performed to define the image error models, commented in the subsequent Section V. Section VI discusses how such error models can be exploited to speed-up the reliability analysis. Finally, Section VII draws the conclusions and presents future work.

II. Related Work

Several fault injection frameworks have been proposed in the last decades; recent examples, compliant with modern device families are [8]–[11]. Their architecture is pretty standard; it is based on the FPGA internal memory configuration interface that is exploited to emulate an SEUs, directly accessed by custom modules or by the Xilinx SEM Intellectual Property (IP), and a controller to coordinate the execution of the fault injection campaign on a Design Under Test (DUT). These fault injectors are employed to evaluate the robustness of image processing applications (or part of them), such as a Convolutional Neural Network (CNN) [9], an image compression algorithm [10] or a K-means image clustering one [11]. In the reported experiments, outputs are analyzed in a more advanced way than the classical taxonomy (no-effect, crash/timeout, silent data corruption) by using image quality metrics (such as the SSIM in [10], [11]). In fact, the aim of these approaches is to accurately analyze how disruptive the fault is on the produced output. However, these frameworks present several drawbacks, starting from the fact that the final implementation of the system needs to be available and integrated in the fault injection framework, waiving also an early-stage evaluation of the system resiliency.

As an alternative to fault injection, error simulation can be adopted. In error simulation, a prototype of the system under analysis is exercised with emulated faults, i.e., corrupted data are injected into the application. In [12], [13] two different frameworks for error simulation in CNNs have been proposed; they integrate the injection facilities in popular machine learning development frameworks, i.e., Caffee and TensorFlow, respectively. However, the supported error models have not been validated (especially in [12]), and may be arbitrary or not represent a realistic effect due to a fault, such as, for instance, the error modeling a bitflip in the same position of all values of a tensor. Indeed, error simulation may incur in two risks: simulating errors that do not correspond to the effects of any real fault and, ignoring effects of statistically relevant faults.

Our contribution proposes a set of generic, validated and algorithmically reproducible error models, that are the observable effects of faults injected in the hardware platform. The final goal is to provide a library of such models for the implementation of easy-to-use, fast, flexible but also accurate error simulation engines for image processing applications.

III. The Considered Image Filters

Image processing applications receive one or more input images either to produce an enhanced image or to extract features. Such applications are organized as a pipeline of processing steps, where each step implements a specific image processing operation, generally called *filter*. A filter takes in input an image, or other data structures (e.g., a heatmap or a feature map) and produces a output image (or, again, a data structure). Filters may perform operations spanning from pixel-wise manipulations (e.g., the RGB to gray scale conversion) up to advanced Machine Learning-based activities (e.g., Neural Networks devoted to the classification of the image or part of it). In this paper we focus on a widely used class of the convolutional filters.

A *convolution* is a mathematical operation that is applied on an input image i with the use of a kernel (or filter), a 2D matrix f, having dimensions $N \times M$. The convolution operation slides the kernel on the matrix of pixels of the input image, to produce an output image o where the value in each position x, y is computed as the dot product between the kernel and the corresponding values on the input:

$$o(x,y) = \sum_{i=\ N/2}^{N/2} \sum_{j=\ M/2}^{M/2} f(i,j) \cdot i(x-i, y-j) \qquad (1)$$

The above processing is applied separately on the three channels for RGB images. Moreover, specific padding strategies are applied to compute the pixels on the borders.

Based on the weights specified in the kernel, the convolution operation produces different output images and extract specific features. In this work we consider the following filters:

Gaussian: removes the noise and details from the input image. The kernel has decreasing weights from the central position to the borders, so to provide a blur effect preserving the edges.

Mean: similar to the Gaussian filter; all weights in the kernel are equal, thus providing a slightly different smoothing result.

Median: another denoising filter; it replaces each pixel value with the median value of the neighbouring pixels. It is a non-linear filter because it does not apply Equation 1.

Sobel: an edge detection filter. It highlights the regions of an image that have a clear change in intensity or colour. This filter computes the gradient of the brightness for each pixel and it detects the edges by analyzing the direction and the velocity of the light variations.

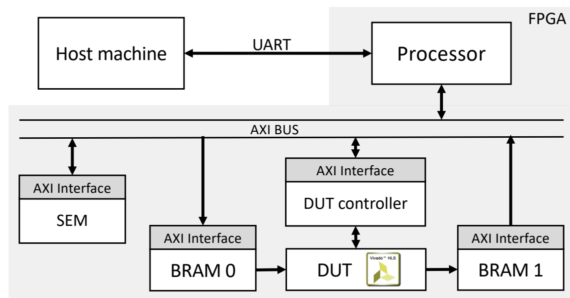


Figure 1: The fault injection framework.

Laplacian: another edge detection filter based on the application of a 2D isotropic measure of the second spatial derivative of an image.
Sharpen: emphasizes the details and enhances the edges of all the objects in the image.

IV. Fault Injection Framework

To run the experimental campaigns we implemented the fault injection framework depicted in Figure 1 onto a Xilinx FPGA device. We exploited the standard fault injection architecture; the DUT, i.e., the filter under analysis is connected to two different BRAM blocks, to read from and save to the input image and the produced results, respectively. A custom control module is used to drive the *clock*, *reset* and *start* signals and to monitor the *done* output of the DUT. Fault emulation is performed by means of the Xilinx SEM IP which injects bit-flips in the configuration memory. All the components are connected through the AXI bus to a microprocessor in charge of controlling and managing the overall fault injection campaign, and communicating with the host machine through a serial port. A prototype of the system has been synthesized on a Xilinx Virtex-7 FPGA VC707 Evaluation Board (and it is also compliant with any Series-7 device).

The software layer of the fault injection framework is organized in a host subsystem and an FPGA subsystem. The former performs a preliminary analysis of the bitstream to select configuration bits to be flipped, based on Xilinx SEM guidelines and consequently plans the fault injection campaign, in terms of list of fault locations and injection clock cycles. The fault injection list together with the input image and the golden output are sent to the FPGA, which runs in sequence all the fault injection experiments. For each experiment, at the required injection time, the DUT clock is frozen to perform the injection, and the experiment is resumed to receive the final result. Results are preliminarily filtered by the FPGA to discard non-corrupted outputs thus forwarding to the host machine only the corrupted output images.

For each of the six considered filters we have designed a hardware accelerator by means of the commercial Xilinx Vivado HLS tool, starting from an algorithmic description in C language. More precisely, a basic sequential implementation of the algorithm has been obtained. Future work will be devoted at investigating how different implementation choices may affect the identified error models. For each one of the considered filters, we defined a campaign of 10,000 random fault injection experiments. Since SEUs in the FPGA configuration memory have permanent effects, we wanted to analyse faults' effects at the steady state; so injection time has been initially set to 0 for all the experiments. Then, we also analyzed the effect of faults injected at random time. We considered a dataset of 1,000 satellite images downloaded from Microsoft Bing Maps [14] and converted in gray scale at a resolution of 290x204 pixels.

V. Error Modeling

For each filter, we executed the planned experimental campaign; the corrupted output images have been collected and visually inspected. The fault injection campaign produced about 5791 corrupted images for the 60,000 injected faults. The goal of this phase is the identification of recurrent visual patterns of the corrupted images, which can then be used to define error models whenever they are data independent. Every time a visual pattern is statistically relevant, data independent and can be implemented by means of an algorithm, it leads to the definition of an image error model.

We partially automated this activity by highlighting the areas of each image presenting differences w.r.t. the golden counterpart; however, the actual identification of the visual patters of the errors has been manually performed. The final result of this modeling activity is the description of the identified error models in terms of an algorithm to reproduce their effects and their relevant aspects, that are occurrence probability and error intensity. This detailed information is at the basis of the library of error models to support the error simulation frameworks.

A. Visual Error Patterns

The analysis of the recurring visual error patters of the corrupted output images caused by the injected faults led to the definition of the following 15 classes:

Horizontal noise (label: *H Noise*). It is characterized by a constant width noise placed on a regular distance between the rows. This noise produces an alteration of colours, with a final effect of both darkening and brightening the pixels.
Vertical noise (*V Noise*). It is characterized by a constant width noise placed on a regular distance between the columns. The effect is the same one as the *H noise*.
Oblique noise (*O Noise*). It is a similar effect of colour alteration, affecting oblique lines rather than horizontal/vertical lines.
Horizontal shift (*H Shift*). A different kind of horizontal noise, but in this case, the effect is an image shift. For this specific error model, the shift is horizontal.
Vertical shift (*V Shift*). Similar to the “Horizontal shift” but in the vertical direction.
Oblique shift (*O Shift*). Similar to the “Horizontal shift” but with an oblique direction.

Corrupted colors (*Colors*). It is characterized by an alteration of the colors and it can involve the whole image or a portion of it.

Few corrupted pixels (*Pixels*). It is characterized by a small number of pixels that have a different value, either clustered in a region to define a regular pattern, or scattered in random positions, as the case of the salt & pepper noise.

Corrupted borders (*Borders*). It is characterized by an alteration of the image borders, due to a corruption of the zero-padding.

Flattened image (*Flattened*). It is characterized by the fact that the output is flattened so fitting only part of the entire image; the remaining part of the output is either black (the 0-initialized BRAM content not overwritten by the computed output) or it contains part of the image processed in the previous run.

Points texture (*Texture*). It is characterized by a point-like texture overlay of the image, either for the entire image or for areas of it.

Corrupted patches (*Patches*). It is characterized by portions of the image that are either black (the 0-initialized BRAM content not overwritten by the computed output) or other parts of the image erroneously repeated, affected a limited portion of the entire image.

Mix of models (*Mix*). Two or more of the previous patterns can be identified on the same image, with a combination of effects.

Disruptive (*Disruptive*). It is characterized by a relevant part of the image that differs from what expected. The impact is disruptive and no single pattern actually prevails. The common aspect is the high number of corrupted pixels in the output.

Other (*Other*). It includes all visual patterns that are either not easily identifiable and reproducible in an algorithmic way, or are statistically not relevant because only a limited number of fault/input/time combinations can produce such error.

The complete list of the identified classes of visual error patterns is reported in Table I, together with their frequency of occurrence for each filter, to show their statistical relevance.

An example of these visual effects for the Sharpen filter is shown in Figure 2. Figure 2a is the golden output of the filter, in a fault free condition, the remaining images are the collected corrupted outputs resulting from the fault injection campaign, one for each one of the listed classified visual effects.

B. Error model definition

As mentioned, the image error models need be generic, input independent and statistically relevant, and finally they must be defined in terms of an algorithmic implementation, so that they can be included in an error simulation framework. Therefore, the first thirteen classes of visual error patterns have been used to define the image error models, implemented in terms of an algorithm capable at applying the identified visual pattern to the output of a filter. Although the *Disruptive* error

Table I: Visual pattern frequencies (10,000 injected faults, 1,000 input images)

Patterns	Gaussian	Mean	Median	Sobel	Laplacian	Sharpen
H Noise	0.76%	0.88%	2.94%	2.84%	2.4%	4.44%
V Noise	0.38%	0.88%	4.12%	4.32%	2%	1.83%
O Noise	6.29%	0.75%	1.18%	0.62%	0.6%	2.6%
H Shift	0.38%	1%	2.35%	6.6%	1.7%	2.41%
V Shift	0%	0%	1.18%	0.99%	0.7%	1.64%
O Shift	1.9%	0%	0.88%	0.56%	0.8%	1.64%
Colors	46.48%	72.56%	39.71%	25.19%	49.2%	26.85%
Pixels	4.76%	3.76%	5.88%	1.85%	2.1%	3.95%
Texture	9.71%	6.02%	5%	0.74%	0.9%	4.23%
Borders	0.38%	0%	0.59%	0%	0%	0.38%
Patches	0.76%	0.13%	3.53%	6.36%	1.3%	3.27%
Flattened	0%	0.13%	1.76%	0.99%	0.2%	0.19%
Mix	6.86%	2.01%	6.47%	7.1%	3.39%	5.49%
Disruptive	7.62%	5.01%	14.41%	30.55%	25.75%	31.28%
Other	13.52%	6.89%	10%	8.52%	8.98%	8.37%
# samples	583	799	688	1620	1002	1089

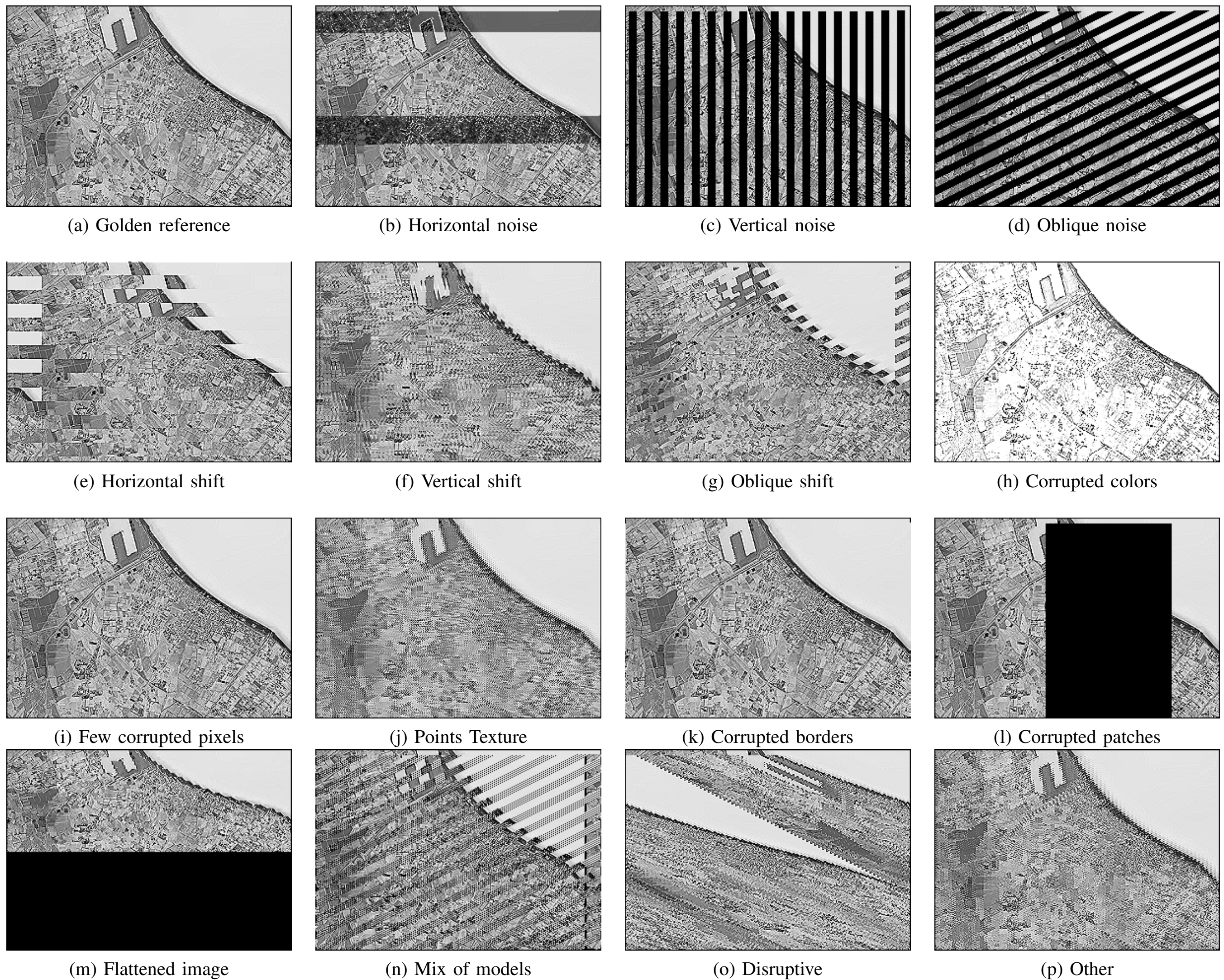
patterns are not statistically irrelevant for some filters, it is not possible to identify the effect of the fault and to reproduce it with an algorithm. However this is not an issue, in our opinion, since the actual output is completely "destroyed". Therefore, a functional analysis of the effects of this error in a complete application pipeline would cause an "unusable" final output.

As far as the *Other* class of visual error patterns is concerned, it includes a number of cases that is below 10%. This allows us to state that the relevance of an error simulation campaign based on the sole error models is relevant of an early and accurate analysis of the reliability of the application.

The different frequencies of occurrence of the various error patterns allow to state that the identified set of models is a superset, and for some filters some models do not apply, as for instance the *Borders* model for the Mean and Sobel filters. For other filters, the statistical relevance might be limited. Indeed, the additional information on the frequency of occurrence allows us to support the implementation of a library of error models that are general and can be extended in the future.

We already commented on the generality of the error models w.r.t. the specific filter, supported by the frequency of occurrence of each visual pattern. The selected classes can support the definition of proper and general error models provided they are independent of the fault injection campaign, that is i) the input image, and ii) the injection time. To this end, we re-run all the fault injection campaigns used to collect the visual error patterns by changing the input image in a first campaign and injection time in a second one. In the former experiment 1,000 new input images have been used; in 98% of the experiments we observe the occurrence of the same error models. In the latter we randomly selected the injection time; results confirm that in 99% of the cases the error model does not change. Finally, we also defined a completely new campaign for each filter, injecting 120,000 new random faults, leading to the same results with an error below 1%.

As a conclusion, we argue that the effect of a fault and the consequent error model depend only on the specific location of the FPGA configuration memory where the fault is injected.



(a) Golden reference (b) Horizontal noise (c) Vertical noise (d) Oblique noise

(e) Horizontal shift (f) Vertical shift (g) Oblique shift (h) Corrupted colors

(i) Few corrupted pixels (j) Points Texture (k) Corrupted borders (l) Corrupted patches

(m) Flattened image (n) Mix of models (o) Disruptive (p) Other

Figure 2: Visual patterns for the corrupted images when executing the Sharpen filter.

C. Error Model Parametrization

When analysing all the corrupted output images belonging to the same class, the same pattern may present variable intensities. As an example, Figure 3 depicts several output images corrupted by the *Vertical Noise* effect, for the output of the Sharpen filter. The number of bars and their width vary from one thin black line up to several thin lines and large bars. These aspects can be exploited to parametrize the error model, to make it more flexible, and to allow its integration in an error simulation engine. Table II lists the final list of defined error models and the associated parameters when available.

Table II: Error model parameters

Error model	Parameters
H Noise	# bands, band width, band color, corrupted region coords.
V Noise	# bands, band width, band color, corrupted region coords.
O Noise	# bands, band width, band color, corrupted region coords.
H Shift	# bands, band width, shift offset, corrupted region coords.
V Shift	# bands, band width, shift offset, corrupted region coords.
O Shift	# bands, band width, shift offset, corrupted region coords.
Colors	color intensity
Pixels	# pixels
Texture	texture color
Borders	–
Patches	corrupted region coords, corruption type: black/repetition
Flattened	corruption type: black/repetition
Mix	selected models

VI. Exploitation of the Error Models

The identified image error models and their parameters are implemented in terms of an algorithmic description. The designer may pre-characterize a repository of error models specific for the classes of filters to be employed in the application being developed and for the target hardware platform. Such models can then be used for an early reliability analysis of overall application through error simulation.

More precisely, given an image processing application such as the one in the upper part of Figure 4, the error simula-