Classification-based Improvement of Application Robustness and Quality of Service in Probabilistic Computer Systems *

Andreas Heinig[1], Vincent J. Mooney[2,3], Florian Schmoll[1], Peter Marwedel[1], Krishna Palem[2,4], and Michael Engel[1]

[1] Department of Computer Science, TU Dortmund, Germany
[2] Inst. Sustainable and Applied Infodynamics, NTU, Singapore
[3] School of ECE, Georgia Institute of Technology, Atlanta, GA, USA
[4] Department of Computer Science, Rice University, Houston, TX, USA

Abstract. Future semiconductors no longer guarantee permanent deterministic operation. They are expected to show probabilistic behavior due to lowered voltages and shrinking structures.
Compared to radiation-induced errors, probabilistic systems face increased error frequencies leading to unexpected bit-flips. Approaches like probabilistic CMOS provide methods to control error distributions which reduce the error probability in more significant bits. However, instructions handling control flow or pointers still require determinism, requiring a classification to identify these instructions.
We apply our transient error classification to probabilistic circuits using differing voltage distributions. Static analysis ensures that probabilistic effects only affect *unreliable* operations which accept a certain level of impreciseness, and that errors in probabilistic components will never propagate to critical operations.
To evaluate, we analyze robustness and quality-of-service of an H.264 video decoder. Using classification results, we map unreliable arithmetic operations onto probabilistic components of an ARM-based simulator, while remaining operations use deterministic components.

1 Introduction

Future electronic components for embedded systems will increasingly use lowered supply voltages and shrinking structure sizes. The positive effects of this technology scaling, lowered energy consumption and reduced costs, however, do not come for free. These semiconductor circuits will be susceptible to faults due to electromagnetic noise to a much greater degree than current devices, often resulting in erroneous program execution or system crashes. In order to obtain acceptable fabrication yields, it is necessary not to reject chips with a certain level of error. Thus, the decades-old assumption of deterministic operation of a computer will no longer be valid. Future chips will exhibit *probabilistic behavior*.

* This work is supported by the DFG SPP1500 priority program under grant no. MA943/10-1

Recently developed technologies like probabilistic CMOS (PCMOS) [4, 18–20] control the error distribution in order to reduce the probability of errors showing up in more significant bits of a data word. Using Biased Voltage Scaling (BIVOS), different voltage distributions are employed to achieve this effect [9].

This leads to a new fault model that is not yet considered in fault-tolerance approaches. These models assume a comparatively low error rate and a uniform distribution of faults over all components of a semiconductor. Using probabilistic components an adapted fault-tolerance approach can benefit from the fact that the *locations* of faults and their distribution are well-known.

However, only a certain subset of all arithmetic operations performed by a microprocessor can be safely mapped onto a probabilistic arithmetic component. While this is feasible for typical embedded signal processing applications such as calculations in audio and video decoders, other instructions cannot tolerate imprecision in the result. Some obvious examples for these are address calculations for branch targets or pointer arithmetic when accessing array elements. The difference is that an imprecise result in a signal processing operation will only lead to a decreased output quality (which may, depending on the quality and compression ratio of the input signal, not even be visible), whereas a fault in the latter case would most probably result in a system crash.

In order to distinguish between these error classes, we apply a classification approach we have previously developed for classifying the effect of transient, radiation-induced errors. Using the results of a static analysis of the application source code, the classification determines for each operation if the operation can accept imprecise results or not. It has already been shown that this approach can improve the resilience of embedded systems against transient errors [11], but it has so far not been applied to probabilistic systems. For transient errors, the classification is used to decide which error correction method to apply. In contrast, in probabilistic systems, the classification gives hints which machine-level operations can be mapped to probabilistic arithmetic functions and which have to be performed in a reliable way.

The main contributions of this paper are as follows:

1. We evaluate the effects of probabilistic behavior of semiconductors on the robustness and quality of service provided by a real-world application,
2. we show that a mapping using static analysis results can mitigate the effect of otherwise fatal errors,
3. and we show unexpected effects of different voltage scaling methods on the quality of service (QoS) and devise an approach to improve the QoS while continuing to use probabilistic components.

The rest of the paper is organized as follows. Section 2 gives an overview of PCMOS, its error models, and its implementation in the context of the MPARM simulation platform. Section 3 describes our static analysis method and the target H.264 video decoder application. Section 4 presents evaluation results focusing on the robustness and QoS provided by the H.264 decoder application under a probabilistic error assumption. Section 5 discusses related work, followed by conclusions and an outlook onto future research challenges in Section 6.

2 PCMOS

The notion of probabilistic CMOS (PCMOS) was first introduced by Palem [20] in the context of probabilistic bits (PBITS) and probabilistic computing [19]. Briefly, the idea is to allow previously deterministic Boolean bits to have a *probability* of being a zero or a one. Thus, logic functions have probabilistic outputs instead of deterministic outputs (deterministic bits). In the context of computation based on silicon, one possible prediction of future PCMOS behavior is based on thermal noise [24, 2]. In this section, we describe the probabilistic components considered in this paper and their use in a system simulation environment.

2.1 Component Models and Probabilistic Error Model

We consider probabilistic behavior of adders and multipliers. As basic component for building multi-bit adders and multipliers, we use the three-stage model for probabilistic full adders (PFA) described in [24] (models 1–6). Based on logic paths, these models describe the effect of distinct loads per output of a gate. These models yield fast simulation time but are within 7%-8% accuracy of more complex SPICE-based models. However, these more complex simulations take orders of magnitude more time to execute. In other words, the error rates calculated with the three-stage models of [24] are fairly accurate and fast to compute, which enables their use in a full system simulation of a complex application.

Supply Voltage Schemes. One important property of the probabilistic components considered here is that each low-level component, like a single-bit probabilistic full adder, can be supplied with a different voltage, causing a difference in its susceptibility to noise. When combining single-bit PFAs to form larger circuits, this leads to various non-uniform *biased voltage scaling* (BIVOS) schemes.

The BIVOS schemes considered here provide more significant bits with a higher voltage than less significant bits, so that the probability of noise-induced errors in more significant bits of a word is reduced. For small benchmarks, [4] and [18] show that using BIVOS the accuracy of the probabilistic ripple carry adder described below can be increased compared to *uniform voltage scaling* (UVOS), where supply voltage is reduced for all bits equally, so that they have the same suspectibility to noise. However, this has not been analyzed in the context of a large real-world application. The qualitative and quantitative results of our evaluation are described in Section 4.

Adder Implementation. The probabilistic adder considered in our system is a probabilistic ripple carry adder (PRCA). The PRCA simulation uses different models of the PFA, depending on the different output loads of each full adder in the overall circuit. For clarity, Fig. 1 shows a four-bit adder instead of the 32-bit adder actually used. In order to construct a PRCA using the three-stage-model, three different PFA models are required. The PRCA simulation starts with the PFA calculating the least significant bit s_0 on the right hand side. The sum and carry bits are calculated deterministically. Thereafter, probabilistic behavior is modelled by bit-flips on the interconnections as shown in Fig. 1. These bit-flips will occur according to the error probabilities $p_i(m_j, v_k)$ determined by the SPICE simulation. Here, the probability depends on the PFA model and the configured supply voltage.

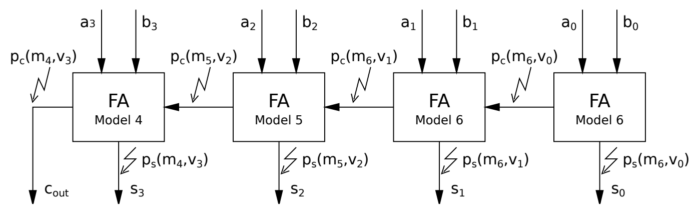


Fig. 1. Simulated PRCA

Multiplier Implementation. As multiplier we use is a probabilistic version of a Wallace tree multiplier (PWTM), as shown in Figure 2. Like the PRCA, the PWTM is constructed from multiple probabilistic full adders. For clarity, we only show an four-bit multiplier and do not indicate error injection. Bit-flips occur analogously to the PRCA case. Each PFA can be supplied with a different supply voltage V_i and uses a PFA model M_j according to the specific output load, enabling the analysis of different BIVOS configurations.

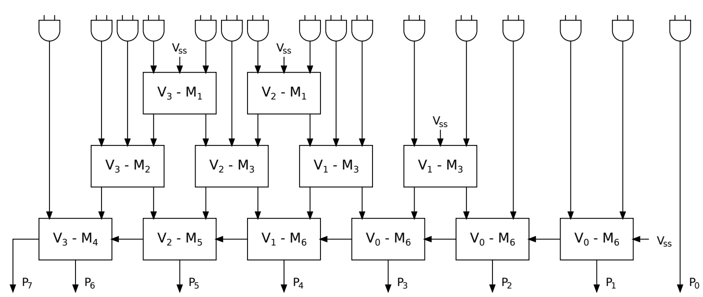


Fig. 2. Simulated PWTM

2.2 Implementation in MPARM

In order to perform an analysis of a complex real-world application, an execution platform for the application binary is required. Here, we extended the MPARM ARMv3m architecture simulator [1] to include PRCA and PWTM components in the CPU core in addition to the standard deterministic ALU and multiplier. Four new instructions of the simulated MPARM CPU core use the probabilistic components, whereas all other instructions continue to use deterministic components only. The new instruction are addition (`padd`), subtraction (`psub`), and reverse subtraction (`prsb`) using the PRCA, as well as multiplication (`pmul`) using the PTWM described in Section 2.1

3 Annotations and Static Analysis

If probabilistic behavior of system components is to be expected, the developer writing software for such a platform has to be enabled to *control* the implications of using these probabilistic components. In this section, we describe how the notion of *reliable* and *unreliable* type qualifiers for annotating data objects of a C program, already successfully employed for handling transient errors [7], can be used in case of probabilistic behavior of well-known components.

In order to indicate the error tolerance of a variable or other data object, like a structure in a C program, it has to be annotated. The annotations indicate if the data contained in a variable or data structure is expected to be **reliable** – i.e., deterministic behavior is required – or **unreliable**. In the latter case, probabilistic calculation results assigned to a data object can be tolerated since it will have no fatal consequences like abnormal program termination. However, such an operation may influence the quality of the generated output.

To avoid fatal consequences and unintentional propagation of errors in an application to a reliable data object, the use of unreliable data objects is restricted using a compile-time static analysis approach. Basic semantic rules governing the analysis are described in detail in [7]. Summarized, the basic rules prohibit the assignment of unreliable data objects to a data object with reliable data. In addition, it must be ensured that unreliable operations do not affect the control flow. Thus, the analysis restricts the use of probabilistic expressions in *if* and *loop conditions*. A third class of critical operations in C use pointers or array indices. These may also not use probabilistic expressions. Several other conditions, like avoiding probabilistic divisors, are considered in addition.

Accordingly, the source code of our H.264 decoder was extended by *reliable* and *unreliable* annotations. As a starting point, the luminance and chrominance arrays of a video frame have been annotated as `unreliable`, as shown in Listing 1.1. By default, data without explicit annotation is treated as reliable.

Listing 1.1. Frame data structure

```
typedef struct __frame {
    int Lwidth, Lheight, Lpitch;
    int Cwidth, Cheight, Cpitch;
    unreliable uchar * L, * C[2];
} frame;
```

To check compliance with the semantic rules we use our probabilistic C compiler *prob-cc*, a source-to-source compiler based on ICD-C [12]. Besides semantic rule checks, prob-cc also propagates reliability annotations along the control flow path. Additionally, prob-cc is able to determine further objects which can be safely annotated as unreliable according to the semantic rules described above.

An example of an annotated function is shown in Listing 1.2. This function is used to add a value generated by an inverse cosine transformation to a specific position in the frame buffer. This function computes a value that is guaranteed not to change the control flow. However, it may result in a change of the output data, i.e., a disturbance of the decoded video frame.

Listing 1.2. Function example

```
void enter(unreliable uchar *ptr, unreliable int q_delta) {
    unreliable int i = *ptr + ((q_delta + 32) >> 6);
    *ptr=Clip(i);
}
```

In an additional step, prob-cc can transform C code with probabilistic annotations to code using the probabilistic instructions we added to MPARM. The converted form of the function shown in Lst. 1.2 is depicted in Lst. 1.3.

Listing 1.3. Code transformed by prob-cc

```
void enter(uchar *ptr, int q_delta) {
    int i = __paddsw((*ptr), (__paddisw(q_delta, 32) >> 6));
    *ptr = Clip(i);
}
```

Our compiler substitutes probabilistic operations with special macros using the related inline assembler instruction. For example, `__paddisw` performs a probabilistic add of a signed word with an immediate value.

4 Evaluation

4.1 Experimental Environment

We evaluate the influence of noise on the stability and quality provided by an H.264 video decoder application under different voltage distribution schemes for probabilistic adder and multiplier components. The H.264 decoder is annotated and compiled using prob-cc and executed on our extended MPARM simulator.

We simulate the decoding of a set of videos using UVOS schemes with voltage levels from 1.2 V to 0.8 V in steps of 0.1 V as well as different BIVOS schemes described below. In this paper, we assume Gaussian distributed noise. The RMS value for the noise is set to 0.12 V, 10% of the nominal supply voltage.

4.2 Qualitative Analysis: Applicability of Probabilistic Arithmetics

As a first step, we evaluated if a significant percentage of the instructions of a program are actually capable of being safely executed using our probabilistic adder or multiplier. Using MPARM, we counted the number of instruction executed reliable resp. unreliable. Table 1 shows the relative frequencies. Here, 76.27 % of `mul` instructions means that three quarters of all multiplications were computed using probabilistic components, whereas all other multiplications were computed using the deterministic ALU. In total, 13.36% of all operations were executed on probabilistic arithmetic components. This is a significant result, since this percentage considers all possible operations executed by the ALU[5] including logic and compare instructions.

[5] We count multiplication as ALU operation, even though it is not implemented as ALU instruction in the ARM architecture

The results also show that using reliability annotations, the control flow of the decoder is not altered. Thus, the application does not exhibit crashes or hangs in any of the benchmarks performed when using probabilistic arithmetics.

Table 1. Instructions executed using probabilistic components

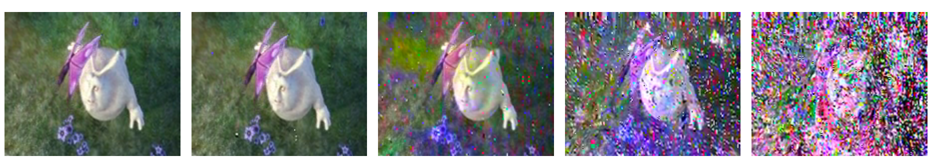
Instruction Type	add	sub	rsb	mul	overall
Executed using PRCA/PWTM	18.59%	18.60%	43.01%	76.27%	13.36%

4.3 Quantitative Evaluation: Signal-to-Noise Ratio Using UVOS

After showing that probabilistic operations can actually be used by a significant fraction of the H.264 decoder, the second step of our evaluation now considers the effect of noise on the output quality (in general, the quality of service) of the video decoder under different uniform supply voltages. The quality is evaluated using peak signal-to-noise ratio (*PSNR*) values for each decoded frame using probabilistic components compared to a correctly decoded frame:

$$PSNR = 10 \log_{10} \frac{2^B - 1}{WMSE} \, \texttt{[dB]}$$

Here, $WMSE$ denotes the weighted mean squared error between the frames, and B is the number of bits per sample. A higher *PSNR* value indicates better quality. A perfect video has a PSNR value of infinity. Commonly, a *PSNR* value of at least 35 dB is recognized as good quality. In contrast, a value of less than 25 dB indicates very poor quality. However, the interpretation of video quality and *PSNR* values depend on the perception of the viewer and the output quality requirements. The values indicated are accepted for consumer video applications.



(a) $V_{DD} = 1.2\,V$ (b) $V_{DD} = 1.1\,V$ (c) $V_{DD} = 1.0\,V$ (d) $V_{DD} = 0.9\,V$ (e) $V_{DD} = 0.8\,V$

Fig. 6. Uniform voltage scaling results

Figure 6 shows the results for one of our test videos using different voltages. Picture (a) shows the reference frame simulated at the nominal voltage. When lowering the supply voltage, the effects of noise are increasingly visible, leading to mostly indiscernible pictures at 0.9 V and 0.8 V in (d) and (e). Detailed PSNR values are shown in Fig. 8. It can be easily seen that a better quality is achieved using a higher supply voltage. Using UVOS, the PSNR values for 1.0 V are already below the acceptability limit of 25 dB.

4.4 Quantitative Evaluation: Signal-to-Noise Ratio Using BIVOS

Due to the disappointing results achieved using UVOS, it is interesting to analyze if employing BIVOS schemes provides a better quality using equivalent energy budgets as the UVOS schemes. The UVOS and BIVOS energy consumption is calculated with the energy model used by MPARM based on [22].

We consider three different BIVOS models, as shown in Fig. 7. The *PSNR* results for the various UVOS and BIVOS schemes evaluated are shown in Fig. 8.

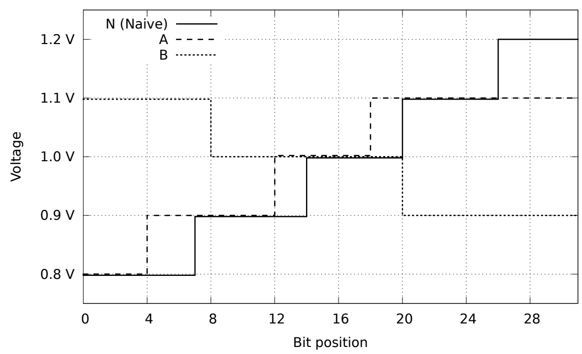


Fig. 7. BIVOS setups used

The first BIVOS scheme, naive BIVOS (N), supplies less significant bits with a low voltage and the most significant bits with the nominal supply voltage. Here, only a very low PSNR rate could be achieved.

Due to the fact that 1.1 V UVOS shows good PSNR values, we constructed a second BIVOS scheme (A). Here, we do not supply the most significants bit with the nominal voltage. Instead, we reduce this voltage to 1.1 V and spend the energy saved to increase the supply voltage of less significant bits. As shown in Fig. 8, this version achieves improved PSNR values using the same amount of energy as the 1.0 V UVOS scheme. However, the PSNR value is still quite poor.

Analyzing the H.264 code further, we discovered that most parts of the code do not actually use more significant bits of the 32 bit probabilistic adders and multipliers. Hence, we constructed a third BIVOS scheme B. Here, we supply the least significant bits with a higher voltage than the most significant bits. Again, we use the same amount of energy as the 1.0 V UVOS scheme. The PSNR values of this version are in fact better than all other BIVOS versions, but still worse than using the 1.0 V UVOS scheme.

4.5 Quantitative Evaluations: Summary

Figure 8 shows the *PSNR* values for our benchmark videos using the described UVOS and BIVOS schemes. Contrary to the micro benchmarks described in [4] and [18], applying probabilistic BIVOS components in the context of a real-world application *does not improve the output quality* under identical energy budgets. We tried to improve the PSNR by applying different simulated BIVOS schemes but we were not able to achieve the quality of the simple 1.0 V UVOS scheme.

We identified one reason for this phenomenon. It is caused by the H.264 specification when transferring a 32 bit integer into a 8 bit value to be stored in the frame buffer. In some parts, a simple clipping function (shown in Lst. 1.2) is used which implements saturation. This function restricts the value to 255 if the input is larger then 255. For BIVOS scheme B this implies that if, e.g., bit 11 flips, the precision of the less significant eight bits is irrelevant. In the opposite case, e.g., using BIVOS scheme A, the correct clipping is performed, but the least significant eight bits are too imprecise. For different operations, like the

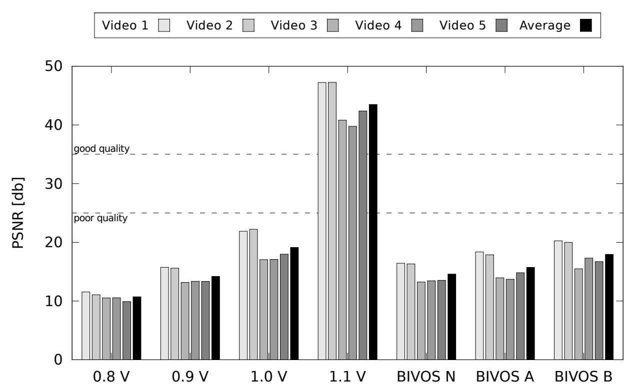


Fig. 8. PSNR values for simulated videos

selection of luminance and chrominance values for macro blocks or even larger parts of a frame, this effect is even worse.

Thus, the unexpected result of our quantitative analysis shows that due to the properties of H.264 videos, we are unable to find a BIVOS scheme that reaches 25 dB *PSNR* using a comparable amount of energy as the 1.0 V UVOS scheme. To the best of our knowledge, all recent papers which optimize the power distribution for BIVOS assume input values with a range that is uniformly distributed over the value range of the corresponding data type [13]. For H.264, this assumption does in many cases not hold.

Since it is unrealistic to assume that separate adders for different, commonly used data widths will be provided in future architectures, an analysis of the number of bits actually used in arithmetic operations is required. However, this implies further complications. An idea of an approach that combines bit-width analysis methods for arithmetic operations and code transformations to use bits with optimal supply voltage for the operation at hand is described in Sect. 6.

5 Related Work

The use of type qualifiers as type annotations has already been proposed by [8] and [5]. They present frameworks for the extension of typed programming languages by user defined type qualifiers. In this way, types are augmented with additional semantics that can be used to ensure invariants statically at compile-time. Also, they implemented tools that can infer type qualifiers to ease the annotation of the application. Hence, their work is very similar to ours. Nevertheless, they do not exploit type qualifiers for code generation.

In [23], type qualifiers for the allocation of data to low-power storage and processing units are described. Like PCMOS, these components consume less energy, but are potentially imprecise. Using the type qualifiers approximate and precise, they distinguish between data that may tolerate inaccuracies and that does not. A checker ensures that type qualifiers are used in compliance with rules that are similar to our semantic rules. Finally, approximate data is mapped to the low-power components. Energy savings of 10%-50% are reported, while the loss in quality of service highly depends on the assumed approximation strategy. In contrast to our work, annotations have to be added manually and a high-level simulation in Java is used for the evaluation.

PCMOS was first introduced by Palem [20] in the context of probabilistic computing [19]. Various methods for modeling thermal noise based probabilistic primitives like logic gates and adders have been developed [2, 10, 24]. Lau describes a mathematical approach to model probabilistic components [15]. Here, HSPICE simulations of simple PFAs are used to determine the probability of a bit flip in a larger PRCA. In [6], Dhoot describes a motion search algorithm based on probabilistic components. Kedem [14] uses data flow graphs to minimize expected errors in the FFT of a JPEG decoder for a given energy budget.

The impact of soft errors was studied for several applications by [16, 21, 11]. It could be shown that a large number of transient faults do not have any effect on application correctness. Another fraction of faults changes the output or state of the application, but causes no crashes while providing acceptable quality.

6 Conclusions and Future Work

In this paper, we presented an analysis of probabilistic effects on a real-world benchmark application. Using a processor model extended with probabilistic arithmetic components, we were able to avoid all application crashes due to probabilistic results by mapping only suitable operations onto the probabilistic components. A significant percentage of all arithmetic operations could be performed using probabilistic components, so our classification serves as an additional verification of the feasibility of using probabilistic components.

However, our experimental results also show an unexpected effect. The currently available BIVOS schemes are not guaranteed to improve the quality of service compared to a UVOS scheme using the same amount of energy. An analysis of the application identified a possible cause of this problem. Since the probabilistic arithmetic components use a fixed bit width (32 bits) using BIVOS distributions, the most visible effect on the output quality would only be achieved if the most significant bits were actually significant for the operation at hand. A profiling-based analysis on selected variables showed that the actual value range used was significantly smaller than 2^{32}. Often, two to ten of the most significant bits of a 32 bit unsigned integer variable contained no useful information.

This observation guides our future research in this area. We intend to extend our static analysis approach by methods that can determine the number of unused bits of probabilistic variables. Using this information under a BIVOS distribution, additions and subtractions could be performed by shifting the parameters by the unused number of bits minus one[6] to the left. For multiplications, the result may in general require twice as many bits as the largest operand. Thus, we expect shifted multiplications to have a lowered potential to improve the QoS.

In order to reliably employ shifted operations, several approaches seem useful. Obtaining the maximum bit width of a variable is possible using either safe static approaches [3, 25] or heuristic approaches [17], both of which are in common use when optimizing bus widths in semiconductor synthesis. It depends on the application whether an overapproximation of the bit range or a cutoff of the most significant bit(s) will have a larger effect. Our static analysis should thus be extended by one of these approaches.

It is obvious that probabilistic behavior can have different effects on operations, even when only considering those operations that can accept imprecise results. We will have to extend our annotations by changing the current binary error impact model (crash/no crash) to include more precise information on the QoS impact of an error. A model similar to a probability distribution (numbers in an interval from 0 to 1) could be used to indicate QoS impact. This would be compatible with the current semantics. An impact factor of 1 would be the worst possible impact (application crash or hang leading to a service failure), whereas a value of 0 would indicate that no visible QoS impact is to be expected. This model is, in turn, also expected to be useful for transient error models in order to obtain more detailed information on the urgency of error correction.

References

1. Benini, L., Bertozzi, D., Bogliolo, A., Menichelli, F., Olivieri, M.: MPARM: Exploring the Multi-Processor SoC Design Space with SystemC. The Journal of VLSI Signal Processing 41, 169–182 (2005)
2. Bhanu, A., Lau, M.S.K., Ling, K.V., Mooney, V.J., Singh, A.: A more precise model of noise based CMOS errors. In: Proc. Intl. Symp. on Electronic Design, Test and Applications. pp. 99–102 (Jan 2010)
3. Budiu, M., Sakr, M., Walker, K., Goldstein, S.: Bitvalue inference: Detecting and exploiting narrow bitwidth computations. In: Proc. of Euro-Par 2000 Parallel Processing, LNCS, vol. 1900, pp. 969–979. Springer (2000)
4. Chakrapani, L., Muntimadugu, K., Lingamneni, A., George, J., Palem, K.: Highly energy and performance efficient embedded computing through approximately correct arithmetic. In: Proc. of CASES. pp. 187–196. ACM (2008)

[6] Binary addition results may require one more bit than the largest operand.

5. Chin, B., Markstrum, S., Millstein, T., Palsberg, J.: Inference of user-defined type qualifiers and qualifier rules. In: Programming Languages and Systems, Lecture Notes in Computer Science, vol. 3924, pp. 264–278. Springer (2006)
6. Dhoot, C., Mooney, V.J., Chau, L.P., Chowdhury, S.R.: Low power motion estimation with probabilistic computing. In: Proc. ISVLSI. pp. 176–181. IEEE (2011)
7. Engel, M., Schmoll, F., Heinig, A., Marwedel, P.: Unreliable yet useful – reliability annotations for data in cyber-physical systems. In: Proc. of the Workshop on Software Language Engineering for Cyber-physical Systems. Berlin (Oct 2011)
8. Foster, J.S., Fähndrich, M., Aiken, A.: A theory of type qualifiers. In: Proc. of PLDI. pp. 192–203. ACM, New York, NY, USA (1999)
9. George, J., Marr, B., Akgul, B., Palem, K.: Probabilistic arithmetic and energy efficient embedded signal processing. In: Proc. of CASES. pp. 158–168. ACM (2006)
10. Gupta, A., Mandavalli, S., Mooney, V.J., Ling, K.V., Basu, A., Johan, H., Tandianus, B.: Low power probabilistic floating point multiplier design. In: IEEE Computer Society Annual Symposium on VLSI (ISVLSI). pp. 182–187 (2011)
11. Heinig, A., Engel, M., Schmoll, F., Marwedel, P.: Improving transient memory fault resilience of an H.264 decoder. In: Proc. of ESTIMedia. IEEE (Oct 2010)
12. ICD e.V.: ICD-C Compiler framework, http://www.icd.de/es/icd-c/
13. Kedem, Z., Mooney, V., Muntimadugu, K., Palem, K.: An approach to energy-error tradeoffs in approximate ripple carry adders. In: Proc. Intl. Symposium on Low Power Electronics and Design (ISLPED). pp. 211 –216 (Aug 2011)
14. Kedem, Z., Mooney, V.J., Muntimadugu, K.K., Palem, K., Devarasetty, A., Parasuramuni, P.D.: Optimizing energy to minimize errors in dataflow graphs using approximate adders. In: Proc. of CASES. pp. 177–186. ACM (2010)
15. Lau, M.S.K., Ling, K.V., Bhanu, A., Mooney, V.J.: Error rate prediction for probabilistic circuits with more general structures. In: Proc. of the WS on Synthesis And System Integration of Mixed Information technologies. pp. 220–225 (Apr 2010)
16. Li, X., Yeung, D.: Application-level correctness and its impact on fault tolerance. In: Proc. Symp. on High Performance Comp. Architecture. pp. 181–192 (2007)
17. Özer, E., Nisbet, A.P., Gregg, D.: A stochastic bitwidth estimation technique for compact and low-power custom processors. ACM TECS 7, 34:1–34:30 (May 2008)
18. Palem, K., Chakrapani, L., Kedem, Z., Lingamneni, A., Muntimadugu, K.: Sustaining moore's law in embedded computing through probabilistic and approximate design: retrospects and prospects. In: Proc. of CASES. pp. 1–10. ACM (2009)
19. Palem, K.: Energy aware algorithm design via probabilistic computing: From algorithms and models to Moore's Law and novel (semiconductor) devices. In: Proc. of CASES). pp. 113–116 (Sep 2003)
20. Palem, K.: Energy aware computing through probabilistic switching: A study of limits. IEEE Trans. Computers 54(9), 1123–1137 (Sep 2005)
21. Polian, I., Becker, B., Nakasato, M., Ohtake, S., Fujiwara, H.: Low-Cost Hardening of Image Processing Applications Against Soft Errors. In: Proc. of the Intl. Symp. on Defect and Fault-Tolerance in VLSI Systems. pp. 274–279. IEEE (2006)
22. Pouwelse, J., Langendoen, K., Sips, H.: Dynamic voltage scaling on a low-power microprocessor. In: Mobile Computing and Networking. pp. 251–259 (2001)
23. Sampson, A., Dietl, W., Fortuna, E., Gnanapragasam, D., Ceze, L., Grossman, D.: EnerJ: approximate data types for safe and general low-power computation. In: Proc. of PLDI. pp. 164–174. ACM, New York, NY, USA (2011)
24. Singh, A., Basu, A., Ling, K., Mooney, V.: Modeling multi-output filtering effects in PCMOS. In: Symp. on VLSI Design, Automation and Test. pp. 1–4 (Apr 2011)
25. Stephenson, M., Babb, J., Amarasinghe, S.: Bitwidth analysis with application to silicon compilation. In: Proc. of PLDI. pp. 108–120. ACM, New York, USA (2000)