Fast, Approximate Error Prediction for Unreliable Embedded Processors

Zheng Wang*, Saumitra Chafekar*, Hui Xie*, Anupam Chattopadhyay†
*UMIC Research Centre, RWTH Aachen University, Aachen, Germany
wang@umic.rwth-aachen.de
†Nanyang Technological University, Singapore
anupam@ntu.edu.sg

***Abstract*—Reliability has emerged as an important design criterion due to shrinking device dimensions. To address this challenge, researchers have proposed techniques compromising the Quality-of-Service (QoS) across all design abstractions. Performing reliability-QoS trade-off from a high-level design abstraction is a major challenge. In this paper, we propose an analytical reliability evaluation framework, based on probabilistic error masking matrices (PeMMs). The reliability evaluation is performed by propagating erroneous tokens through an abstract circuit model. We report detailed experiments using a RISC processor and several embedded applications. The proposed approach demonstrates significantly faster reliability evaluation compared to pure simulation-driven approach.**

***Keywords*-Fault Propagation Analysis; Error Prediction; Approximate Computation**

I. Introduction

With reduced transistor size in deep sub-micron CMOS technology, the performance and power density of processor have increased dramatically. High power density on the semiconductor chips increases failure rate and consequently reduces device lifetime as well as causes soft errors [1]. Furthermore, soft errors are caused by external radiations which are increasingly reported even at ground level [2]. These trends forced digital designers to treat reliability as a serious design concern.

Two major categories of faults, namely transient and permanent faults, lead to unreliability of a design [3]. *Transient faults* manifest into *soft errors* which temporarily corrupt the data or the output of a combinational circuit. On the other hand, *permanent faults* lead to permanent damage of the circuit and manifest as *hard errors*. Permanent faults are caused by *extrinsic* sources to the design such as manufacturing defects, process variation or *intrinsic* sources, which are caused by wearing out of the design material.

A. Reliability Estimation Techniques

The impact of faults can be investigated through simulation. While faults can be simulated accurately only at the circuit level of abstraction, many proposals exist to inject the fault at high level of abstraction for early reliability exploration. Pure software fault injection techniques alter the processor state (memory, register) to simulate a fault, however, it suffers from a restricted view of the micro-architecture. Register-Transfer Level (RTL) and gate-level fault injection approaches [4] simulate the hardware behavior with more accuracy and therefore, are usually slow and in some cases need repeated compilation. In order to hit a balance, fault injection can be performed during instruction-set simulation (ISS) [5], which offers different degrees of accuracy and speed trade-off. Cho [6] presents a quantitative evaluation on the accuracy of different fault injection techniques by tracking error propagation using FPGA-based emulation system, where the inaccuracy analysis is conducted through exhaustive fault injection experiments.

In contrast to the simulation techniques, analytical methods have also been proposed to investigate fault tolerant behavior of circuits. Mukherjee et al. [7] introduced the concept of architecturally correct execution (ACE) to compute the vulnerability factors of faulty architecture components. In [8], ACE analysis is performed to compute architectural vulnerability factors for cache and buffers. Reliability-aware software transformations are proposed by Rehman et al. [9], [10] to use the ACE for instruction vulnerability analysis. The vulnerability of the instruction is analyzed by investigating the constituent logic blocks. While the instruction vulnerability index model proposed in [9] includes the logical masking effects, the details of derivation of the masking effect are not mentioned. Generally, analytical methods suffer from estimation accuracy.

B. Importance of Error Prediction

With reliability evaluation techniques, design of reliable system in presence of faults, is still a challenging problem. Designing a reliable processor requires thorough understanding of all the causes of failures such as external radiation, electromigration and thermal cycles. Furthermore, reliability brings forth trade-off with other design dimensions [11], [12], [13], [14]. In recent research reliability is treated as a *cross-layer design issue* [15]. This stresses the fact that separate error mitigation techniques from individual design abstractions may result in an over-protected system. The design should take the support of architectural and algorithmic error resilience [16], [17]. However, this requires strong understanding of the fault propagation through different design abstractions, based on which resultant error properties such as location, timing and probabilities could be predicted. Such knowledge are difficult to acquire through analytical or fault injection techniques [6].

In particular approximate error prediction is an issue when *algorithmic reliability* is explored or when *inexact, probabilistic computing* [18] is performed. Similar research was pursued earlier for floating-to-fix point conversion in DSP design [19]. However, there the error localities were restricted to variables (sizes of fixed points) and operators (saturation, rounding effects) without any architecture-level concern. Krishnaswamy et al. proposed a framework called Probabilistic Transfer Matrix (PTM) [20] which captures the probabilistic behavior of the circuit while estimates the approximate error probability of faults **inside** the circuit. Analytical study of error propagation could be potentially addressed using PTM. However, PTM suffers from scalability problem for large design due to its bit-level accuracy and is not initially designed for handling error masking effects.

C. Contribution

We first develop the concept of a new algebraic representation named Probabilistic error Masking Matrix (PeMM) to investigate the masking effects on errors occurring at the **inputs** of the circuits. Compared with PTM, PeMM has a reduced calculation complexity due to the scope of error focusing on coarse-grained signal level. Next we integrate PeMM algebra into a high-level processor design framework and represent logic error as an abstract data structure named *token*. In this paper, we focus on **soft error** occurring at registers and memories, which are more susceptible to transient fault than combinational logic circuits [21]. An automated analysis flow is presented where the token propagation can be predicted using cycle-accurate instruction-set simulator while the error masking effects are carefully addressed using PeMM for individual micro-architecture unit. Fine-grained PeMM is also proposed which calculates nibble-wise or byte-wise error probabilities on data signals. Consequently, the significance of logic faults through design abstractions could be approximately predicted in earlier design phases.

The rest of the paper is organized as following. Section II illustrates the PeMM algebra in details. Section III introduces the approximate error prediction framework in a high-level processor design environment. Section IV shows the usability of proposed framework using several embedded applications on a RISC processor. The paper is concluded and future work is outlined in section V.

II. Probabilistic Error Masking Matrix

Faults within logic circuits are masked with certain probability before propagating to the circuits' outputs as visible errors. Such masking effects are seen due to several reasons as:

- Logic primitives performing algorithmic calculations have inherent ability of masking faults at inputs, which give error-free outputs.
- Micro-architecture features such as inter-stage data bypass can neglect the faulty input by replacing it with fault free input as a feedback from other pipeline stages.
- Faulty resources of processor such as registers and memory elements can never be read by computational circuit, giving always a fault free output.
- The faulty value of storage element or wires are overwritten before being read.

PTM [20] calculates the error probability of circuits' output by considering the logic circuits as a white box with faults inside logic gates. The approach suffers from scalability problem for large circuits since the size of the PTM is $2^n \times 2^m$ where n and m imply the total number of bits for inputs and outputs. Besides, for large scale circuits the derivation of PTM can be extremely time consuming since PTMs of individual logic gates needs to be accumulated.

We introduce *Probabilistic error Masking Matrix* (PeMM) to address the scalability issue, where the faults reside in inputs of circuits only. In contrast to PTM, PeMM has the size of $m \times n$ for a circuit with n bits input and m bits output. The size of matrix can be further reduced depending on the level of error existence. For instance, n and m represent number of input and output signals when signal level error existence is considered.

A. PeMM Definition

Consider a circuit with n inputs and m outputs. We label the n inputs as $in_0, ...in_{n-1}$ and the m outputs as $out_0, ...out_{m-1}$. The PeMM P of the circuit is a matrix with dimension $m \times n$. Each entry in $P(out_i, in_j)$ represents error masking probability $M_{in_j}^{out_i}$, where $i \in [0, m-1]$ and $j \in [0, n-1]$. It shows the error masking effect on output out_i with regard to input in_j, where 0 means the error has been completely masked while 1 implies no masking effect at all. Note that $e_{out_i} \in [0, 1]$ so that value larger than 1 will be truncated. The inputs of the circuits are represented by a column vector I with dimension $n \times 1$. Entry in $I(j)$ represents the error probability e_{in_j} associated with input in_j. When the input vector is left multiplied with P, resultant output vector is represented by a column vector O with dimension $m \times 1$, whose entry shows the error probability e_{out_i} associated with output of circuits. Figure 1 visualizes the abstract circuit model with its PeMM.

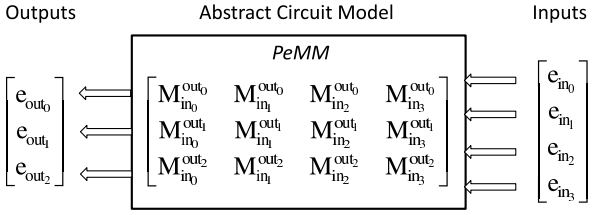


Fig. 1. Probabilistic error Masking Matrix (PeMM)

Besides the entire circuit, PeMMs can also represent error masking effects of individual micro-architecture units. Such divide and conquer approach considers the circuit PeMM as the concatenation of PeMMs for architecture units. Figure 2 shows the architecture units for executing the ALU instructions and data signals between logic operations. The dimensions of selected PeMMs are shown based on the counts of input and output signals. In case that input faults are not completely masked, the unit outputs errors with certain probability.

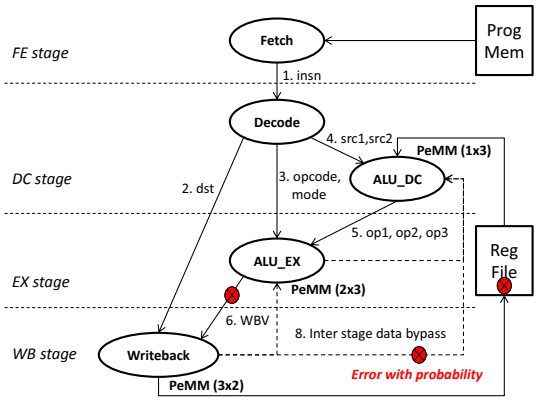


Fig. 2. Logic Blocks Involved for ALU Instruction

B. PeMM for Processor Building Blocks

1) Combinational Logic Blocks: PeMM performs a transformation of error masking probability from logic inputs to outputs for linear circuits. However, such approach is not applicable for logic blocks with internal data dependencies. To address such issue, larger circuits are decomposed into logic sub-blocks with individual PeMMs. Figure 3 indicates PeMM decomposition based on data dependencies, where the large logic block *alu_ex* is split into 3 sub-blocks. Signals *alu_in1* and *alu_in2* connect

first two sub-blocks while *alu_out* connects the last two sub-blocks. PeMMs for sub-blocks are characterized individually. The intra-token pool is used to keep the temporary tokens with error probabilities for processing by following logic sub-blocks.

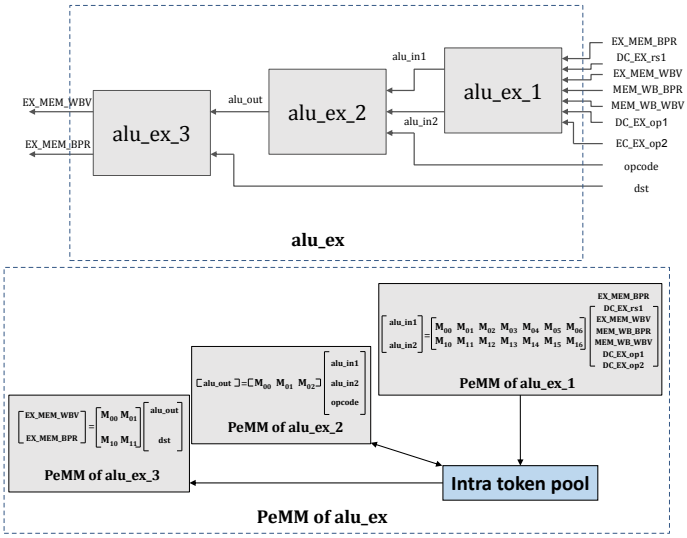


Fig. 3. Decomposition of Large Logic Block Using PeMM

2) Control Flow inside Logic Block: The other factor contributes to the inaccuracy of PeMM estimation is the dynamic control flow within logic blocks. The run-time circuits masking capability shows significant difference compared with purely random characterization. This can be seen from the behavioral description of circuits shown in Figure 4, where a 3-to-1 multiplexer is generated during logic synthesis for the if/else statements. During execution, various active path shows different PeMM for the same logic block which leads to exclusive elements in PeMM. Random characterization results in a PeMM elements of $[0.33\ 0.33\ 0.33]$ indicating error probability on each path is statically masked to 33%. To increase accuracy, additional *helper_signal* is adopted to indicate selected branch dynamically and fill the corresponding elements in PeMM. In Figure 4, vector $[1\ 0\ 0]$ is filled into PeMM when the first branch of if statement is selected. Such approach reviews trade-off between accuracy of characterization and modeling efforts.

3) Sequential Logic and Memory: Other than combinational logic, sequential logic and memory exhibit no internal error masking effects on their inputs. Such elements have equal number of inputs and outputs whose PeMM can be modeled using identity Matrix $I_{m\times m}$, where m is the number of inputs and outputs. For pipeline registers, errors on input ports are mapped consistently to corresponding output ports during pipeline shift. For registerfiles, input errors are stored for write access while same errors are loaded during read access. Similarly, PeMM for memory is modeled as identity matrix with m equaling to the number of storage cells inside the memory. PeMM does not model error occurring inside sequential logic, which can be alternatively addressed using PTM [20].

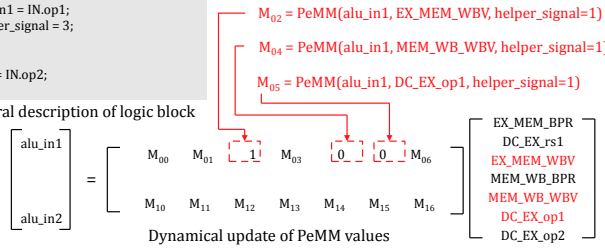
4) Inputs with Multiple Faults: Multiple faults on the inputs of the circuit also affects PeMM characteristics. Matrix multiplication sums up the contribution of each input error after individual masking effects, which achieves good masking accuracy for non-algorithmic operations. For algorithmic operations multiple input faults can strongly vary the values of $M_{in_j}^{out_i}$ compared with fault on single input, especially in the case of correlated faults

```
BEHAVIOR
{
  if (BYPASS_ACTIVE(MEM.IN.BPR, IN. rs1))
  {
    alu_in1 = MEM.IN.WBV;
    helper_signal = 1;
  }
  else
  {
    if (BYPASS_ACTIVE(WB.IN.BPR. IN.rs1))
    {
      alu_in1 = WB.IN.WBV;
      helper_signal = 2;
    }
    else
    {
      alu_in1 = IN.op1;
      helper_signal = 3;
    }
  }
  alu_in2 = IN.op2;
}
```

Behavioral description of logic block

helper_signal	MEM.IN.WBV	WB.IN.WBV	IN.op1
1	1	0	0
2	0	1	0
3	0	0	1

PeMM LUT w.r.t. value of helper signal



Dynamical update of PeMM values

Fig. 4. Control Flow Handling for PeMM

which are completely or partially originated from the same fault. Correlated faults are possible to inversely affect each other and even cancel the resultant errors depending on the algorithmic operation, such as in the case of XOR operator with bit-flip faults on both inputs.

C. PeMM Characterization

We characterize PeMM elements through high-level behavioral simulation where primary inputs of logic blocks are subject to fault injection. For a specific circuit with random data inputs, the masking probability $M_{in_j}^{out_i}$ can be acquired by averaging the error probability on out_i among multiple experiments, where single bit-flip fault is injected onto input in_j in each experiment.

1) Accuracy of PeMM Characterization: The characterization testbench compares the fault-free (golden) simulation results with fault injection on the specific inputs. To characterize each element in PeMM with a required confidence level and confidence interval, the number of experiments is determined by all the possibilities of bit flips in given input space (population) according to [22]. For a circuit with n inputs of m bits each, the size of input space is $2^{m\times n}$, the overall size of possible experiments equals $2^{m\times n}\times m$ with the possibilities of random bit-flip in a given input space. For instance, a circuit with 2 inputs, each 32 bits wide, requires 9604 experiments to achieve 95% confidence level with confidence interval of 1.

2) Fine-grained PeMM: As stated previously, granularity of PeMM varies depending on the level of error existence. When the input vectors represent signal-level error probabilities, non-zero values in output vectors indicate the existence of errors in particular signals. An instinctive extension to the approach would be the focus of error existence in smaller granularity such as byte or nibble level, where PeMM predicts not only signal-level error existence but also in which byte/nibble the error exists. This could be of particular importance in the field of approximate application, where the achieved QoS could be traded off with other design metrics such as power consumption and area overhead.

Fine-grained PeMM can be created using additional look-up-table for values of $M_{in_j}^{out_i}$ as in Table I. The first column represents the targeted operations while the second column forms a *Key* variable showing in which bytes the faults locate for both inputs of logic primitives. For instance, key 13 shows

faults in 1^{st} byte of first input and 3^{rd} byte of second input while key 10 shows no fault in second input but only 1^{st} byte of the first input. The byte-wise $M_{in_j}^{out_i}$ shows the probabilities of error existence in particular output bytes. Depending on targeted field of application, granularity can be further fine-grained, which requires additional efforts for characterization.

Operation	*Key*	*Byte-wise* $M_{in_j}^{out_i}$			
SUB	10	1.000000	0.126830	0.000520	0.000000
OR	22	0.000000	0.721690	0.000000	0.000000
AND	10	0.499030	0.000000	0.000000	0.000000
AND	13	0.500400	0.000000	0.499900	0.000000
XOR	33	0.000000	0.000000	0.873990	0.000000

TABLE I
EXAMPLES OF FAULT APPROXIMATION WITH BYTE-LEVEL GRANULARITY

III. APPROXIMATE ERROR PREDICTION FRAMEWORK

The approximate error prediction framework is proposed using PeMMs for individual logic blocks. In this work it is integrated with LISA-based processor design flow [23] while the approach is generic for any architecture simulator such as Verilog and SystemC simulators. Figure 5 shows an overview of the framework.

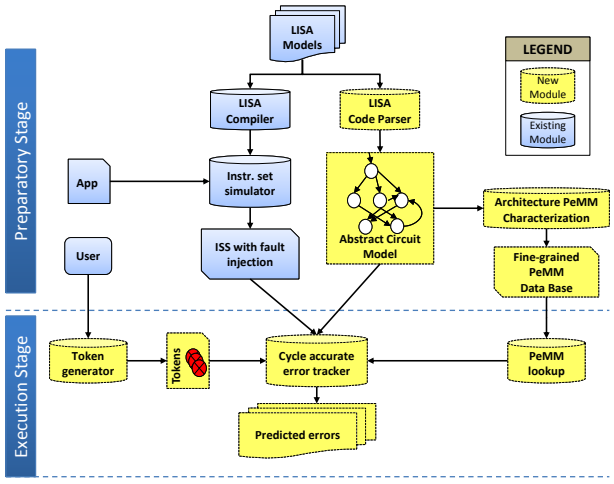


Fig. 5. Error Tracking and Prediction Framework

The flow consists of the preparatory and execution stage. In preparatory stage, cycle accurate instruction-set simulator (ISS) running specific applications is first generated from processor description using ADL LISA [24]. With the simulator extension, ISS is prone to fault injection where user can configure fault information using graphical interfaces. An extra LISA code parser is used to translate LISA description into abstract circuit models, which contain the information on the directed acyclic graph (DAG) of LISA operations and the inputs and outputs resources for individual architecture units. The PeMM characterization module translates the behavior of processor architecture units into C-based functions with signal inputs and output as function arguments. Testbenches are generated to inject random faults to function inputs, where PeMMs are fast characterized. Special language directives are checked by the LISA parser for finer PeMM characterization according to techniques in section , II-B2 and II-C2.

The execution stage starts with the creation of token by user using a graphical interface. The token represents the fault in terms of error probability, along with other fields representing the micro-architectural and timing information required to track the token as it propagates. The cycle accurate error tracker works for generic processor models, which tracks the token propagation with possible masking effects applying PeMMs for architecture units. The reports on predicted errors are generated by the error tracker, which documents detailed paths of token propagation and various masking effects through architecture units.

A. Error Representation

In contrast to fault injection method where the value of resources is changed dynamically, token is created as an abstract data structures which does not change the resource values. Instead, error probability is updated during token propagation. Error probability is initially set to 1 during token creation while fine-grained probability is set based on the approximated error granularity in section II-C2. When error probability is masked to 0 the token is removed. Hardware resource ID and memory array index are associated with each token so that the PeMM addresses the corresponding token correctly. Specific resources are possible to contain multiple *sub-tokens*, for instance the *instruction* register, which consists of multiple decoding fields such as *opcode*, *source* and *destination* operands. Due to various coding functionality so as the masking probabilities, such fields are treated as separated elements in the input vector of PeMM although located in the same resource.

Algorithm 1 Token Tracking Routine

```
1: function TRACKTOKEN(*op, *token, *PeMM)
2:     for all op_id do                          ▷ Create tokens by activation analysis
3:         if op[op_id] is active then
4:             if ∃token[tk_id] in op[op_id].inputs then
5:                 Update with PeMM[op_id] for op[op_id];
6:                 Schedule to create tokens in op[op_id].outputs;
7:                 New tokens labeled as high priority;
8:             end if
9:         end if
10:    end for
11:    for all tk_id do                          ▷ Create tokens by pipeline behaviors
12:        if token[tk_id] is in pipeline registers then
13:            Schedule to remove token[tk_id];
14:            if token[tk_id] is not in last pipeline stage then
15:                Schedule forwarding token in next stage as low priority;
16:            end if
17:        end if
18:    end for
19:    Remove_tokens();
20:    Create_tokens();      ▷ Create/remove tokens at end of the control step
21: end function

22: function CREATE_TOKENS
23:    for all tokens in schedule creating list do
24:        if ∃old token in new location then
25:            Overwrite old token;
26:        end if                                ▷ Overwrite existing tokens
27:        if multiple tokens are scheduled in the same location then
28:            Create token with high priority;
29:        else                                  ▷ Forwarded tokens have less priority
30:            Create token;
31:        end if
32:    end for
33: end function
```

B. Token Tracking

Since no actual faults are injected but only abstract tokens, the simulator maintains correct execution flow while indicates potential errors. Algorithm 1 shows the token tracker routine called between consecutive processor control steps. The routine starts with the activation analysis of LISA operations. If any operations whose inputs contain tokens are activated, the tokens are updated by PeMMs and propagate to the outputs of the

operation by the end of the cycle. Due to synchronized register behaviors, the tokens cannot be immediately created or removed before the completion of analysis for the current cycle, but are scheduled for creation and removal. After activation analysis for operations, the tokens in pipeline registers are forwarded to the next pipeline stage. However, forwarded tokens have less priority compared with the ones created from the active operations. For memory and register files, old tokens are overwritten by new arriving ones.

IV. Results in Reliability Estimation

To demonstrate the usability of the approximate error prediction flow, we present several case studies using an embedded RISC processor modeled using LISA language from IPs of Synopsys Processor Designer [23]. The processor has five pipeline stages with fully bypassing and forwarding functionality. Verilog models are generated automatically for fault injection experiments.

A. Error Prediction Analysis

We compare the predicted error probability with Verilog-based fault injection experiments [4], where the faults can be injected into physical resources such as pipeline registers, RTL signals, register file and memory arrays of the processor in Verilog representation. A testbench containing all types of instructions for the RISC processor is designed, which processes data in a loop using general purpose register arrays and stores the final data into memory. Error prediction results using different modes of PeMM construction are compared with RTL fault injection. In each fault injection experiment single bit-flip fault is randomly injected among 32 bits of register containing input data of the testbench. *1,000* such fault injection experiments with random input values are performed to calculate the average error probabilities on selected hardware resources. On the contrary, proposed error prediction analysis is performed only once to show the predicted error probabilities with the same fault configuration. Table II shows the time gained in evaluating error prediction using proposed approach against Verilog-based fault injection for the same testbench.

	Proposed error prediction	Verilog fault injection [4]
Time (sec)	0.015	187

TABLE II
Run-time Benefit against Verilog-based Fault Injection

1) Run-time of Preparatory Stage: The preparatory stage consists of two phases, the parsing stage analyzes processor description in LISA language and converted into C-based characterization testbenches, while the characterization phase generates PeMM elements based on the pre-defined PeMM modes. The parsing phase splits larger logic blocks and inserts helper signals based on user defined language constructs to decomposed larger logic blocks.

The preparatory stage analyzes 42 operations for the RISC processor. Table III shows the timing overhead for the preparatory stage on the host machine with Intel Core i7 CPU at 2.8 GHz. For each PeMM element, 100,000 characterization experiments are performed. It is observed that analysis of enhanced PeMM modes consumes more time in both phases.

	Initial PeMM (sec)	Split only PeMM (sec)	Split+full ass-signals PeMM (sec)
Parsing	0.14	0.17	0.26
Characterization	2.80	2.85	4.75

TABLE III
Processing Time for Automated PeMM Preparation

2) Error Prediction Accuracy: Figure 6 shows the prediction results. It is indicated that the initial PeMM without matrix decomposition achieves least similarity compared with fault injection, while proper PeMM decomposition and appliance of assistant signals for control flow prediction shorten the gap significantly. When assistant signals are inserted for control paths in all logic blocks, the predicted error probabilities and locations perfectly match the results from fault injection.

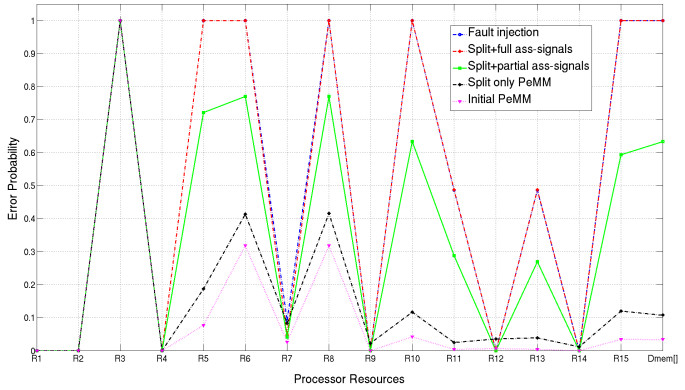


Fig. 6. Error Prediction Comparison for Different Modes of PeMM against Fault Injection [4]

B. Run-time Timing Overhead for Token Tracking

1) Timing Analysis against Token Counts: The token tracking framework is an extension of the cycle accurate instruction-set simulator from Synopsys Processor Designer [23]. Table IV shows the timing overhead caused by the token tracking against original simulation using several embedded benchmarks. Analysis without any token adds in average 28.4% simulation overhead due to the additional analysis for operation activation in each clock cycle. When there is one token created, the tracking routine consumes 6.7% overhead. When 20 tokens are created, in average 79.3% overhead is added. The exact overheads differs among experiments due to the host machine usage and randomness in both injected location and time instance, where in most of the cases tokens are masked during execution. Tracking of multiple tokens scales the simulation time in a linear fashion due to the fact that tokens are managed in the unordered hash map with algorithmic complexity of *O(1)* for element searching [25].

Apps	Synopsys ISS simulation [23]	Error tracker					
		0 token		1 token		20 tokens	
	No fault (sec)	(sec)	+%	(sec)	+%	(sec)	+%
Cordic	1.7	2.3	35	2.5	9	3.3	32
CRC	2.1	3.2	52	3.4	6	6.2	82
IDCT	2.5	2.8	12	2.9	4	7.0	141
Rijndael	2.3	3.5	52	3.9	11	5.8	49
Sobel	1.8	2.1	17	2.4	14	6.4	167
Viterbi	3.3	4.3	30	4.4	2	8.1	84
Average	-	-	28.4	-	6.7	-	79.3

TABLE IV
Timing Overhead Analysis against Architecture Simulator

2) Timing Analysis against Modes of PeMM: Figure 7 shows time consumed by several embedded benchmarks using different modes of PeMM. It is observed that run-time efforts of error prediction using split PeMM with helper signals increase, which shows a trade-off between prediction accuracy and analysis time.

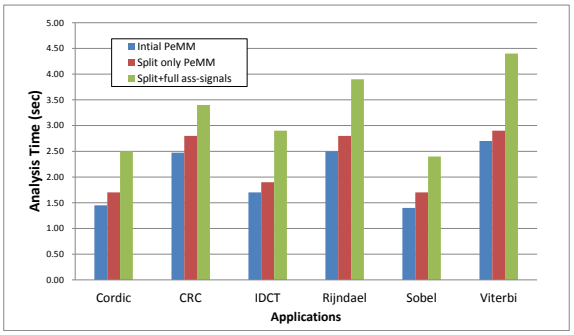


Fig. 7. Run-time among Different PeMM Modes

C. Prediction of Error Locations

The advantage of error prediction goes beyond its speed and accuracy. Fault injection is impossible to track where exactly the errors result in the huge memory address space. The ability to predict error locations especially in memory array assists the designer to find the potential error effects in particular hardware resources. We demonstrate the usage of data prediction using the median filter [26] where both input and output images are shown in figure 8. When two tokens are injected in the memory location which contains the input image, their affecting regions are predicted in the output image accordingly. Such prediction matches the algorithm specification in [26], where the value of each pixel in the output image is related to the median value of the pixel at the same position and its surrounding 8 values in the input image.

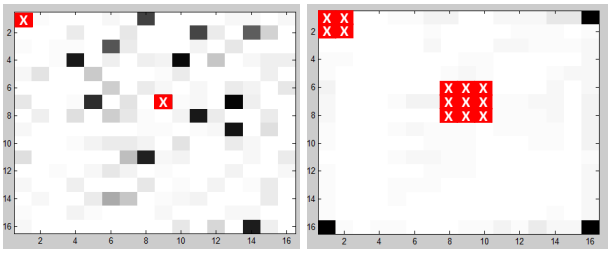


Fig. 8. Error Prediction for Median Filter Application

V. Conclusion

In this work, probabilistic error masking matrix is introduced to investigate the error masking effects of logic circuits. Taking advantage of PeMM, a fast and approximate error prediction framework integrated with commercial processor design flow is introduced which tracks the paths of error propagation and estimates fine-grained error probabilities. Vulnerability of hardware resources can be easily estimated, while location and significance of errors are predicted. Benchmark results with state-of-the-art RTL fault injection indicates that the proposed framework achieves high accuracy of error prediction and significant speed-up.

Future work includes detailed QoS-reliability exploration for applications in approximate computing using the framework. Besides, the setup of proposed framework for generic processors will be automated so that error propagation on application specific architectures can be fast and accurately analyzed.

References

[1] C. Constantinescu. Trends and challenges in vlsi circuit reliability. *IEEE Micro*, 23(4):14–19, 2003.

[2] E. Normand. Single event upset at ground level. *IEEE Transactions on Nuclear Science*, 43(6):2742–2750, 1996.

[3] P. Bose J. Rivers J. Srinivasan, S. V. Adve and C. K. Hu. RAMP: A model for reliability aware microprocessor design. *IBM Research Report*, 2003.

[4] 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 *Proc. 3rd IEEE International Conference on Secure Software Integration and Reliability Improvement (SSIRI)*, 2009.

[5] M. Sugihara, T. Ishihara, M. Muroyama and K. Hashimoto. A Simulation-Based Soft Error Estimation Methodology for Computer Systems. In *ISQED*, pages 196–203, 2006.

[6] H. Cho, S. Mirkhani, C. Cher, J. A. Abraham, and S. Mitra. Quantitative evaluation of soft error injection techniques for robust system design. In *DAC*, page 101, 2013.

[7] S. S. Mukherjee, C. T. Weaver, J. S. Emer, S. K. Reinhardt, and T. M. Austin. A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor. In *MICRO*, pages 29–42, 2003.

[8] A. Biswas, P. Racunas, J. S. Emer, and S. S. Mukherjee. Computing Accurate AVFs using ACE Analysis on Performance Models: A Rebuttal. *Computer Architecture Letters*, 7(1):21–24, 2007.

[9] S. Rehman, M. Shafique, F. Kriebel, and J. Henkel. Reliable software for unreliable hardware: embedded code generation aiming at reliability. In *CODES+ISSS*, pages 237–246, 2011.

[10] S. Rehman, M. Shafique, F. Kriebel and J. Henkel. RAISE: Reliability-Aware Instruction SchEduling for unreliable hardware. In *ASP-DAC '12*, pages 671 –676, 30 2012-feb. 2 2012.

[11] D. Ernst, N. S. Kim, S. Das, S. Pant, R. R. Rao, T. Pham, C. H. Ziesler, D. Blaauw, T. M. Austin, K. Flautner, and Trevor N. Mudge. Razor: A low-power pipeline based on circuit-level timing speculation. In *MICRO*, 2003.

[12] J. Srinivasan, S. V. Adve, P. Bose and J. A. Rivers. The Case for Lifetime Reliability-Aware Microprocessors. *SIGARCH Comput. Archit. News*, 32(2):276, 2004.

[13] A. Kahng, S. Kang, R. Kumar, and J. Sartori. Designing Processors from the Ground up to Allow Voltage/Reliability Tradeoffs. In *HPCA*, 2010.

[14] S. K. Reinhardt and S. S Mukherjee. Transient Fault Detection via Simultaneous Multithreading. ISCA '00, pages 25–36, 2000.

[15] A. DeHon, H. M. Quinn, and N. P. Carter. Vision for cross-layer optimization to address the dual challenges of energy and reliability. In *DATE*, pages 1017–1022. IEEE, 2010.

[16] R. Hegde and N.R. Shanbhag. Energy-efficient Signal Processing via Algorithmic Noise-tolerance. ISLPED '99.

[17] H. Cho, L. Leem and S. Mitra. ERSA: Error Resilient System Architecture for Probabilistic Applications. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 31(4):546–558, 2012.

[18] K. Palem and A. Lingamneni. Ten years of building broken chips: The physics and engineering of inexact computing. *ACM Trans. Embed. Comput. Syst.*, 12(2s):87:1–87:23, May 2013.

[19] M. Willems H. Keding, F. Hrtgen and M. Coors. Transformation of floating-point into fixed-point algorithms by interpolation applying a statistical approach. In *Proc. Int. Conf. on Signal Processing Application and Technology (ICSPAT)*, Toronto, sep 1998.

[20] S. Krishnaswamy, G. F. Viamontes, I. L. Markov and J. P. Hayes. Probabilistic transfer matrices in symbolic reliability analysis of logic circuits. *ACM Trans. Design Autom. Electr. Syst.*, 13(1), 2008.

[21] Norbert Seifert, Balkaran Gill, Shah Jahinuzzaman, Joseph Basile, Vinod Ambrose, Quan Shi, Randy Allmon, and Arkady Bramnik. Soft error susceptibilities of 22 nm tri-gate devices. *Nuclear Science, IEEE Transactions on*, 59(6):2666–2673, 2012.

[22] David Roxbee Cox and David Victor Hinkley. *Theoretical statistics*. CRC Press, 1979.

[23] Synopsys. *Processor Designer http://www.synopsys.com/Systems/BlockDesign/processorDev*.

[24] A. Chattopadhyay, H. Meyr and R. Leupers. *LISA: A Uniform ADL for Embedded Processor Modelling, Implementation and Software Toolsuite Generation*, chapter 5, pages 95–130. Morgan Kaufmann, jun 2008.

[25] Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, and Charles E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2nd edition, 2001.

[26] T. Huang, G. Yang, and G. Tang. A fast two-dimensional median filtering algorithm. *IEEE Trans. Acoust., Speech, Signal Processing*, 27(1):13–18, 1979.