Quantitative Analysis of Soft Error Propagation at RTL

Liang Chen · Mojtaba Ebrahimi · Mehdi B. Tahoori

Received: 29 March, 2013 / Accepted: 25 April, 2013

Abstract Radiation-induced soft error is one of the main issues of system reliability with the continuous technology scaling. Soft error analysis at early design phase is essential for applying appropriate mitigation techniques to meet the reliability requirements. This paper proposes a novel approach to investigate the soft error propagation properties at behavioral register transfer level (RTL), especially for the control paths of the design. Modeling the control path as state transition system, the formal method - probabilistic model checking - is adopted to quantify the error sensitivities of all signals in the RTL design over multiple time steps. The experimental results reveal that error sensitivities across different variables show significant variance, which provides great potential for cost-efficient selective protection against soft error at higher abstraction levels.

Keywords Soft Error · RTL · Model Checking

1 Introduction

With continuous technology scaling, radiation-induced soft error becomes one of the major reliability concerns in nano era. To obtain cost-effective soft error mitigation, it is essential to estimate its impact in early design phases.

Previous work on soft error analysis can be generally categorized into two groups. Logic-level soft error estimation are based on synthesized netlist to obtain an accurate estimation of the soft error rate [1,3], which are not applicable in early design stages. Architecture-level technique [10] uses Architecturally Correct Execution (ACE) analysis to quantify the vulnerability of regular structures such as memory, register files. However, the ACE analysis is most limited to regular structures, and over-estimate the control flow error propagation [6]. Furthermore, it is mainly adopted for high-level reliability evaluation, but not proper for the usual hardware design at register-transfer level.

RTL digital system can be generally divided into data paths and control paths [5]. Compared with data paths with well-defined operators, the control paths are much more sophisticated to analyze with the introduction of branches and loops. Previous work [9,4] already investigated the error propagation probabilities in data paths, while accurate and efficient analysis of soft error in control path is still missing.

The method presented in [11] performs a qualitative analysis on the soft error vulnerabilities of flip-flops using formal verification techniques at RTL. It provides a binary classification whether an SEU in one specific flip-flop will cause system failure or not. In this paper we propose to quantitatively evaluate the vulnerabilities of RTL variables from a probabilistic point of view. We investigate the soft error propagation properties at behavioral RTL, focusing on the control paths. The formal method - probabilistic model checking [2] - is adopted to quantify the error sensitivities of all variables in the design. The experimental results verify the insights of non-uniform vulnerability distributions, which provide valuable information for efficient soft error mitigation.

The organization of the rest of this paper is as follows. Section 2 discusses how to model error propagation at RTL. Section 3 introduces proposed technique to estimate error probabilities and how to improve its scalability. Section 4 describes the experimental results and finally Section 5 concludes this paper.

2 Error Propagation Modeling at RTL

At RTL the low-level circuit details are not available yet, therefore the soft error model used here is single bit-flip, i.e. Single Event Upset (SEU). Our main focus is on transient behavior of the system impaired by soft error, i.e. the error probabilities at primary output (PO) in the successive cycles after SEU occurs. Quantitative information of this behavior is useful to identify...
not only the sensitive flip-flops, but also the sensitive time intervals in which error correction are essential.

Hardware circuits can be generally modeled as state transition systems and the clock signal triggers each state transition. One state is represented by the current values of the flip-flops and primary inputs. As hardware circuit exhibits finite number of states due to the finite number of flip-flops and primary inputs, it is usually referred as Finite State Machine (FSM).

In our work the multi-bit signal in RTL behavioral description is considered as a whole, therefore all the symbolic signals will be hereafter referred as variables instead of flip-flops.

When an SEU occurs in the control path, during its propagation to primary output the generated errors can be logically masked in several ways. To investigate the error sensitivities of different variables, we need to perform the comparison between error-free FSM and erroneous FSM, as the example shown in Figure 1. After SEU occurs in a system state, we compare the error-free values of POs with possible erroneous values in the successive states. The number of state transitions following SEU is a user-specified parameter. Note that due to the random property of soft error, the SEU can occur at any possible system state.

3 Error Sensitivities Estimation using Probabilistic Model Checking

After the RTL circuit is modeled as FSM, we turn to Probabilistic Model Checking (PMC) [2] to analyze its erroneous behaviors. PMC is a quantitative model checking technique, which takes the probabilistic aspects of the system into consideration.

The PMC process can be generally divided into three phases: i) constructing probabilistic model, ii) specifying concerned properties, and iii) model checking properties against the constructed model to generates final results.

3.1 Probability Model Checking of RTL Circuit

The probabilistic behavior of RTL circuit in our work is formally modeled as finite Discrete Time Markov Chains (DTMCs) [2], in which each state transition is associated with a probability. One DTMC state represents the unique value assignments to a set of state variables (primary inputs and flip-flops).

To obtain quantitative measurements of the modeled system, we need to specify the desired properties expressed with probabilistic extensions of temporal logic [2]. Then we use PRISM [8], a symbolic model checker to analyze the RTL DTMC model. It adopts the compact binary decision diagrams to efficiently represent and manipulate DTMCs.

3.1.1 Probabilistic Modeling of Soft Error

As mentioned in Section 2, the random property of soft error means that the SEU can occur at any possible DTMC state. The SEU lasts for only one timing step (i.e. clock cycle for hardware circuit) and in next state, only error propagation exists. In addition, the probability of error occurring at each state is not uniform, and is actually the probability that circuit stays at this state in the long run. Generally, the error probability of primary output PO_i after k transitions following a SEU can be expressed as following:

$$P_{e^k}(PO_i) = \sum_{n=1}^{N} P_{s_n} \cdot P_{e^k_n}(PO_i)$$

(1)

where N is the total number of DTMC states, P_{s_n} is the long-run probability that DTMC stays at state s_n and P_{e^k_n}(PO_i) is the error probability of PO_i after k transitions following an SEU occurring at state s_n.

To model these effects, we make use of the parallel composition mechanism of DTMC, which means for one DTMC composed with several parallel submodules, the global states of the whole system are the interleaved states of each submodule. In this way, the random occurrence of SEU is interleaved with each possible state in the system functionality. As shown in Figure 2, we model the erroneous DTMC as the composition of one system function module and one soft error occurring module. In the soft error occurring module, at state e_0 no SEU has occurred yet, and e_i, i = 1, 2,..k correspond to the successive states after SEU occurs. In the scope of hardware circuits, k corresponds to the number of clock cycles to investigate the transient behavior of the circuits impaired by soft error. The system function module is generally the same as error-free DTMC except that the variables here have the possible erroneous values caused either by SEU or error propagated from SEU.
3.1.2 Property Specification for Model Checking

To obtain the error probabilities of primary outputs in DTMC, we need to specify the correct properties which will be sent to the model checker. From Equation 1, traversal of all paths reaching the state $e_k$ in Figure 2 is necessary. Actually this can be done automatically after we specify the following linear temporal logic property:

$$P = ? [F (s = e_k \land PO_{i} = PO_{l})]$$

(2)

where $P = ?$ indicates that we want to obtain the quantitative probability, $F$ means starting from the initial state, eventually (i.e. in the Future) the event ($s = e_k \land PO_{i} = PO_{l}$), that at state $e_k$ the $i$th primary outputs from error-free and erroneous DTMC are different, will occur.

3.2 Scalability Improvement

The formal model checking techniques exhaustively explore the whole state space to obtain the desired results with high confidence. However, the well-known state explosion problem [2] limits its applicability and hence scalability improvement techniques are essential.

3.2.1 Dependency Graph Extraction

To reduce the number of concerned variables in Figure 2, the variable dependency graph is extracted in the successive state transitions after SEU. Figure 3 shows that SEU occurs at cycle $T_k$, then until the errors propagate and finally arrive at primary outputs, the influenced variables are only $v_1, v_4, PO_1, PO_2$, while all the other variables can be shared between error-free and erroneous DTMC to reduce the unnecessary exploration of state space.

3.2.2 Data Type Reduction

Another technique for scalability improvement is data type reduction, which means that error propagation properties of some RTL codes can be much simpler than the correct functionality. Take the following two code segments as example:

```
"saved_addr_r <= start_addr + 1;" (1)
"case coda0 is
  when U1 => grant:="1000";
  when U2 => grant:="0100";
  when U3 => grant:="0010";
  when U4 => grant:="0001";
  when others => grant:="0000";
end case;"
```

In Code(1) the start_addr and saved_addr_r are always wide words (32 or 64 bits), therefore directly exploring all the possible values are infeasible. However, the error propagation within this assignment is as simple as a transparent channel. Code(2) is from one benchmark we use later, and coda0 is actually an enumerated type with values $U0, U1, U2, U3, U4$. If coda0 is erroneous due to errors propagated from other variables, for the whole case statement grant will be erroneous at the same time as coda0. In other words, in the scope of error propagation the original data type can be reduced from wide word to narrower word or even boolean type.

3.2.3 Statistical Model Checking

Even with several scalability improvement techniques, it is still possible that the number of concerned states exceeds memory limit or significantly slow down the model checking process. In this scenario, statistical model checking can be used to obtain approximate results with guaranteed accuracy [7]. This is essentially achieved by probabilistically sampling large number of paths in the
model and obtaining an approximately correct result. This approach is particularly useful on very large state space without explicitly constructing the system model, and currently supported by the PRISM model checker.

4 Experimental Results

The proposed approach was implemented using PRISM tool and experiments were performed for several behavioral ITC’99 benchmarks on a workstation with Intel Xeon E5540 2.53GHz and 16GB RAM. We carried out the manual conversion from behavioral VHDL to the PRISM input language, therefore 5 relatively small benchmarks without memory access operations are selected to demonstrate the applicability of our proposed method. To validate the correctness of our DTMC modeling, RTL Fault Injection (FI) with random inputs are also implemented. For benchmark b01, b02 and b06, 1000 faults are injected into each variable, while for benchmark b03 and b09, 10000 faults are injected because of the larger state space.

Table 1 shows the average runtime for checking 10 properties corresponding to 10 cycles after SEU occurrence using PMC and FI. Note that benchmark b01, b02, b03 and b06 are checked using numerical PMC with scalability improvement techniques introduced in Section 3.2.1 and 3.2.2, while b09 is checked using statistical PMC with 99% confidence level. The average error probability differences between PMC and FI are very small - below 0.007. For runtime, on average our approach is 77x faster than the fault injected simulation. Figure 4 shows clearly that different variables in the RTL design have different transient error propagation behaviors.

5 Conclusion

Performing soft error analysis at early design phase is essential for cost-efficient error mitigation. This paper proposed a novel approach to quantitatively investigate the soft error propagation properties at behavioral register-transfer level using probabilistic model checking. Our future work will focus on using state minimization techniques to further improve the scalability of our method.

<table>
<thead>
<tr>
<th>Bench</th>
<th>PI+FF</th>
<th>FFI</th>
<th>PMC</th>
<th>Speedup</th>
<th>Average Diff.</th>
</tr>
</thead>
<tbody>
<tr>
<td>b01</td>
<td>7</td>
<td>21.53</td>
<td>0.43</td>
<td>50.58</td>
<td>0.0071</td>
</tr>
<tr>
<td>b02</td>
<td>5</td>
<td>24.55</td>
<td>0.09</td>
<td>288.56</td>
<td>0.0041</td>
</tr>
<tr>
<td>b03</td>
<td>34</td>
<td>207.09</td>
<td>22.24</td>
<td>9.31</td>
<td>0.0083</td>
</tr>
<tr>
<td>b06</td>
<td>11</td>
<td>25.96</td>
<td>0.74</td>
<td>35.03</td>
<td>0.0074</td>
</tr>
<tr>
<td>b09</td>
<td>30</td>
<td>207.29</td>
<td>41.31</td>
<td>5.02</td>
<td>0.0061</td>
</tr>
<tr>
<td>average</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>77  0.0068</td>
</tr>
</tbody>
</table>

### Acknowledgements

This work was partly supported by the German Research Foundation (DFG) as part of the national focal program “Dependable Embedded Systems” (SPP-1500, http://spp1500.ira.uka.de/).

### References