Microprocessors and Microsystems 38 (2014) 598–604

Contents lists available at ScienceDirect

Microprocessors and Microsystems

journal homepage: www.elsevier.com/locate/micpro

Bit Impact Factor: Towards making fair vulnerability comparison

Serdar Zafer Can [a], Gulay Yalcin [b,*], Oguz Ergin [a], Osman Sabri Unsal [b], Adrian Cristal [b,c]

[a] TOBB University of Economics and Technology, Söğütözü Avenue 43, Söğütözü, Ankara 06560, Turkey
[b] Barcelona Supercomputing Center, c/ Jordi Girona, 29 08034 Barcelona, Spain
[c] IIIA – CSIC – Spain National Research Council, Campus de la UAB, E-08193 Bellaterra, Catalonia, Spain

ARTICLE INFO

Article history:
Received 30 October 2013
Revised 26 April 2014
Accepted 27 April 2014
Available online 10 May 2014

Keywords:
Architectural vulnerability factor
Soft errors
Vulnerability
Fault injection

ABSTRACT

Reliability is becoming a major design concern in contemporary microprocessors since soft error rate is increasing due to technology scaling. Therefore, design time system vulnerability estimation is of paramount importance. Architectural Vulnerability Factor (AVF) is an early vulnerability estimation methodology. However, AVF considers that the value of a bit in a clock cycle is either required for Architecturally Correct Execution (i.e. ACE-bit) or not (i.e. unACE-bit); therefore, AVF cannot distinguish the vulnerability impact level of an ACE-bit. In this study, we present a new dimension which takes into account the vulnerability impact level of a bit. We introduce Bit Impact Factor metric which, we believe, will be helpful for extending AVF evaluation to provide a more accurate vulnerability analysis.

1. Introduction

Transient faults such as bit flips mainly caused by particle strikes, are important problems in a digital system design [1,2]. These particle strikes do not result in permanent faults but may lead to system crashes, and hence, are termed as "*soft errors*". It is predicted that the soft error problem will increase in the future systems since, in every new generation of manufacturing technology, feature sizes decrease; consequently error susceptibility of digital circuits increases [3]. This increasing soft error rate makes reliability a major design concern in contemporary microprocessors.

The vulnerability of the system to soft errors should be quantified as early as possible at design time, so that required precautions can be taken. Also, it is important not to overestimate/underestimate the vulnerability of the system due to the associated performance/power overheads.

Mukherjee et al. define the Architectural Vulnerability Factor (AVF) of processor components to provide early reliability estimation [4]. AVF analysis is implemented based on the fact that systems are known to mask some of the faults either at the circuit level or architectural level and these faults do not propagate to the final outcome of a program. Quantifying this masking effect allows adjusting the level of error protection in the design of a digital system.

AVF is defined as the average ratio of the bits in the system that are required for Architecturally Correct Execution (i.e. ACE-bits) at a given clock cycle. AVF analysis does not care about the process of a bit flip creating the error, but it rather qualifies the outcome: if the flip on a stored bit results in a system level visible error, the flipped bit is defined as ACE-bit. In reality, a bit flip may lead to an unwanted outcome through different paths.

Mukherjee et al. advocate that the AVF of the whole processor can be calculated by summing the AVFs of all structures multiplied by their area normalized with respect to total chip area. However, this assumption introduces discrepancies in the vulnerability especially for systems with many components or for long-running workloads [5]. This is mainly due to two reasons: (1) AVF always assumes that all of the ACE-bits in a processor component or in an instruction have the same impact on vulnerability. (2) AVF assumes that ACE-bits in different components are equally important for the vulnerability. Due to these limitations, it is not possible to make an apple-to-apple reliability comparison between hardware components (e.g. register file, issue queue) or different parts of a hardware component (e.g. data or tag area of the cache).

In this study, our goal is investigating a new dimension that provides a comparison between different bit types and a comparison between different hardware components. To this end, we start from the fact that not all of the bits belong to the same field type, and we show that the impact of faults occurring on different fields are different.

For instance, a single bit fault occurring on the immediate value of an instruction may cause the result of the instruction to be faulty in one bit position. However, if a failure occurs in the source register

* Corresponding author.
E-mail addresses: szcan@etu.edu.tr (S.Z. Can), gyalcin@bsc.es (G. Yalcin), oergin@etu.edu.tr (O. Ergin), ounsal@bsc.es (O.S. Unsal), acristal@bsc.es (A. Cristal).

http://dx.doi.org/10.1016/j.micpro.2014.04.009
0141-9331/

identifier, it causes reading completely different source value from the wrong register which may cause multiple bit failures in the result of the instruction.

We first classify *bit types* in several hardware components (i.e. Register File, Reorder Buffer and Issue Queue) in terms of vulnerability level, and we examine impact of a single bit flip in each class. We define Bit Impact Factor (BIF), which shows the vulnerability level of a bit, and indirectly allows the quantification of the relative vulnerability across processor components and component fields.

The contributions of this study are:

- We classify bit types in several hardware components (i.e. Register File, Reorder Buffer and Issue Queue) in terms of vulnerability level.
- We define Bit Impact Factor (BIF) which shows the average number of bits affected in the next dependent component when the defined bit fails.
- We extend AVF with BIF dimension in order to provide more accurate vulnerability analysis and to allow connecting the vulnerability of different hardware components.

2. Quantifying soft error impact factor

In this section, we first explain AVF principles. Then, we classify bits within microarchitecture classified according to the information they store and on the vulnerability impact of bits.

2.1. Architectural Vulnerability Factor (AVF)

Even though a bit is flipped as a result of a particle strike, the program running on the processor may not be affected. This is because of the fact that the values of many bits inside the processor components are not required for the correct execution. The bits that are needed for the Architecturally Correct Execution (ACE) of the running program were previously termed as the "*ACE-bits*" by Mukherjee et al. [4]. Each structure inside a processor contains a variable number of ACE bits over program execution time which determines the level of its vulnerability to soft errors. If the number of ACE bits in the structure is high, it is more probable that a bit flip will result in an error. The level of a component's vulnerability to soft errors is termed as *Architectural Vulnerability Factor (AVF)*. AVF is a useful metric in the design time of a processor in order to verify that the processor meets the reliability targets and is calculated with the following equation:

$$AVF = \frac{Average\ Number\ of\ ACE\ Bits\ in\ a\ Hardware\ Structure}{Total\ Number\ of\ Bits\ in\ the\ Hardware\ Structure} \quad (1)$$

2.2. Bit classification

In all of the vulnerability factor definitions, the objective is to decide whether or not a bit flip results in a visible fault by a predefined time. This approach assumes that there is no level of vulnerability; a bit can either be vulnerable or not. In reality, the impact of a fault in different fields of a processor component differs. Some faults may result in more bit flips during the fault's propagation and lead to the ultimate system crash faster, while others may be localized into a single bit.

In Fig. 1, we present a case in an ACE instruction. In the instruction, any flips in some bits are more likely to cause an error than other bits. In the figure, the code multiplies a number by 3 and if the result is lower than 8, the number is increased by 2. At the end of the correct execution, R1 holds 4. After the comparison operation (i.e. I3), R0 becomes un-ACE since, later, it is overwritten by 0. In the figure, we present two faulty cases for the multiplication instruction (i.e. I2). In the first case, the fault affects the immediate value and the number is multiplied by 2 instead of 3. The change in the immediate value do not change the result of the branch instruction and the final result in R1 still becomes 4. In the second case, the fault affects the destination register identifier and the result is written to R1 instead of R0. This fault does not affect the result of comparison operation either and branch is still not taken. However, the value written to R1 is quite different than the correct one. Thus, in the example, a possible fault in the destination register identifier is more vital than a fault in the immediate value.

The number of faulty bits that a bit flip causes can be used as a metric to show the impact of an error as it quantifies the maskability of the fault. As in the previous example, the probability of a maskable fault in the immediate value is higher than a maskable fault in the destination register identifier. Thus, we define Bit Impact Factor (BIF) which shows the average number of bits affected in the next dependent component when the defined bit fails.

A faulty value does not affect the system unless it is read from its location and used for a calculation. All of the read data will eventually arrive at the functional unit and an operation will be performed on it. Therefore the functional unit is the actual location where the degree of masking is determined by the number of faults arriving to the inputs of the unit and by the operation performed on the inputs (showed in Fig. 2). We classify information types stored in processor components according to the effects of them on the input parameters of the functional units. The classification is as following:

1. Regular Data (registers, immediate values, etc.)
2. Opcodes (ALU op, Function bits)
3. Source Identifiers (source physical register tags)
4. Destination Identifiers (destination tags, structure entry ids such as reorder buffer id and LSQ id)
5. Control Information (These bits hold the status of the stored information like ready and valid)
6. System Level Bits

In this section, we explain these groups with a Single Event Upset (SEU) model.

2.2.1. Regular data

Each bit, other than the control bits, of the register file as well as immediate values belong to the regular data group. When a bit flip occurs on any of the bits, a new value that is hamming distance 1 apart from the original value is created. This new value may create a single bit difference at one of the source inputs of the functional unit as shown in Fig. 3.

Many operations can mask a single bit fault and the fault may not propagate to the result value depending on the operation. If any fault is observed at the output of the functional unit, it propagates to the next dependent instruction as the result value is stored inside the register file and used as a source operand.

2.2.2. Opcode

Opcode is passed to the processor by the compiler and is decoded to generate the instruction payload by the processor. Depending on the design of the instruction set, many opcodes may be similar to each other in terms of the opcode value performing similar operations. For example, in Alpha instruction set, opcodes between 14.02A and 14.7 EB perform variants of the SQRT operation. Therefore changing the opcode in a single bit location sometimes leads to no error and is totally masked.