INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH & DEVELOPMENT

ISSN: 2278 – 0211 (Online)

To Improve Register File Immunity Against Soft Error

B.Ashok Kumar
M.Tech Student, Department of ECE, Prakasam Engineering college
K.Koteswara Rao
Assistant professor,, Department of ECE, Prakasam Engineering college

Abstract:
Gradually shrinking in feature size, increasing power density etc. Increase the vulnerability Of microprocessors against soft errors even in terrestrial Applications. The register file is one of the essential Architectural components where soft errors can occur in regular manner. This errors may rapidly spread from there throughout the whole system and the output results my get damage. Thus, register files are recognized as one of the major concerns when it comes to reliability. This paper introduces Self-Immunity, a technique that improves the integrity of the register file with respect to soft errors. Based on the observation that a certain number of register bits are not always used to represent a value stored in a register. This paper deals with the difficulty to exploit this obvious observation to enhance the register file integrity against soft errors. We show that our technique can reduce the vulnerability of the register file considerably while exhibiting smaller overhead in terms of area and power consumption compared to state-of-the-art in register file protection .we developed our project by using 64-bits register **.**

Keywords:*ECC(ERROR CORRECTION CODE); FPGA(SPARTAN-3,XC3S400).*

1.Introduction

Over the last decade, and in spite of the increasing Complex architectures, and the rapid growth of new Technologies, the technology scaling has raised soft errors to become one of the major sources for processor crashing in many systems in the nanotechnology era. Soft errors caused by charged particles are dangerous primarily in high atmospheric, where heavy alpha particles are available & high power dissipation However, trends in today's nanometer technologies such as aggressive shrinking have made low-energy particles, which are more superabundant than high-energy particles, cause appropriate charge to provoke a soft error.

Furthermore, there is a prevailing prediction that soft errors will become a cause of an inadmissible error rate problem in the near future even in earthbound application Researchers have mainly and traditionally focused on mitigating soft errors in memory and cache structures due to their large sizes. On the other hand
relatively work had been Conducted for register files although they are very Susceptible against soft errors Despite the overall rather Small area footprint of the register file, it is accessed more frequently than any other architectural component.Thus, corrupted data in any register, if not taken care of,may propagate rapidly throughout the other parts of processor, leading to drastic system reliability problems In fact, soft errors in register files can be the cause of a large number of system failures.& more power consumption was conventionally a major concern in embedded system due to their conceder effect on system So in my project we present a technique for improving the immunity of register file against the software error by sorting ecc (error correction code) in the unused bits of register file , so that the data may not corrupt. By this process we can secure our data from hackers also ,instead of using additional circuit to protect the register file from soft error we introduce error correction code which we can reduce circuit size & simple processes to develop .my project can also support register file integrity i.e which reduce space in the memory cells &highly protection scheme.

2.Block Diagram

The block diagram and operation processes writing the code ecc in to .64-bits register here in this architecture consist of encoder , mux , *self-π* &register

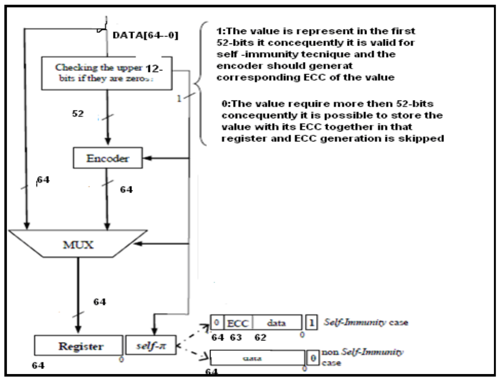
DATA[64--0]
Checking the upper 12-bits if they are zeros:
1:The value is represent in the first 52-bits it concequently it is valid for self -immunity tecnique and the encoder should generat corresponding ECC of the value
0:The value require more then 52-bits concequently it is possible to store the value with its ECC together in that register and ECC generation is skipped
52
1
Encoder
64
64
MUX
64
Register
64
0
self-π
0
ECC
data
1
Self-Immunity case
64 63 62
0
data
0
non Self-Immunity case
64
0

Figure 1:Block diagram represent writing ecc in to register in encoder

2.1.Writing In To Register

The operation how to write ecc code as given below .when ever an instruction writes a value into a register it checks the upper six bits of that value if they are '0' or not. If they are (52-bit register value case), the corresponding *self-π* bit is set to '1' indicating the existence of *Self-Immunity*. The ECC value is generated and stored in the upper unused bits of the register. Hence, the data value and its ECC are stored together in that register. In the second case (over-52-bit register value), the corresponding *self-π* bit is set to '0' and the value is written into the register without encoding shown in Fig. 1

2.2.Reading From Register

In read operations, the self-π bit is used to distinguish between a Self-Immunity case and a non self-Immunity case. In the first case, the value and the corresponding ECC are stored together in that register and consequently the read value should be decoded. In the second case, the stored value is not encoded and as resulte there is no need to decode is

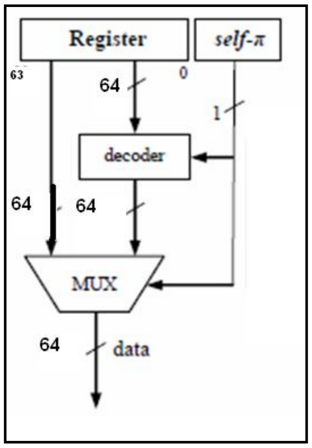

Figure 3: Micro architecture support to read a register value decoder

2.2.1.Goal

The goal of our technique is to reduce the

register file vulnerability with minimum impact on both area and power overhead. Let *N* be the total number of registers and *V* the vulnerability of a register, then the vulnerability of the register file is (Σi=1 Vi) . Since the power overhead2 mainly stems from accessing the encoder and decoder, it can approximately be modeled through the number of accesses Let *M* is the number of protected register values and *A* the number of accesses, then the total power overhead can be estimated as

2.2.2. Effectiveness Of Our Technique

in a full protection scheme, an ECC generation is performed with each *write* operation and similarly ECC checking is performed witheach *read* operation. Our technique decides to protect the value depending if it is valid for *Self-Immunity*, then it activates the ECC generator to compute the ECC bits.

Otherwise, the ECC generation is skipped. Similarly, on

every register *read* operation, instead of always checking ECC, our technique checks whether the ECC is being embedded in the register value, and only if it is, ECC checking is performed. As is demonstrated in

Fig. 1, on average 12% of the data will be stored in the register file without protection. As a result, our technique reduces *M* and it may lead to reduce the consumed power. As is shown in Fig. 3, when studying 64-bit architectures, 93% (on average) of the total *vulnerable intervals* are *vulnerable intervals* of valid register values for our technique. In other words, around 93% of *vulnerable intervals* will potentially be invulnerable. Thus, our technique promises to reduce the vulnerability of the

2.2.3.Effectiveness Of Our Technique

In a full protection scheme, an ECC generation is performed with each *write* operation and similarly ECC checking is performed with each *read* operation. Our technique decides to protect the value depending if it is valid for *Self-Immunity*, then it activates the ECC generator to compute the ECC bits.

Otherwise, the ECC generation is skipped. Similarly, on every register *read* operation, instead of always checking ECC, our technique checks whether the ECC is being embedded in the register value, and only if it is, ECC checking is performed. As is demonstrated in Fig. 2, on average 12% of the data will be stored in the register file without protection. As a result, our technique reduces *M* and it may lead to reduce the consumed power. As is shown in Fig. 3, when studying 32-bit architectures, 93% (on average) of the total *vulnerable intervals* are *vulnerable intervals* of valid register values for our technique. In other words, around 93% of *vulnerable intervals* will potentially be invulnerable. Thus, our technique promises to reduce the vulnerability of the

3.Design

The design process here we are using 64-bits register to store the data and encoder to encode the ecc code here we also using multiplexer to combine encode value and upper twelve value .

We used , self-π to check the values in the register file.here , self-π play a major role in my project .

Here self-π is one we require self –immunity code and ecc is written in register file if self-π is zero we does not require ecc code it directly given to multiplexer. here in here in this project we are using fpeg –kit to dump and check the output waveforms and we used 104 –flip flop to store the value

3.1.Top Level Design Block

the top level diagram which represent 64-bits input here we are using clock signal & reset signal to the decoder to control decoder

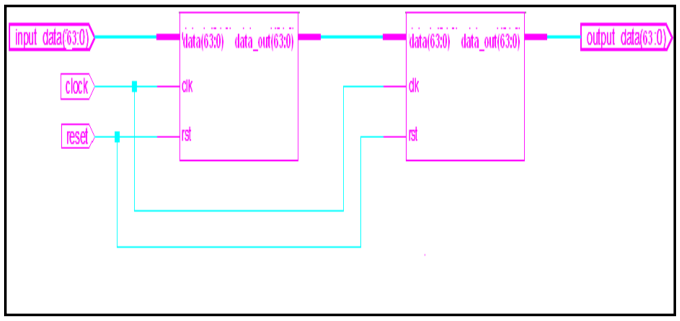

Figure :3.Block design inside the tope level

3.1.2.<u>Simulation Results</u>

the simulation result of the encoder

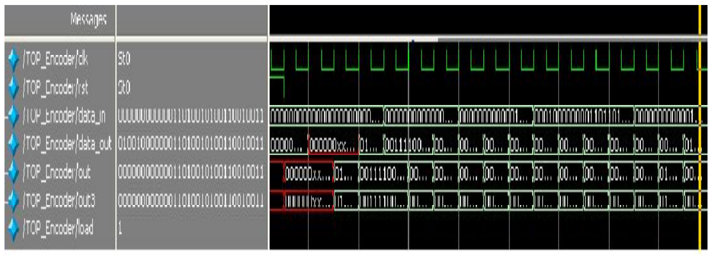

Figure:4.test wave form for encoder

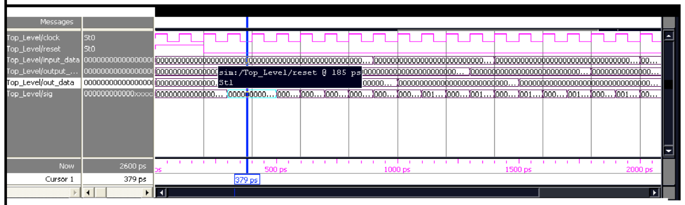

Fig 5.test wave form of the decoder

Figure 5

4.Conclusion

For embedded systems under stringenth cost constraints, where area, performance, power and reliability cannot be simply compromised, we propose a soft error mitigation technique for register files.

we developed our project by using 64-bits register file to improve register against soft error and reduce the hacking technique Our experiments on different embedded system applications demonstrate that our proposed Self-Immunity technique reduces the register file vulnerability effectively and achieves high system fault coverage.. It is very highly secure for both transmit & receive data in communication system Moreover, our technique is generic as it can be implemented into diverse architectures with minimum

5.Reference

1. The Design Warrior’s Guide to FPGAs--Clive “Max” Maxfield
2. FPGA Prototyping by VHDL Examples Xilinx SpartanTM-3V ersion----Pong P. Chu
3. *A VHDL Primer* ----Jayaram Bhasker
4. MiBench(http://www.eecs.umich.edu/miben/)
5. Greg Bronevetsky and Bronis R. de Supinski, ”Soft Error Vulnerability of Iterative Linear Algebra Methods,”
6. g.mimik ,m t kandemir & Ozturk increases register file integrity to transist error in design of automation and tested in the Europin