Robustness of Numerical Representations

Brianna Loo and Yashas Kumar; Student Advisors: Mark Gottscho, Liangzhen Lai, John Lee
Faculty Advisor: Professor Puneet Gupta

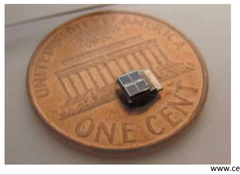
National Science Foundation

Sponsored by the Nicholas Endowment

UCLA Henry Samueli School of Engineering & Applied Sciences – Department of Electrical Engineering – NanoCAD Lab

Abstract

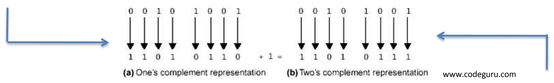
As memory chips enter the nano-scale, manufacturing and environmental variances are likely to cause inconsistencies in chip performance. These variations result in bitwise faults (flipped or stuck bits). The severity of error depends on the location of the faulty bit, the type of error, and the binary representation employed. Thus, a multi-staged approach is necessary to model and analyze the varying accuracy levels. We first use Monte Carlo simulations to model and analyze the effect that different bitwise errors induce within each numerical representation. Furthermore, we propose techniques that can boost error tolerance. With an error tolerant numerical storage system, a greater number of products meet reliability requirements and can be released into the market.



Signed Binary Systems

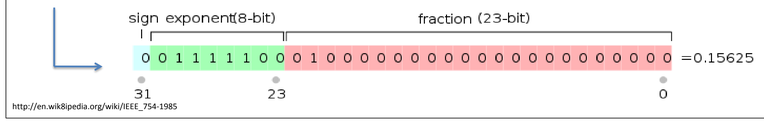
A variety of different signed number representations allow computers to portray both positive and negative numbers:

• One's Complement – The negative form of a number is obtained by inverting each bit to form the 'complement' of the positive number.



• Floating Point – contains sign, exponent, significand; converted to decimal notation using:
sign x (significand x base^{exponent})

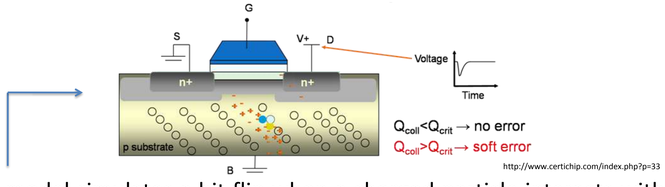
• Two's Complement – negative number is produced using $2^N - z$ (N = number of binary digits (bits), z = positive value of the number)



Error Models

To test error tolerance of the numerical representations, we operate under two different error models: stuck-at-fault and soft error.

• Stuck-at-fault model emulates the unchanging value of a specific bit when a wire is disconnected in a circuit.



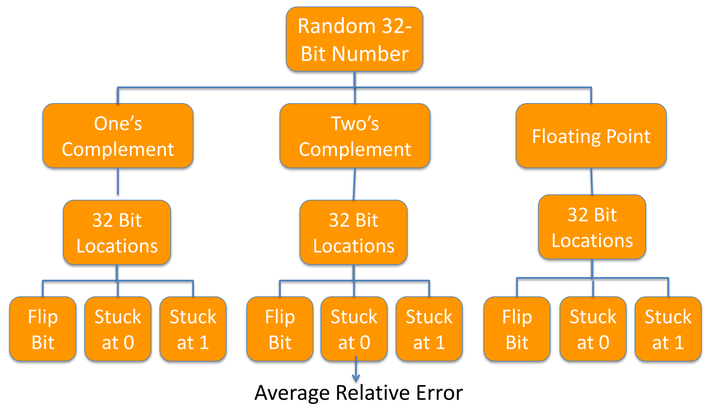
• Soft error model simulates a bit flip when a charged particle interacts with a transistor, altering the voltage, and thus the value of the bit register
  • Soft errors only occur when collected charge > critical charge parameter (which is determined by the design of the chip)

Monte Carlo Method

A total of 2^{32} numbers can be represented in 32 bits.

• Calculating the true average relative error, would require processing numbers in the hundred billions
  • Infeasible within the time constraints of the program

• To save time and preserve accuracy, we employed the Monte Carlo method implemented using Python.
  • generate random inputs over a specified domain, perform a calculation on these inputs, and combine the results.

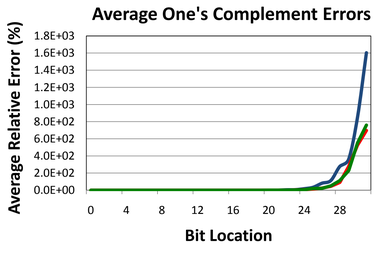


Representation Analysis

Through the Monte Carlo Method, we gathered the average relative errors of each of the error types in each numerical representation.

Blue = Flipped Bit Green = Stuck-at-Zero Red = Stuck-at-One

Figure 1



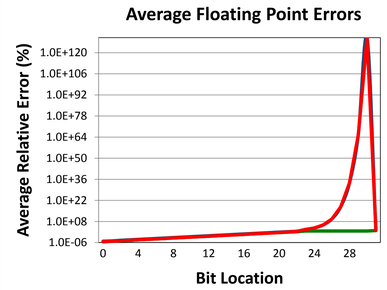
• Error when switching a 0 to a 1 is equal to inverse error (1 to 0)
• Average errors for stuck-at-zero and stuck-at-one are the same
• Average error for flipped bit is large compared to stuck-at errors

• The range of two's and one's complement are almost equal implying that error distributions would be the same as well
• Errors in two's complement and one's complement are equivalent

Figure 2

*Note: Scales are not identical

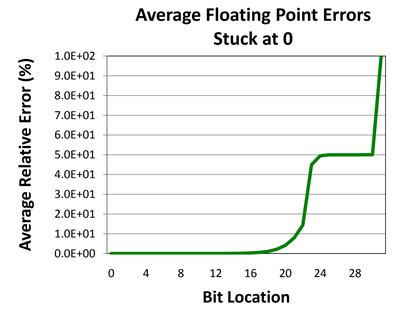
Figure 3



• Errors insignificant until faulty bits reach exponents
• When exponent bit is flipped from 1 to 0, maximum error is 100%
• When exponent bit is flipped from 0 to 1, error increases

• A flip from 1 to 0 in the more significant exponent bits results in an error of nearly 100%
• When the exponent bit where the fault occurs is equal to 0, error is 0%
• Average error becomes (0 + 100)/2, which simplifies to 50%

Figure 4



Conclusion

One's and Two's Complement have similar error distributions, so our conclusions are mainly focused on floating point.

• Floating point Stuck-at-One is much more detrimental than Stuck-at-Zero
• In the exponent bits of a 32-bit floating point representation:

11011011 = 92
01011011 = -36

Stuck-at-0 → 01011011 = -36 → 100% error

Stuck-at-1 → 11011011 = 92 → 10^{128} % error

Future Work

We propose alternate forms of floating point that could be less prone to error:
• Floating Representation – exponent bits placed in areas with a lower probability of error
• Exponent Representation – change exponent representation depending on the type of error incurred
• Parity Bits – add extra bits to ensure the accuracy of each bit

Acknowledgements

Thank you to Liangzhen Lai, Mark Gottscho, John Lee, and Professor Gupta for guiding us through this project and experience. Thanks to The Variability Expedition and the Los Angeles Computing Circle for giving us this opportunity.

Variability Expedition