NORTHWESTERN UNIVERSITY

Hardware Error Rate Characterization with Below-nominal Supply Voltages

A THESIS

SUBMITTED TO THE GRADUATE SCHOOL

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

for the degree

MASTER OF SCIENCE

Field of Computer Engineering

By

Ke Liu

EVANSTON, ILLINOIS

December 2012

ABSTRACT

Hardware Error Rate Characterization with Below-nominal Supply Voltages

Ke Liu

Power dissipation and energy consumption have become one of the most important problems in the design of processors today. While lowering the operational voltage can reduce power consumption, there are limits imposed at design time, beyond which hardware components experience faulty operation. However, not all computations and all data in a workload need to maintain 100% fidelity. In this thesis, we explore the idea of Elastic Fidelity, that judiciously lowering the voltage to trade-off reliable execution for power consumption. By steering each computation to different functional and storage units, Elastic Fidelity Computing obtains power and energy savings while reaching the reliability targets required by each computational segment. We have characterized some execution units to build a correlation between supply voltage and error rate. We also present a simulation of faulty operation by injecting errors to a JPEG decompression program. The results shows that some execution unit can tolerate 50% of supply voltage without experiencing any error. It indicates great potential for applying Elastic Fidelity to more applications.

This work was partially supported by NSF award CCF-1218768.

Table of Contents

List of Tables

List of Figures

CHAPTER 1

Introduction

Power consumption is one of the most stringent constraints in processor design. Although transistor feature sizes are still scaling, voltage scaling has nearly stopped due to high leakage currents associated with low threshold voltages. This has lead to a dramatic increase in power density with decreasing feature size[**1**]. But the cooling limits remain unchanged across technologies, while transistor counts continue to grow exponentially. As a result, we can no longer power all transistors on chip; rather, we can power only a decreasing fraction of them.

At the same time, the increasing demand for computational power comes at the cost of higher aggregate energy consumption. The computers' insatiable appetite for energy has made IT industry a nonnegligible contributor for greenhouse gas emissions. Google's data centers consume 260 million watts today, which is about a quarter of the output of a nuclear power plant [**2**]. Similarly, the power gap between mobile phone processor and battery has greatly broadened over the last decade, resulting in shortened stand-by time.

The power consumption is high in part because the operating voltage of processors is determined by conservative guardbands based on worst-case scenarios. A guardband refers to the timing differential inserted into the hardware design to allow for signals to communicate without being perfectly aligned. However, this design approach results in significant overheads in both power and performance[**3**]. This leads to an interesting question: what if we let go of these guardbands and allow components of the processor

to fail sometimes, and accommodate the errors at the architectural and software levels? By following laws of transistor physics, keeping all else constant, decreasing the operating voltage (V_{dd}) reduces power consumption at a quadratic rate, at the expense of some timing errors. Prior research shows that every large CMOS chip has two voltage operating points: the rated voltage point and the critical voltage point [**4, 5**]. This leads to three operating regions for the processor. First, when the supply voltage is at or above the rated voltage, the processor runs at full accuracy without errors. Second, when the processor operates at a supply voltage between the rated and critical points, small-scale errors emerge due to timing violations in worst-case situations. Finally, operating at a voltage beyond the critical point leads to massive errors.

This thesis presents the idea of *Elastic Fidelity*. The basic approach is to occasionally operate processor components (e.g., functional units) at the supply voltage between rated and critical operating points, to attain significant reductions in power while meeting the reliability requirements requested by each section of the executing application. The errors originated due to this are accommodated at the software layer by exploiting the fact that different sections of the code require variable reliability guarantees to present acceptable results to the user. Portions of the application that are error-sensitive are executed at full reliability, while the error tolerant ones run on variable accuracy to produce an acceptable result. Similarly, error-tolerant sections of the data are stored on low-power units that allow for the occasional error by operating at low voltage levels and foregoing error-correction techniques (e.g., ECC), while the error-intolerant data are stored on units providing full reliability. By not treating all code and all data the same from the viewpoint of reliability requirements, Elastic Fidelity Computing exploits sections of the

computation that are error tolerant to lower power and energy consumption, without negatively impacting executions that require full reliability.

To explore the feasibility of Elastic Fidelity, this thesis evaluates the potential limit of cutting supply voltage and the resulted error rate. Start from OpenSPARC T1 processor at RTL level, some modules of interest are synthesized using Synopsys Design Compiler. Then Mixed-signal simulations are performed at circuit level, to characterize the error rate with each given supply voltage. Simulation results show that for some components, the supply voltage can decrease to as low as 50%, before noticeable errors start to appear.

The reminder of the thesis is organized as follows. Chapter 2 introduces background and related works for Elastic Fidelity and hardware characterization. Chapter 3 discusses the underlying idea of error tolerance and Elastic Fidelity. Chapter 4 describes the experimental methodology used to conduct the simulations. Chapter 5 reports the results of the simulations. Finally, Chapter 6 analyzes the results and Chapter 7 concludes.

CHAPTER 2

Background

This chapter introduces preliminary concepts used throughout this thesis. Section 2.1 starts with a brief discussion on the power and energy consumption of VLSI circuits. Section 2.2 talks about timing parameters and requirements. The contents in Section 2.1 and Section 2.2 also appear in [**6**]. Section 2.3 introduces processor architure. Specifically, the OpenSPARC model used in the thesis will be presented. The description about the model is obtained from [**8**]. Section 2.4 discusses related work.

2.1. Power and Energy Consumption of VLSI Circuits

Power and energy consumption plays an important role in all design hierachies. It influences many critical design decisions, such as the maximum operating frequency, power supply capacity, supply line sizing, packaging and cooling requirements[**6**]. Power consumption can be divided into two components: *dynamic* power and *static* power. Dynamic power occurs only during transitor switchings, while static power is persistent due to leakage current.

2.1.1. Dynamic Power

Consider a simple situation in which a PMOS transitor is driving a load capacitor C_L, as shown in Figure 2.1. Assume that initially, V_{out} is 0. When the capacitor C_L gets charged through the PMOS transitor, V_{out} rises from 0 to V_{DD}, and it stores a certain

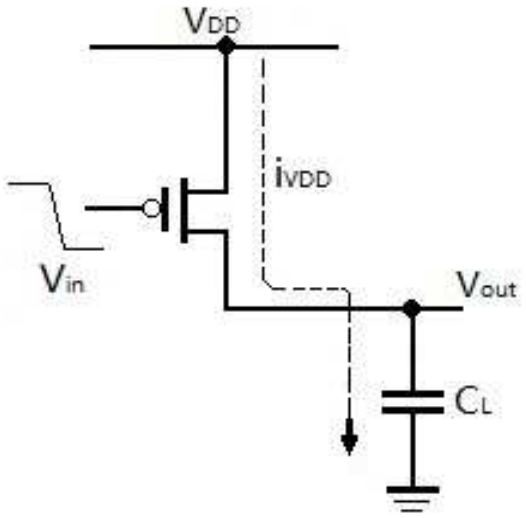

Figure 2.1. Dynamic power consumed during transtion[**6**]

amount of energy. Meanwhile, some amount of energy is dissipated by the PMOS transitor during the transition. A precise measure of the energy can be derived as follows. Let us consider the energy E_{VDD} taken from the power supply V_{DD}. It is the integral of its voltage and current over time.

$$E_{VDD} = \int_0^\infty i_{VDD}(t)V_{DD}dt = V_{DD}\int_0^\infty C_L\frac{dv_{out}}{dt}dt = C_LV_{DD}\int_0^{V_{DD}} dv_{out} = C_LV_{DD}^2 \tag{2.1}$$

Similarly, the energy store in E_L is

$$E_L = \int_0^\infty i_{VDD}(t)V_{out}dt = \int_0^\infty C_L\frac{dv_{out}}{dt}v_{out}dt = C_L\int_0^{V_{DD}} v_{out}dv_{out} = \frac{C_LV_{DD}^2}{2} \tag{2.2}$$

This shows that a half of the energy from supply is stored in E_L, while the other half is dissipated away during transition. Assume that E_L is discharged each time V_in switches from low to high, then every switching cycle takes an fixed amount of energy.

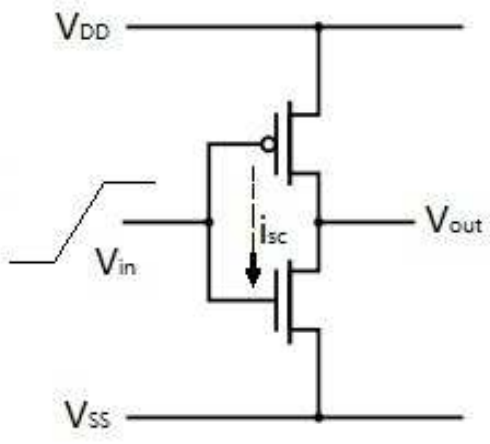

Figure 2.2. Direct-path current in CMOS inverter[**6**]

If the switching frequency is $f_{0\to1}$ times per second, the dynamic power consumption is given by

$$P_{dyn} = C_L V_{DD}^2 f_{0\to1} \tag{2.3}$$

Notice that V_{DD} has a quadratic effect on P_{dyn}.

Another type of dynamic power is the *Direct-Path Current Dissipation*. Consider an CMOS inverter in Figure 2.2 for example. In actual designs, V_{in} has a finite slope and there exists a short period in which the PMOS and NMOS transistor are conducting simultaneously. As a result, a direct-path current will flow from V_{DD} to V_{SS}. The resulting current spikes can be approximated as triangles and assumes that the inverter is symmetrical in its rising and falling responses, the energy consumed each switching period

is

$$E_{dp} = V_{DD}\frac{I_{peak}t_{sc}}{2} + V_{DD}\frac{I_{peak}t_{sc}}{2} = t_{sc}V_{DD}I_{peak} \tag{2.4}$$

where t_{sc} is the time period that both transistors are conducting. It is propational to the V_{in} transtion time. I_{peak} represents the maximum value of the direct-path current, and it is determined by the saturation current of the devices.

2.1.2. Static Power

The static power consumption of a circuit is defined as

$$P_{stat} = I_{stat}V_{DD} \tag{2.5}$$

where I_{stat} is the current that flows between the supply rails when no switching is taking place. Ideallly, when CMOS circuit is in steady state, either the PMOS or NMOS transistor should be in off condition, and no current can flow through. However, this ideal assumption has overlooked two factors.

First, a leakage current can flow through the reverse-biased diode junctions of the transistor, which are located between the source or drain and the substrate. This leakage current is very small in general. However, it is an exponential function of junction temperature. At 85°C, the leakage current increases by a factor of 60 over the room temperature value[**6**]. This, again, imposes an restriction on the power dissipation and cooling system.

Second, and more substantially, a leakage current called *Subthreshold Current* exits when gate-source voltage is below the threshold. In such a condition, a weak-inversion layer under the gate will provide carriers for the current. If the threshold voltage is small

in absolute value, it is easier to form a weak-inversion layer, and hence the larger the leakage current. In the past, the subthreshold conduction of transistors was very small, but as transistors scale, the threshold voltage has been significantly decreased. For a technology generation with threshold voltage of 0.2 V, leakage can exceed 50% of total power consumption[**7**]. On the other hand, given a certain supply voltage V_{DD}, scale the threshold voltage can improve the transistor rise/fall time, which is a desirable goal in circuit design. Thus, the threshold voltage can serve as an knob for performance-power trade-off.

Putting it together, the total power consumption is the sum of the dynamic and static part. Depending on the problem at hand, the total power can be measured in different ways. When considering supply line sizing, the peak power P_{peak} gives an indication of what the supply line has to sustain.

$$P_{peak} = i_{peak} V_{supply} = max[p(t)] \tag{2.6}$$

When studing heat dissipation, one usually chooses the average power P_{avg}.

$$P_{avg} = \frac{1}{T}\int_0^T p(t)dt = \frac{V_{supply}}{T}\int_0^T i_{supply}(t)dt \tag{2.7}$$

Other metrics such as *power-delay product* and *energy-delay product* are also commonly used to evaluate circuits.

2.2. Timing Requirements for Circuit Design

The timing requirements for conbinational circuit is straightforward. The delay of the block is simply the sum of gate propagation delays t_{plogic} along the critical path. To

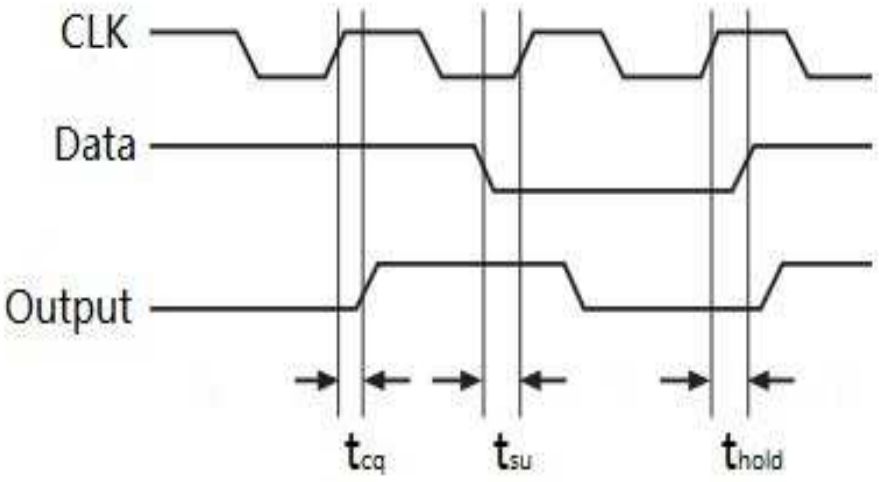

Figure 2.3. Setup time, hold time, and propagation delay of a synchronous register[**6**]

guarantee observing the correct value, one has to wait for t_{plogic} after feeding input values. When it comes to sequential circuit, the problem is more complicated. We will focus on synchronous sequential circuit in this section.

In order to latch the input, there are three important timing parameters a register has to meet. There are shown in Figure 2.3. The *setup time*(t_{su}) is the minimum time an input data has to keep steady before CLK edge comes. The *hold time*(t_{hold}) is the minimum time the input data has to hold steady after the CLK edge comes. Then, the input data is copied to the output after a *propagation delay*(t_{cq}).

Once all timing information for the registers and the combinational circuits is available, we can derive the system level timing constraints. The clock period T must give enough time for the combinational block to settle, and for registers to latch and copy the data.

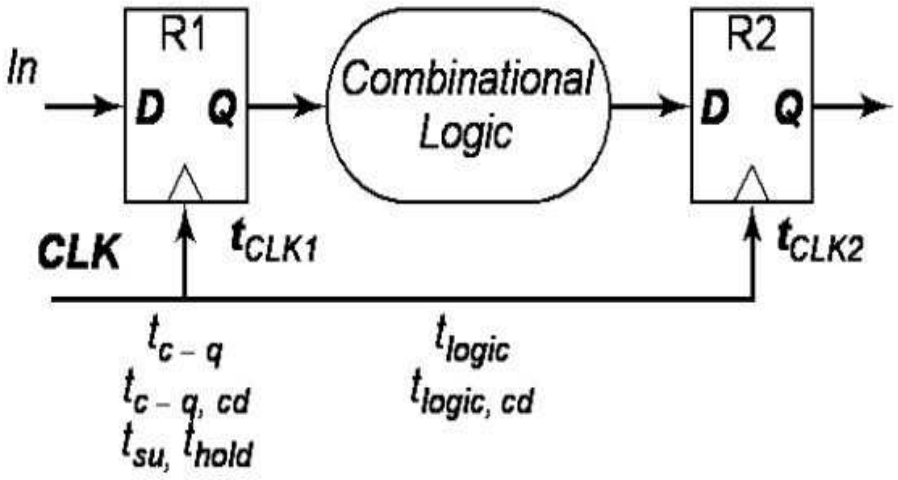

Figure 2.4. Two registers driving by unbalanced clock path

Assume the worst case propagation of combinational circuit is t_{plogic}, T must satisfy

$$T \geq t_{cq} + t_{plogic} + t_{su} \tag{2.8}$$

Moreover, the hold time for the register needs to be

$$t_{cdregister} + t_{cdlogic} \geq t_{hold} \tag{2.9}$$

where $t_{cdregister}$ is the minimum propagation delay of the register, and $t_{cdlogic}$ is the minimum delay of the combinational block.

The above timing specification is based on the assumption that the clock is ideal across all operating points and over all operating time. However, this is not the case. Consider the data transfer between two registers R1 and R2 as shown in Figure 2.4. If the two clock lines are not balanced, meaning that they are different in either length, resistance,

or capacitance, clock signals going through them will experience different delays. The spatial variation in arrival time of a clock transition on an integrated circuit is commonly referred to as *clock skew*. It is constant from cycle to cycle.

The clock skew has important effects on circuit performance and functionality. If CLK2 lags CLK1 by clock skew δ, then the time available for a signal to propagate from R1 to R2 is increased by the skew δ. Notice δ here can be either positive or negative. The constraint on the minimum clock period can be rewritten as

$$T + \delta \geq t_{cq} + t_{plogic} + t_{su} \tag{2.10}$$

The hold time now has to satisfy

$$t_{cdregister} + t_{cdlogic} \geq t_{hold} + \delta \tag{2.11}$$

Clock can also vary temporally, on a cycle-by-cycle basis. This variation is called clock *jitter*, which is often specified at a given point. Jitter is a zero-mean random variable. The sources of jitter include variation in clock-signal generation, power supply variation, and capacitive coupling. Jitter imposes extra restriction on clock period. In the worst case scenario, CLK1 arrives earlier with time t_{jitter}, and CLK2 is delayed by t_{jitter}. In this case, T has to satisfy

$$T - 2t_{jitter} \geq t_{cq} + t_{plogic} + t_{su} \tag{2.12}$$

Combining clock skew and jitter together, we have

$$T \geq t_{cq} + t_{plogic} + t_{su} - \delta + 2t_{jitter} \tag{2.13}$$

$$t_{hold} < t_{cdlogic} + t_{cdregister} - \delta - 2t_{jitter} \tag{2.14}$$

2.3. OpenSPARC T1 Processor Architecture

A processor is the heart of a computer. It executes programs and communicates with other components such as memory and I/O. The fundamental operation of a processor is to execute a sequence of instructions from system or user programs. The execution process of an instruction can be decomposed into five basic stages: fetch, decode, execute, memory, and writeback. These stages involve independent hardware resources and can be overlapped over consecutive instructions. Such a design is called a *pipeline*. It allows instructions to go through one stage each clock cycle(sometimes an instruction needs multiple cycles), and one instruction can be committed each cycle. Today, designs typically employ a much larger pipeline-depth, and instructions can execute in an order different than the order they are issue. However, the basic concept of pipeline design (overlap instructions to gain high thoughput) remains valid.

The processor model we used in the thesis is called OpenSPARC T1. OpenSPARC is an open-source project that started in December 2005. It provides register-transfer level verilog code of the full 64-bit, 8 cores processor OpenSPARC T1. Each SPARC core has an instruction cache, a data cache, and fully associative instruction and data translation lookaside buffers. The eight SPARC cores are connected through a crossbar to an on-chip unified level 2 cache[**8**]. The block diagram of OpenSPARC T1 is shown in Figure 2.5.

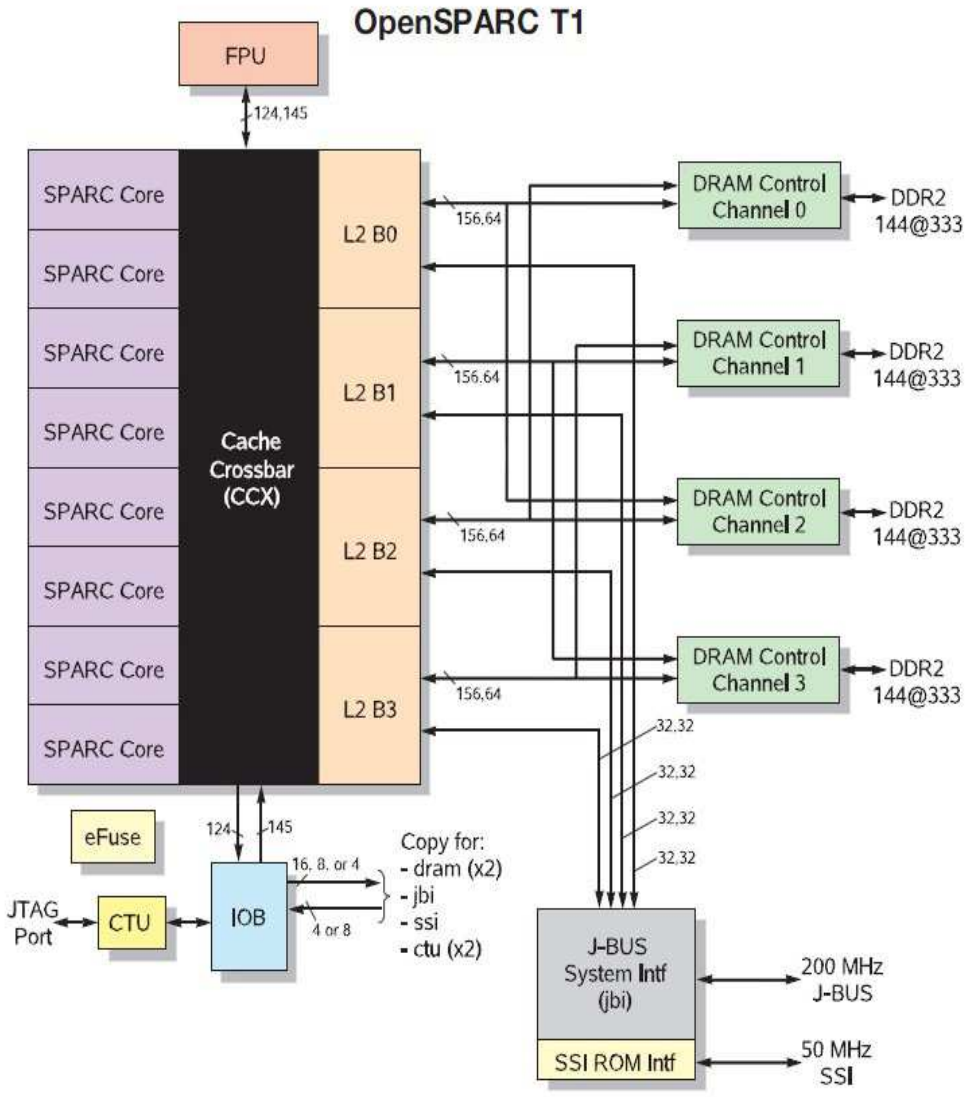

Figure 2.5. OpenSPARC T1 processor architecture[**8**]

Each SPARC core is hardware multi-threaded to support four threads. It has a single issue, six stage pipeline. These six stages are fetch, thread selection, decode, execute, memory, and write back. The structure of the SPARC core pipeline is shown in Figure 2.6. Each SPARC core is implemented by the following units:

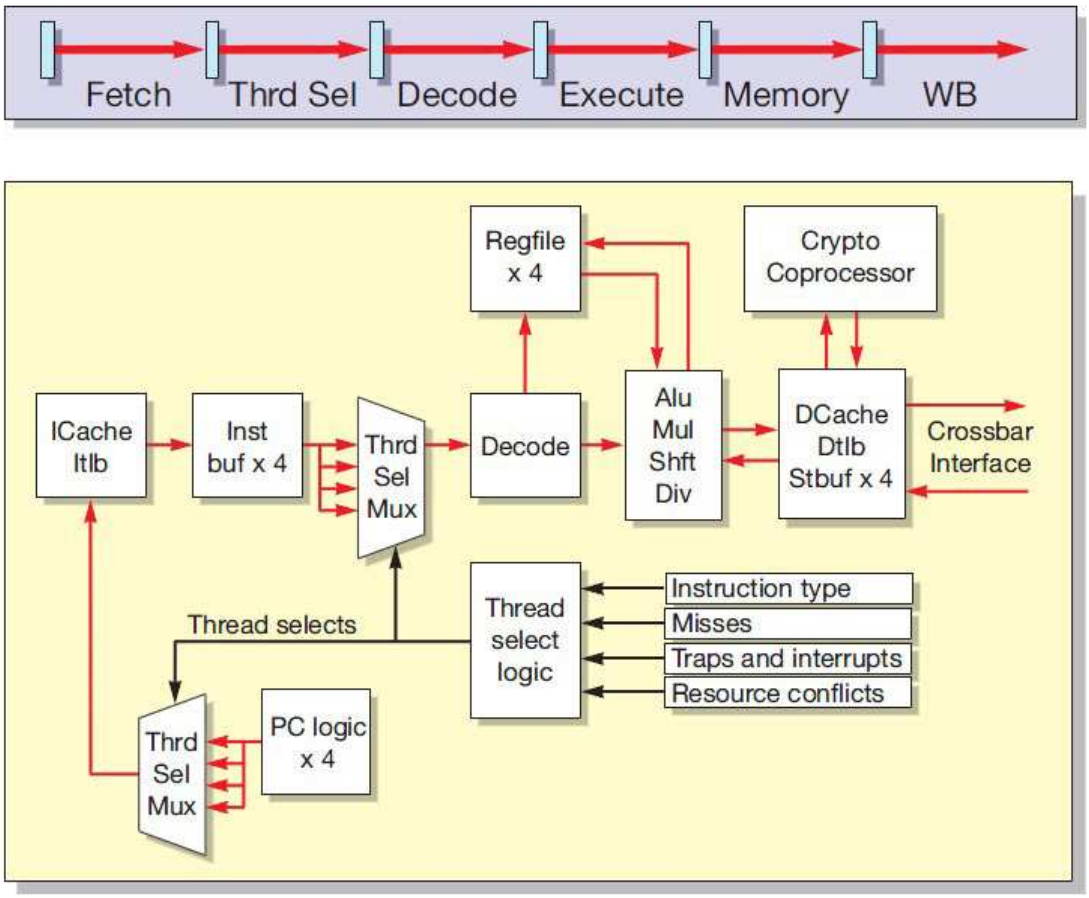

Figure 2.6. SPARC core pipeline[**8**]

1. Instruction fetch unit(IFU) includes the fetch, thread selection, and decode pipeline stages.
2. Execution unit(EXU) includes the execute stage of the pipeline.
3. Load/store unit(LSU) includes memory and writeback stages.
4. Trap logic unit(TLU) includes trap logic and trap program counters.
5. Stream processing unit(SPU) is used for modular arithmetic functions for crypto.
6. Memory management unit(MMU).

7. Floating-point frontend unit(FFU) interfaces to the FPU.

A single floating-point unit(FPU) is shared by all SPARC cores. It is composed of sub-units including:

1. Addition unit(FPU_ADD) that performs adds, subtracts, compares and conversion.
2. Multiplication unit(FPU_MUL)
3. Division unit(FPU_DIV)

Each sub-unit can take 32-bit or 64-bit operands, and has independent execution pipelines.

The CPU-cache crossbar(CCX) manages the communication among the eight CPU cores, the four L2-cache banks, the I/O bridge, and the floating-point unit. These functional units communicate with each by sending packets, and the CCX arbitrates the packet delivery[**8**].

SPARC T1 has two levels of caches on chip. The L1 D-cache is included in the load/store unit, and has 8K capacity. The L1 I-cache is managed by the instruction fetch unit, and has 16K of data. The L2 cache is 3 Mbytes in size and are composed of four symmetrical banks[**8**].

2.4. Related Work

There has been in depth research in modeling the behavior of hardware due to voltage scaling and process variability [**15, 16, 17**], along with techniques to prevent them [**18, 19, 20**]. Designs such as Razor [**21**] perform error correction at the hardware through the use of additional circuitry, while [**5**] corrects the errors at the algorithmic level. The benefits of these techniques in the scope of power reduction are limited due to their error

recovery overheads. Moreover, they are orthogonal to Elastic Fidelity and can be used synergistically with it to lower the power and energy consumption even further.

The most significant works related to Elastic Fidelity appear in [**30, 31, 32**] which aggressively scale supply voltage and allow faulty operation on execution and storage units. In the framework of [**30**], fidelity requirement is denoted by programming language at a coarse-grain level, while we believe multiple fidelity levels will provide the user with more confidence and freedom in trading off between accuracy and power. More importantly, there is a lack of hardware simulation in these projects, and they assume low power is applied to all execution units. We choose to characterize each unit and select the most resilient units.

On the software side, a considerable amount of research has been performed on the effect of single event upsets on software behavior [**24, 25, 9, 26, 27**] due to the rising reliability issues resulting from decreasing feature size. However, we find that not much has been done in the case of continued errors as presented in this paper. Unlike single event upsets, these errors occur continuously due to faulty hardware (as a result of voltage over-scaling in this case.)

Finally, research on empathic systems [**28, 29**] considers human perception and user satisfaction to guide power optimizations. Contrary to our work, empathic systems do not trade-off accuracy for power; rather, they trade-off user satisfaction for power. However, similarly to empathic systems, our output quality metrics for the applications we study in this paper are also exploiting human perception to arrive at a result that is good enough, but not necessarily perfect.

CHAPTER 3

Error Tolerance and Elastic Fidelity

Traditionally, program execution is said to be correct if and only if the underlying computations are perfect. However, a program may still appear to execute correctly if it returns acceptable results from the users' perspective, even if there is some noise in the data or inaccuracies in the computation [**9**].

Prior work such as that in [**9**] shows that the level of error tolerance is application-dependent and depends on how accurate a program's output needs to be. There are applications which are highly resilient inherently and there are others which are very little. Important examples of highly-resilient applications come from the class of soft computing. Unlike hard or exact computing, soft computing takes advantage of the tolerance of imprecision, uncertainty and approximation for a given problem resulting in acceptable rather than exact results [**10, 11**]. Multimedia applications offer a very interesting example of soft computing. These applications primarily depend on human perception and allow considerable leeway in terms of accuracy. Moreover, such applications are typically included in modern mobile platforms and are heavily exercised by users. Similarly, there are applications that already assume unreliable substrates and already have error-correcting capabilities (e.g., networking applications [**12**]). Other examples include Artificial Intelligence applications on forecasting, inference and data mining, scientific computing (e.g., simulations of oceanic currents, weather forecasting), or computations on already noisy

data (e.g., sensor readings). Such workloads tend to perform computations on approximations, and through multiple iterations narrow down to a set of results that are within a qualitative threshold according to the user requirements. On the other hand, other applications rely on exact numerical results and are generally intolerant of errors. Examples include memory management in the operating system, code compilation and lossless data compression.

Looking further into error-resilient applications, we envisage that different portions of the execution offer different error tolerance. For example, pointer operations and control logic such as conditional and branch statements are highly sensitive to errors. A corrupt pointer would usually lead to a segmentation fault while a corrupt control would notably disorder the program execution. On the other hand, operations involving standard arithmetic computations such as matrix processing and decoding are generally error-tolerant and have a relatively benign effect on the final result of the program.

In view of these observations, Elastic Fidelity varies the reliability of the underlying hardware according to the application needs at each point in time. Portions of the application that are error-sensitive are executed at full reliability, while the ones that are error-tolerant are run on variable accuracy to produce an acceptable result to the end user. The reliability of the underlying hardware can be varied by running it on a region between the rated and critical voltage points, resulting in power savings. This scheme can be implemented at the granularity of a core, in both single and multi-core systems. In the former case, the operation of the core can be dynamically changed between full (100%) and variable accuracy, according to the requirements of the executing code segment. In the latter case, certain cores run on full accuracy to accommodate the error-sensitive

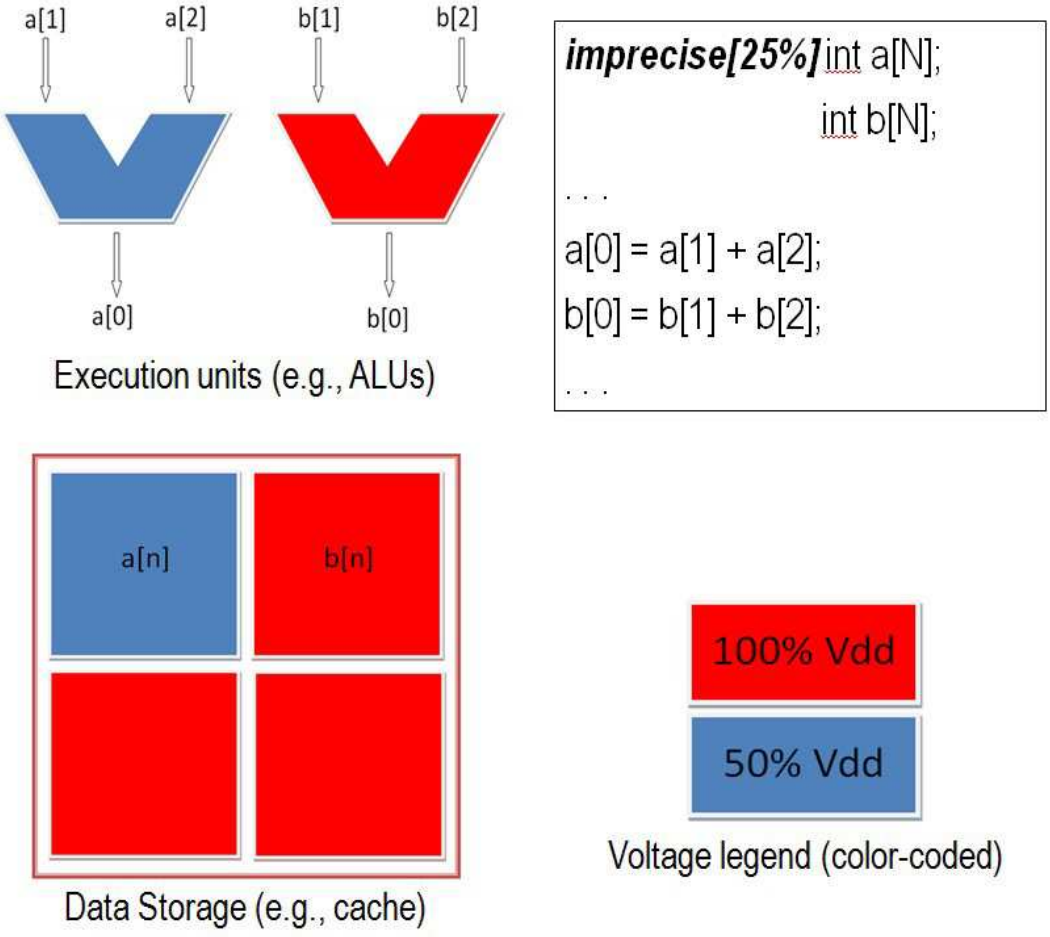

Figure 3.1. Implementing Elastic Fidelity using multiple execution units

operations, while others run on variable accuracy to compute error-tolerant operations. Similarly, Elastic Fidelity can be implemented at a finer granularity by varying the fidelity guarantees of individual functional units or storage elements within a single core. Figure 3.1 illustrates such an example with multiple execution and storage units.

From the viewpoint of software design, Elastic Fidelity can be implemented through programming constructs. A programmer specifies which variables and code segments are allowable to hardware errors and their tolerance margins. In turn, the compiler maps

these constructs to specialized instructions that direct the core to steer the computation to a functional unit with a specific reliability level, by changing its operating voltage. On the hardware end, dynamic voltage scaling and calibration circuitry minimize the power consumption at a given reliability level, based on experimental models of hardware behavior at each voltage level. The fidelity requirements of each code/data segment can be estimated using feedback optimization tools.

Let us now take a close look at what happens inside an execution unit when the supply voltage is below the rated point. As we know, decreasing the supply voltage has a negative impact on the circuit performance. More specifically, the rise/fall time of each gate increases, and a signal takes longer time to propagate to the outputs of combinational blocks. When the propagation delay is increased beyond the clock period, registers will face the risk of latching immature data. If immature data happen to be the same as correct data, which arrive after the clock edge, the user will not perceive an error. However, if they are different, a timing error appears.

In order to get an indepth perspective of the feasibility of Elastic Fidelity, and exploit the full potential of error-resilience, two categories of studies have to be done. First, at the hardware level, we need to characterize the execution units of interest, and establish the correlation between supply voltage and error rate. Since each execution unit tends to have a distinct critical path, and hence different timing slack, lowering the supply voltage may result in a different rate of timing errors. The result will help us find out which units are more resilient, and to what extent we are allowed to lower the voltage. This thesis mainly studies this topic. Second, we need to understand how errors in different

code/data segments under reduced reliability conditions impact the overall accuracy of the end result. The next chapter will introduce the methodology in detail.

CHAPTER 4

Experimental Methodology

This chapter describes the methodology used in analysing hardware error rate and software error tolerance. We take three execution units from the OpenSPARC T1 processor RTL model, and synthesize them using Synopsys tools. Mixed-signal simulation is conducted at the netlist level. For the software side, we refer to the experiment conducted by fellow researcher at Northwestern University. In the research, we manipulate the assembly code of a JPEG decompression program. Errors are generated by software wrappers at bit level with designated probability. Then the program output is compared against error-free counterpart.

4.1. Characterizing Hardware Components

4.1.1. Execution Units Specifications

The three execution units we simulate are FPU_ADD, EXU_ALU, and FPU_MUL. They cover both integer and floating-point operations. OpenSPARC T1 is designed to operate at 1.2GHz. However, since only a 90nm cell library is available to our research project, which lags behind in technology generations, we are not able to simulate the processor at 1.2GHz. Instead, the simulated operating frequency is set at 500MHz, which is the maximum frequency without any timing violations under the full supply voltage. With 500MHz, all these three execution units can generate correct results at the full voltage level. The detail is shown in Table 4.1.

Execution Unit	Operations It Performs	Execution Delay
FPU_ADD	floating-point addition,floating-point subtraction, floating-point comparison	10ns
EXU_ALU	integer addition, logic AND, logic OR, logic XOR, logic MOVE	2ns
FPU_MUL	floating-point multiplication	17.5ns

Table 4.1. Simulated executing units

4.1.2. Synthesis

We use Synopsys Design Compiler(Version F-2011.09-SP3) and SAED90nm cell library to synthesize execution units. Table 4.2 lists some basic parameters of SAED90nm library. OpenSPARC provides a PERL script which contains the setup and constraint for

Techology	90nm
Typical Voltage	1.2V
Operating Temperature	25°C
Operating Frequency	300MHz
Number of Cells	340
PMOS Threshold Voltage	-0.276V
NMOS Threshold Voltage	0.397V

Table 4.2. SAED standard cell library

compiling and optimization. We have modified it for our need and use it for synthesis. An synthesized EXU_ALU netlist is shown in Figure 4.1.

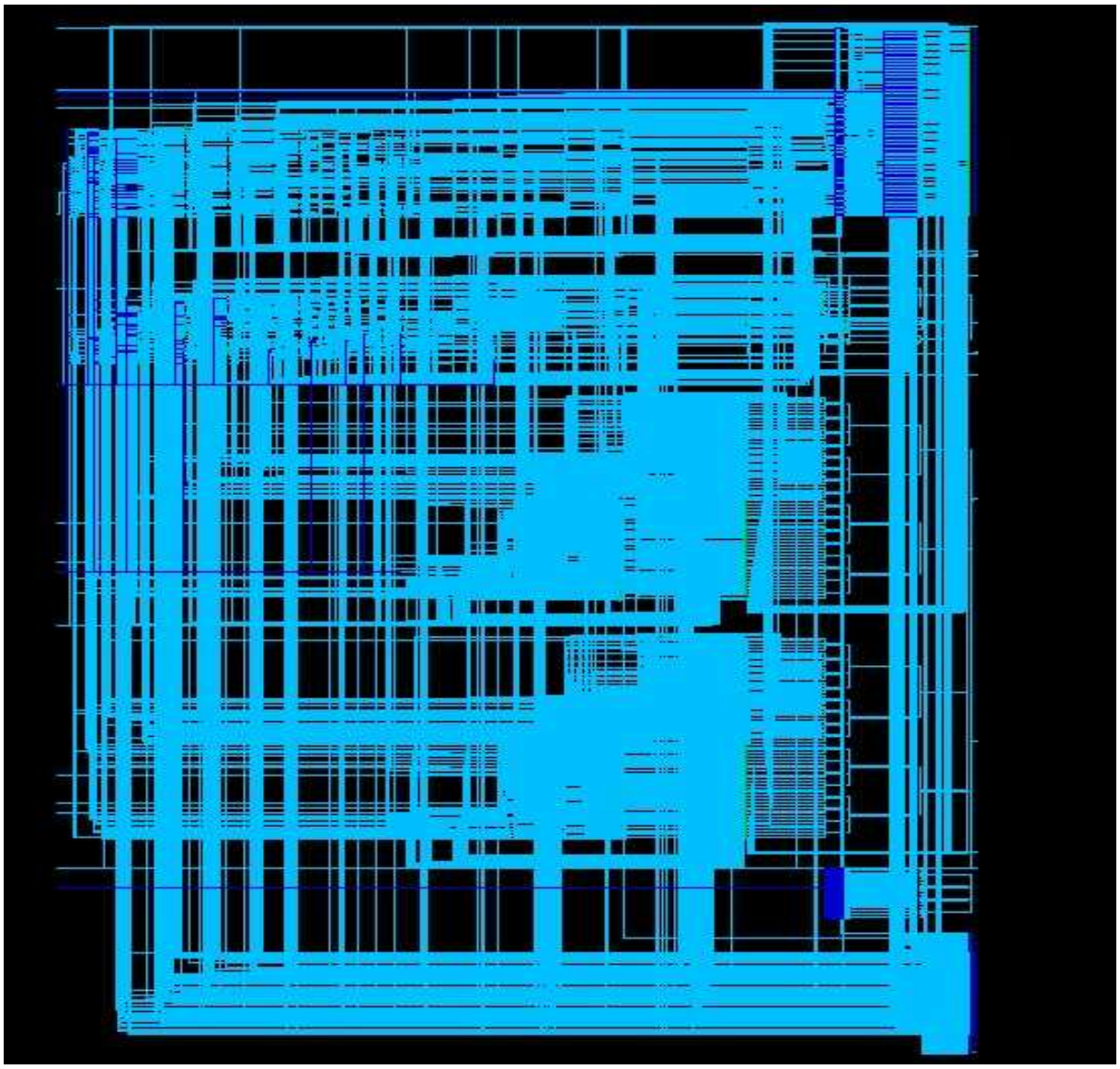

Figure 4.1. Gate level schematic of EXU_ALU

4.1.3. Test Benches

Each execution unit takes two 64-bit operands as input, and can perform different operations on them. Thus, the evaluation space is huge (approximate 3.5×10^{38} testing points for each operation) and it is impossible to cover all of it. Moreover, when timing

error occurs, the output we observe is likely to be the result latched by preceding input. This means the output will not only depend on current input, but also operands given before. To accommodate this possibility, different input sequences have to be tested, and the evaluation space will be further expanded.

Based on these concerns, to get a comprehensive and unbiased characterization, we design the test bench as follows.

1. In Input vector 1, both operands are set to be 0. One input vector refers to a pair of operands that are given to the execution unit together. Both operands are in the format of long long unsigned integer, which is 64-bit on our machine. Then we use two nested loops to increment operand A and operand B. In the inner loop, we increase operand A by a certain increment V_{incr}, which is also a long long unsigned number. In the outer loop, we increase operand B by the same value V_{incr}. V_{incr} is decided by both the size of the evaluation space (which is 2^{64} for one operand in our experiment) and the number of input vectors. We walk through the entire evaluation space with a fixed granularity indicated by the value of V_{incr}.
2. All input vectors (both operand A and B) are given a random variation V_{random}, where $|V_{random}| < V_{incr}$. Notice that, during this step, we have unintentionally converted the input vector to double precision floating-point format by implicit type casting, and later they are converted back to long long unsigned before the assignment. This leads to the undesirable lost of precision. As a result, lower bits (bit0 to bit8) in almost all operands are always '0'. However, we believe this will not affect the correctness of our experiment. We will explain the reason in next chapter.

3. We store all input vectors in binary formats. In this case, although they are generated as long long unsigned integers, the execution unit will view them as floating-point numbers, integers, or boolean values according to the type of the particular execution unit.
4. Randomly permutate all vectors.

Notice that for FPU_ADD, we only simulate floating-point addition, since floating-point subtraction and comparison are similar with addition in nature. We create 100,000 test vectors for FPU_ADD and EXU_ALU, and 1,000 vectors for FPU_MUL. Figure 4.2 shows a scatter diagram of all input vectors for FPU_ADD in log space. As we can see, input vectors are uniformly scattered in the range of 10^{-302} and 10^{302}, which is the arithmetic range of a double-precision floating-point number.

4.1.4. Mixed-signal Simulation

We configure Synopsys VCS(F-2011.12) and HSIM(F-2011.09-SP1) to run mixed-signal simulations. Both Verilog model and SPICE model of all cells are given to the simulator. The SPICE model contains physical information of both the gate and the composing transistors. It provides the simulator with enough details to simulate below-nominal voltage effects. Signals are modeled in digital between each gate, and in analog within gates. The same sequence of randomly permutated input vectors are simulated 12 times, with voltage sweeping from 0.1V to 1.2V. Then the outputs of voltage level 0.1V to 1.1V are compared to the one from full voltage level (1.2V). If lowering the voltage has no impacts on the result, they will be identical. Otherwise, error rate of the result will be computed.

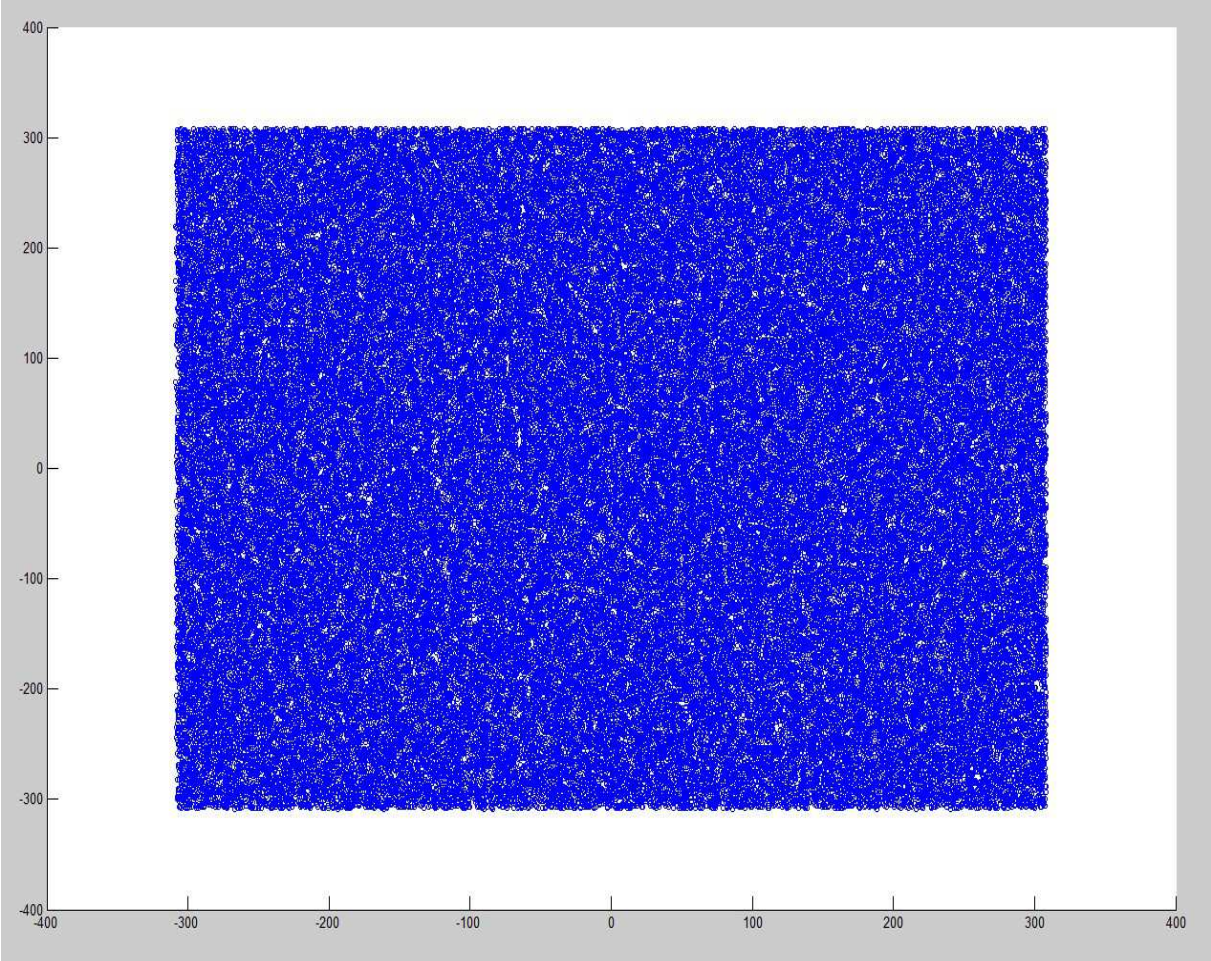

Figure 4.2. Scatter diagram of all input vectors. The X-axis and Y-axis are the exponent of operand A and operand B, respectively. One dot on the diagram indicates one input vector

4.2. Software Error Tolerance

This experiment is done by Georgios Tziantzioulis [**33**]. We simulate a multimedia JPEG decompression program for the experiment. In its assembly code, we only target arithmetic instructions. Moreover, all pointers and controlling variables are excluded to ensure program stability. The types of instructions we inject are list in Table 4.3. The error injection is implemented by software wrappers. These software wrappers model hardware

Instruction	Function
add	Adds two values
mov	Writes a value to the destination register
mul	Multiply two signed or unsigned 32-bit values
orr	Performs a bitwise OR of two values
rsb	Subtracts a value from a second value

Table 4.3. Injected instructions

timing errors by flipping one data bit in targeted instructions at a given probability (error rate).

The use of software wrappers allows flexibility in inserting errors in selective locations and helps us to study the behavior of the application not only when errors are injected in the entire computation, but also when they are injected during the execution of specific functions. To gain finer granularity, we vary the bit positions we flip, and the results are collected separately.

After error injection is done, we take a JPEG image as a sample, and run it through the decompressor. The quality of the decompressed file is then quantitatively measured in *Peak Signal to Noise Ration*(PSNR).

CHAPTER 5

Results and Analysis

In this chapter, we present experiment results of execution units characterization and software error injection. In section 5.1, the overall error rate, bitwise error rate and operand-related error rate are shown. Section 5.2 shows decompression quality with errors injected at different bit positions and with different probabilities.

5.1. Characterization of Execution Units

5.1.1. Overall Error Rate

We define two types of overall error rates. The first one is *Result Error Rate*. If any bit in an execution result is flipped (regarding of result from full voltage simulation), that result is considered wrong. Then the number of wrong results over the number of all results is the result error rate. The second type of error rate is the *Bit Error Rate*. It is simply the number of flipped bits over the number of all bits.

The overall error rates of three execution units are shown in Figure 5.1, Figure 5.2, and Figure 5.3.

As we can see, three execution units exhibit very different behaviours with below nominal supply voltage.

For FPU_ADD and FPU_MUL, timing errors occur right after we decrease the voltage, and error rate increases with lower voltages. When it comes to 0.6V or below, the result error rate is close to 100%, and bit error rate approaches 40%. Notice, when the bit error

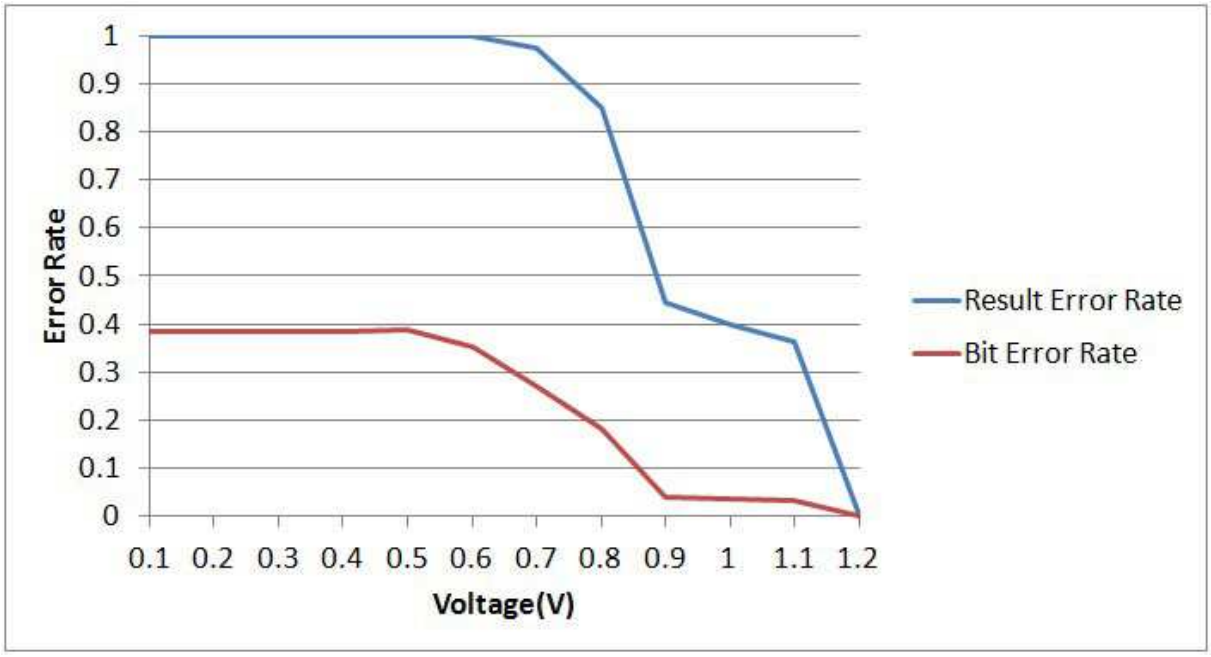

Figure 5.1. Overall error rate of FPU_ADD

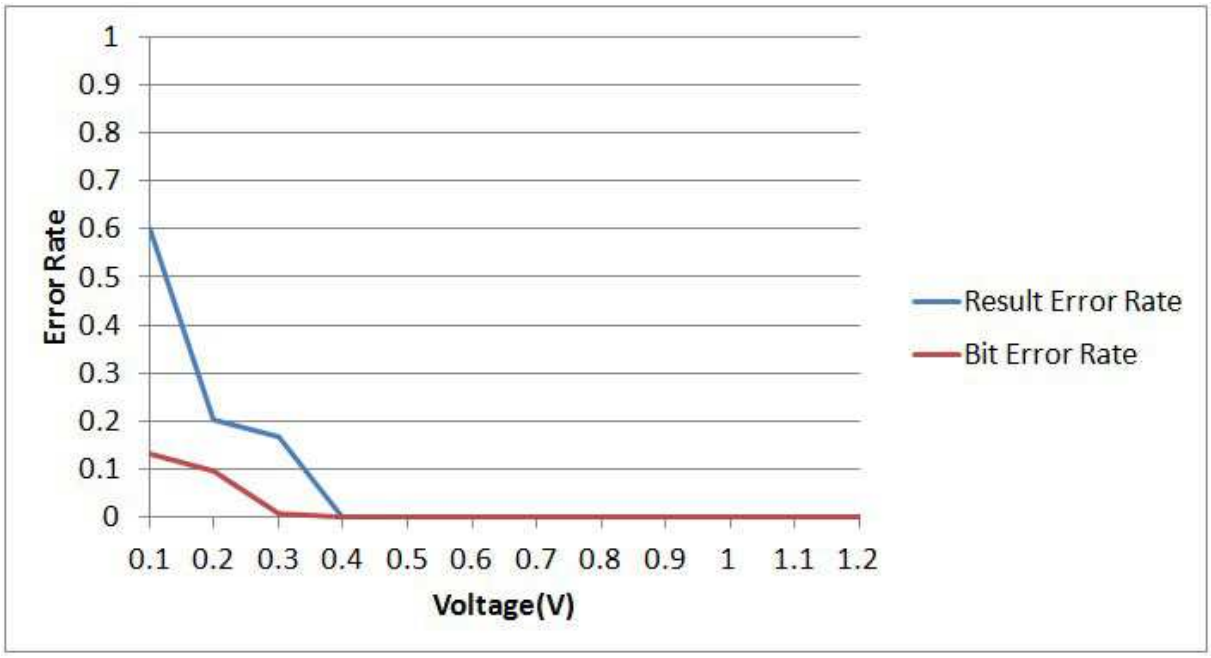

Figure 5.2. Overall error rate of EXU_ALU

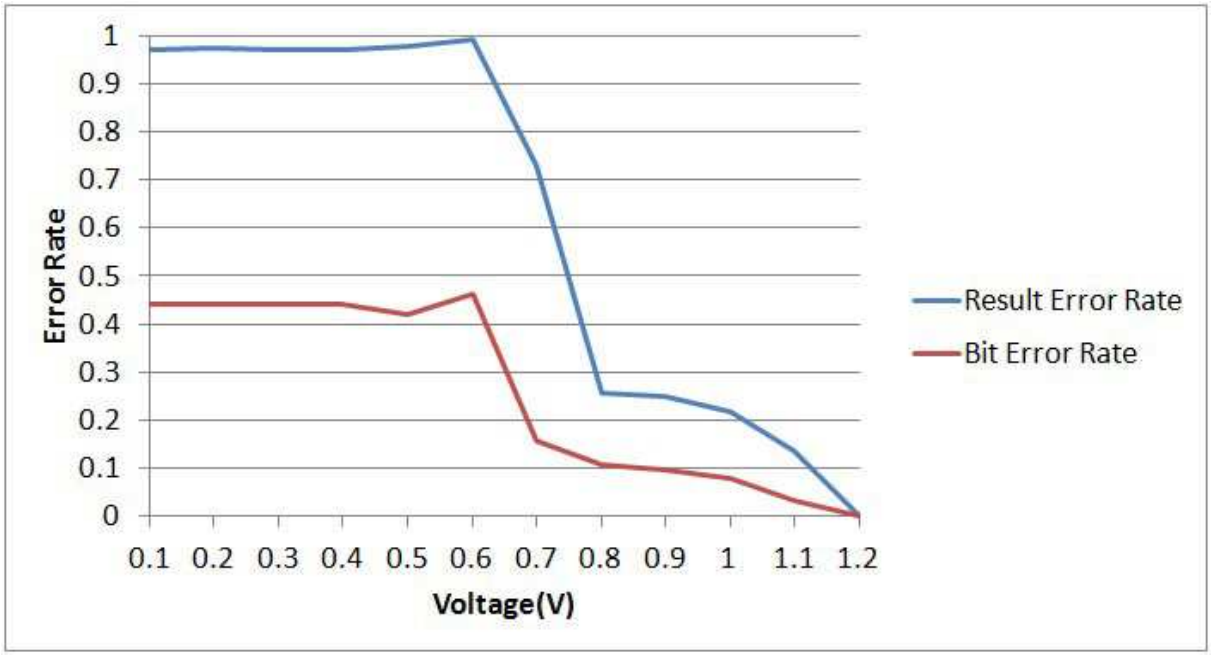

Figure 5.3. Overall error rate of FPU_MUL

rate reaches 50% or above in this experiment, it indicates the circuit does not function any more.[1] This indicates that almost all circuit paths have failed in timing. On the other hand, EXU_ALU is quite insensitive to lowering voltages. We do not see errors until the supply voltage drops to 0.4V.

The reason behind this is that floating point operations have multiple computation stages, and they tend to have long delays and small slacks. As long as the supply voltage is below rated point, timing errors start to occur. However, EXU_ALU performs relatively

[1]It can be understood as follows. First let us assume that the probability of '0's and '1's in the correct results are equal. If the simulation results are randomly generated and uniformly distributed, since each bit is binary, the bit error rate should be 50%. Now the simulation results are actually correct value disturbed by some timing errors, and the timing errors have a value latched by preceding input. Since we have randomly permutated all input vectors, the preceding input have an equal probability to generate a '0' or a '1' on each bit. Therefore, in the worst case that all bit positions have timing errors, we will observe a "random" value on each bit, which will give us 50% bit error rate. If the probability of '0's and '1's are not equal, the worst error rate can be the higher one of the probability of '0' and the probability of '1'. We will see such a case later.

Figure 5.4. Bitwise error rate of FPU_ADD

simple operations. For example, an AND operation can be implemented by parallel AND gates between two operands. The delay of the combinational block is very small, and the voltage margin is larger. This explains why until the voltage reaches 0.4V, which is the threshold voltage of the transistors, no timing error occurs on EXU_ALU.

5.1.2. Bitwise Error Rate

Next, the error rates on each bit position are presented. Figure 5.4 shows the result for FPU_ADD. Figure 5.6, Figure 5.7, Figure 5.8, Figure 5.9, and Figure 5.10 show per bit per operation result of EXU_ALU. Figure 5.11 is the result of FPU_MUL.

The error rate of FPU_ADD can be divided into three segments. From bit 0 to bit 8 the error rate is between 10% to 20%. From bit 9 to bit 51 the error rate is relatively

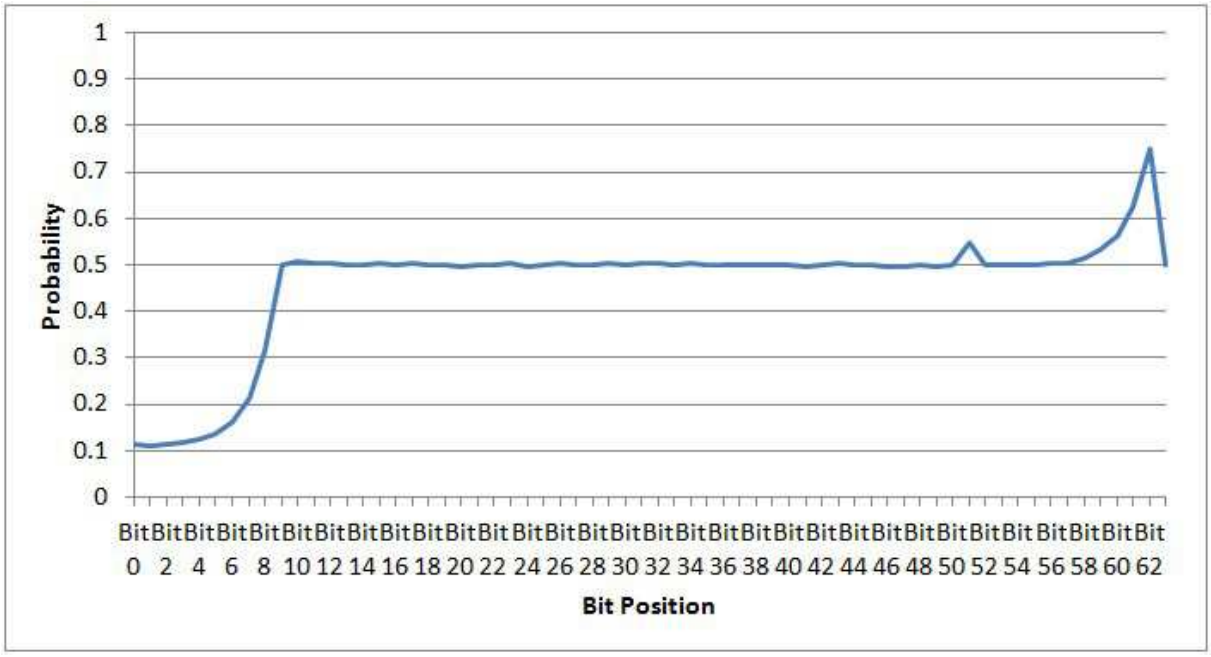

Figure 5.5. The probability of '1's on each bit position in the correct result from FPU_ADD

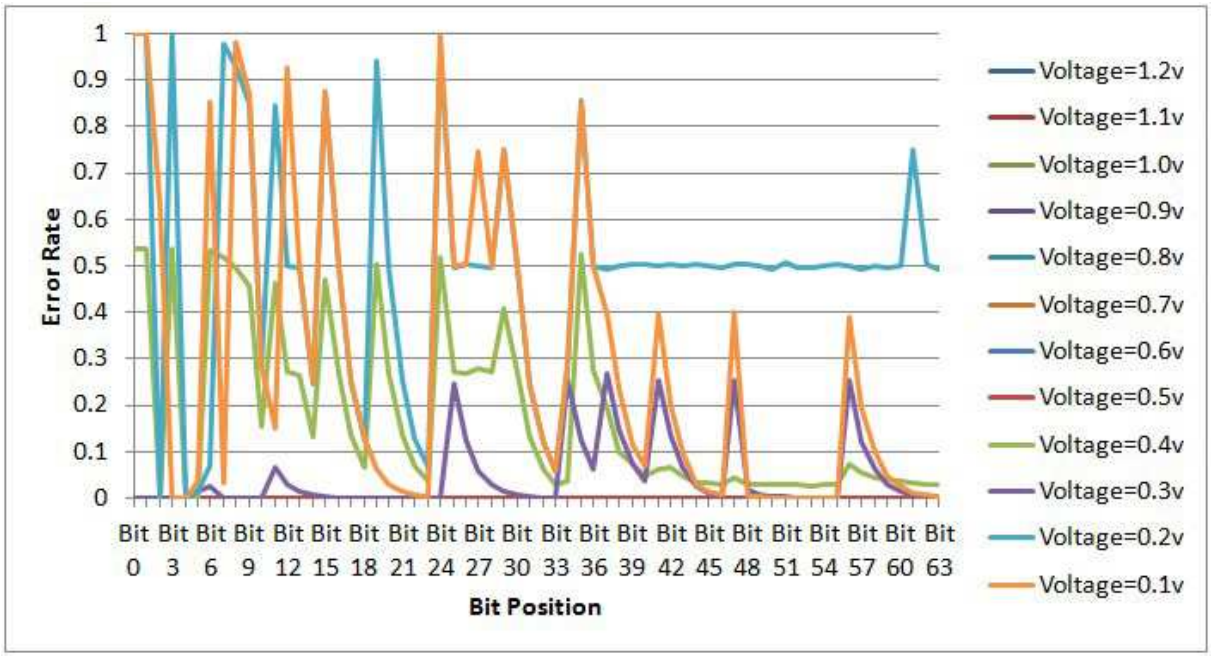

Figure 5.6. Bitwise error rate of EXU_ALU ADD operation

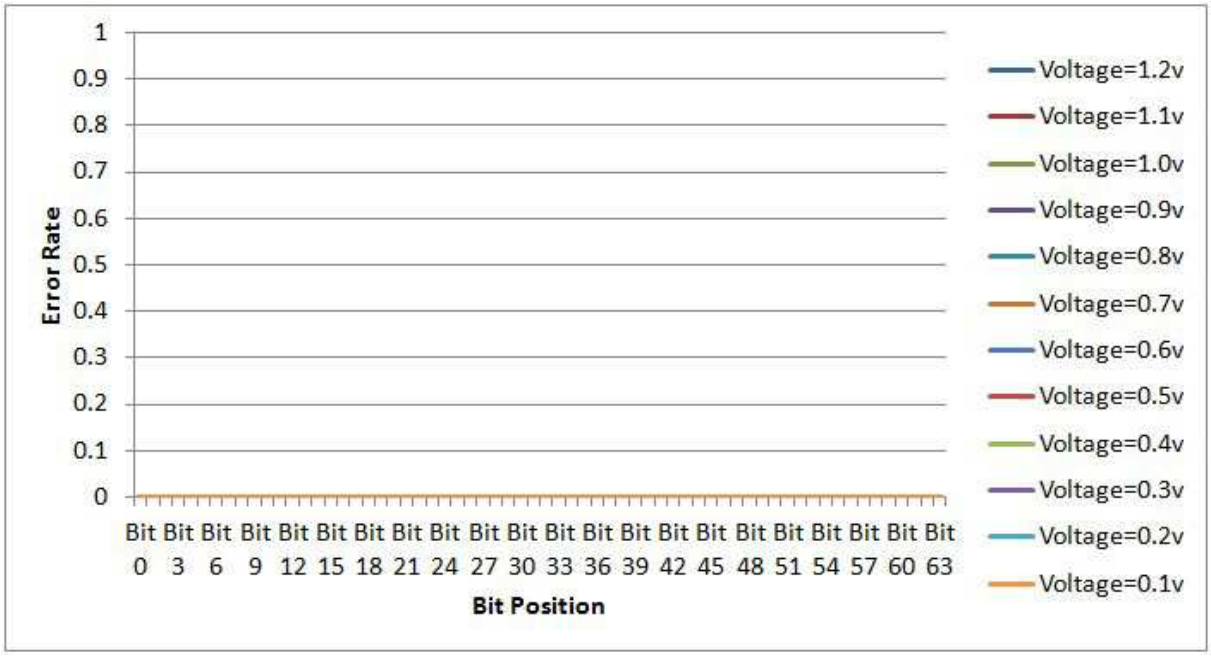

Figure 5.7. Bitwise error rate of EXU_ALU AND operation

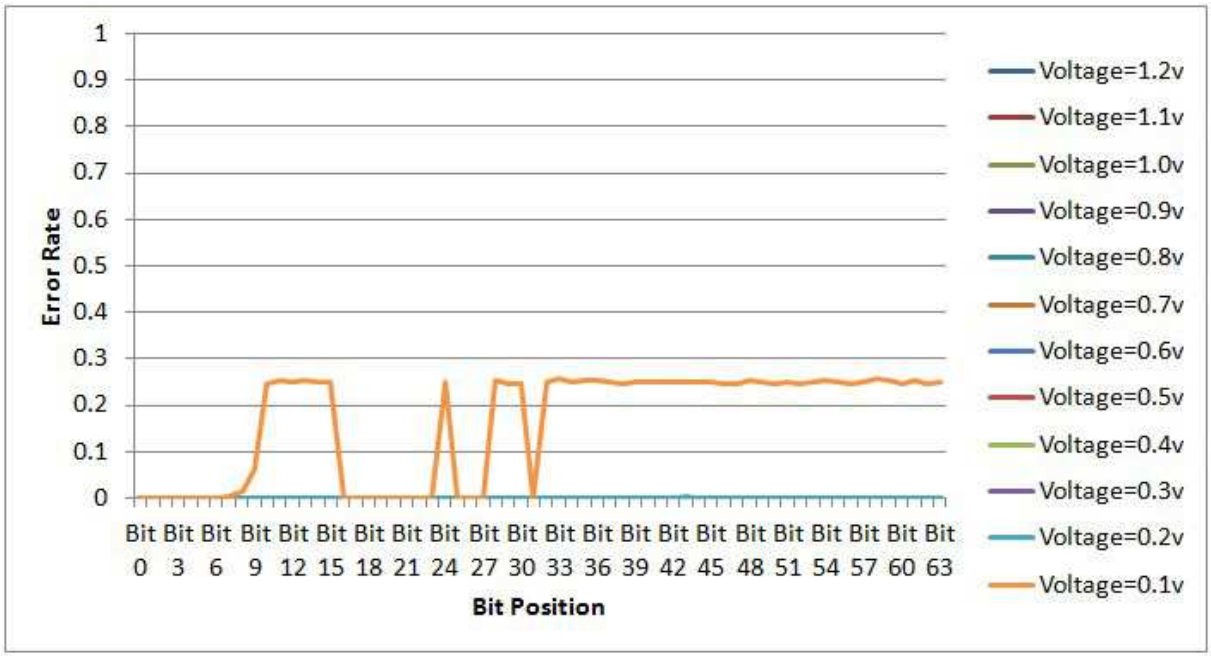

Figure 5.8. Bitwise error rate of EXU_ALU OR operation

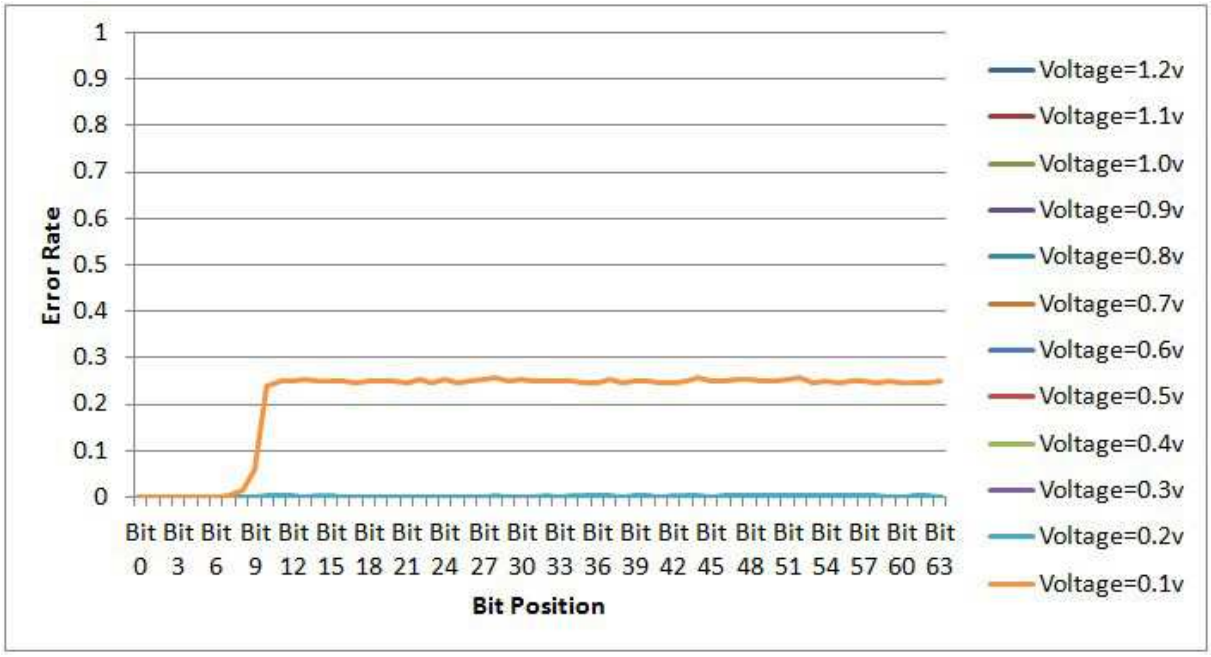

Figure 5.9. Bitwise error rate of EXU_ALU XOR operation

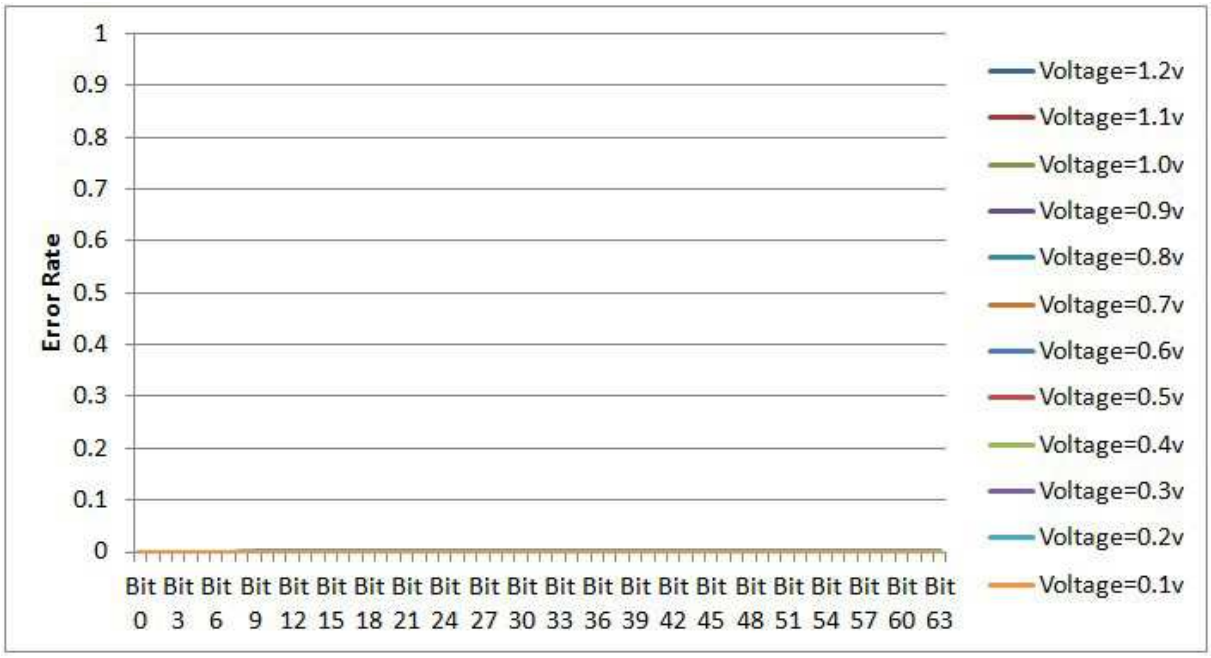

Figure 5.10. Bitwise error rate of EXU_ALU MOVE operation

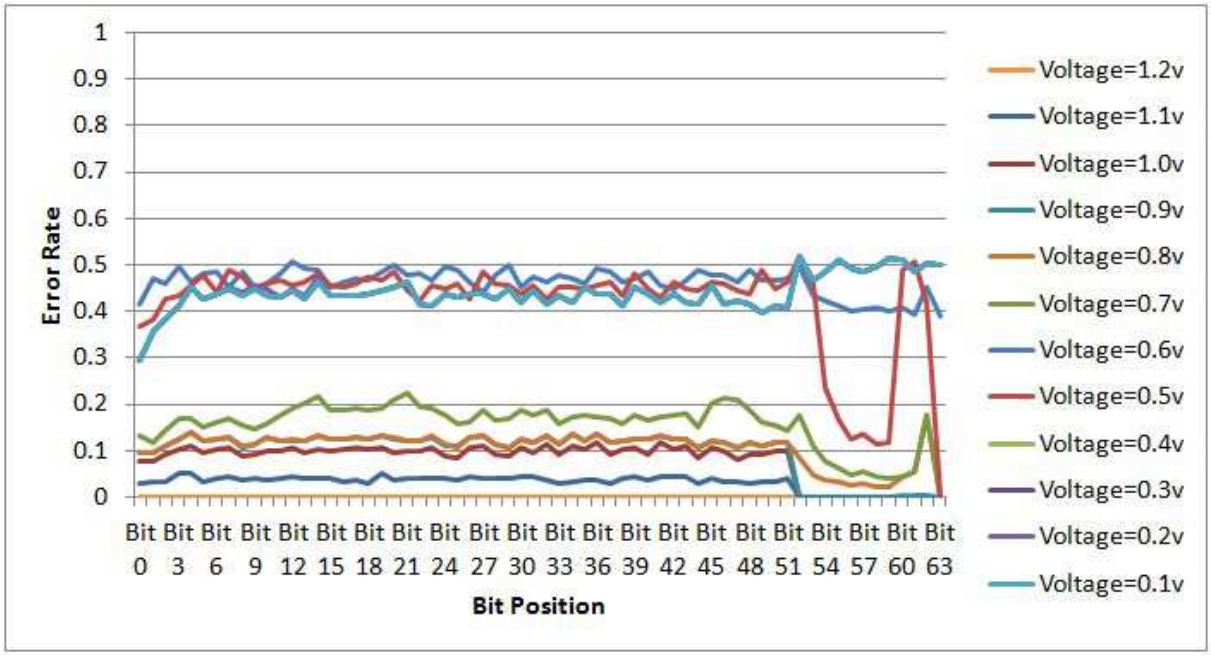

Figure 5.11. Error rate on each bit location of FPU_MUL

high and steady for most voltage levels, then it dives at bit 52. To understand this result better, let us look at Figure 5.5 first.

As it shows, the probability of '1's in the correct results is not the same across all bit positions. The fact that the probability is low in lower bits is caused by lost of precision during type casting (as mentioned in the previous chapter). This makes the lowest several bits in the operands always '0', and it is more likely for a result to contain '0' in lower bits. If we look at the bits unaffected by the type casting precision loss, the bit error rates remain constant for a given voltage. This is a strong indicator that the low order bits that are now erroneously cleared by the type casting operation, would also show the same error rate as the other bits. However, the affected bits are the lowest bits in mantissa, and they have less weight on the floating-point value. The critical timing path

are usually composed of other high order bits, so the correctness of the experiments is still maintained. Also notice that between bit 10 and bit 63, there are two spikes. We are performing addition on two uniformly distributed operands. Their sum, however, is not uniformly distributed in the arithmetic space. The sum has higher density in high magnitude area (close to positive and negative infinity), which is indicated by the spikes (larger percentage of '1's).

Then if we compare Figure 5.4 and Figure 5.5, we can find that they look similar except that the spikes become valleys. This is reasonable. If the correct results are biased (have different probability of '0's and '1's), then it is more likely for a bit to miss timing and have correct result accidentally. The more it is biased, the less is the error rate. The dive at bit 52 is caused by different subcomponents. FPU_ADD use two subcomponents to compute the exponent and mantissa separately. They have different timing slacks and exhibit different error rates.

The results of EXU_ALU have quite low error rate. The reason is the same as we explained before.

FPU_MUL also has a transition at bit 52, and it is caused by different timing slack in subcomponents as well.

5.1.3. Operand-related Error Rate

Next, we show operand-related error rate of FPU_ADD in Figure 5.12 to Figure 5.22. The error rate is given in a 3D space, in which the X axis and Y axis are the exponent of two operands, and Z axis is the number of flipped bits in corresponding output. We have color-coded each dot to make them more identifiable in 3D space.

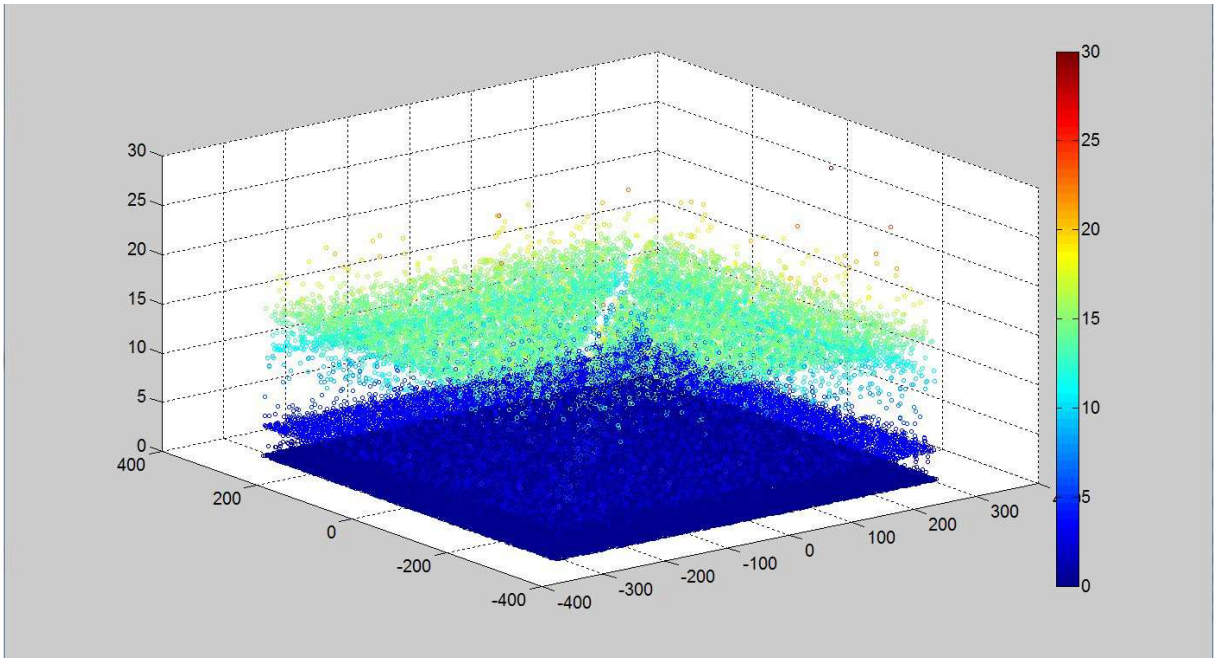

Figure 5.12. Operand-related error rate of FPU_ADD at voltage 1.1V

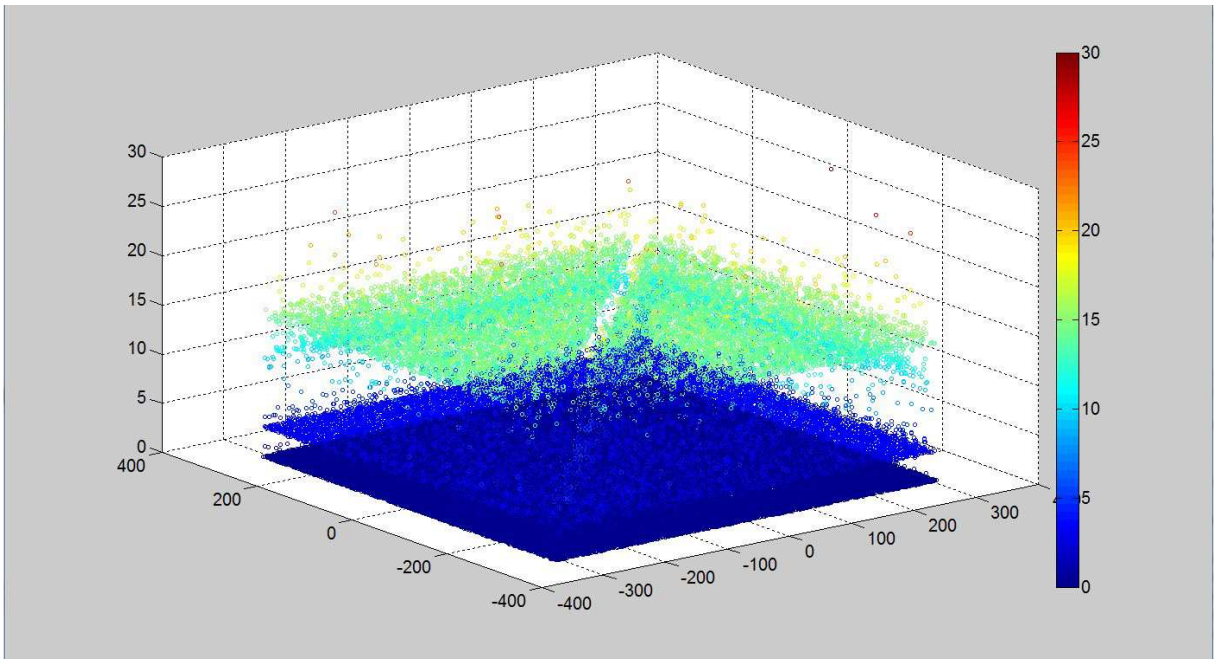

Figure 5.13. Operand-related error rate of FPU_ADD at voltage 1.0V

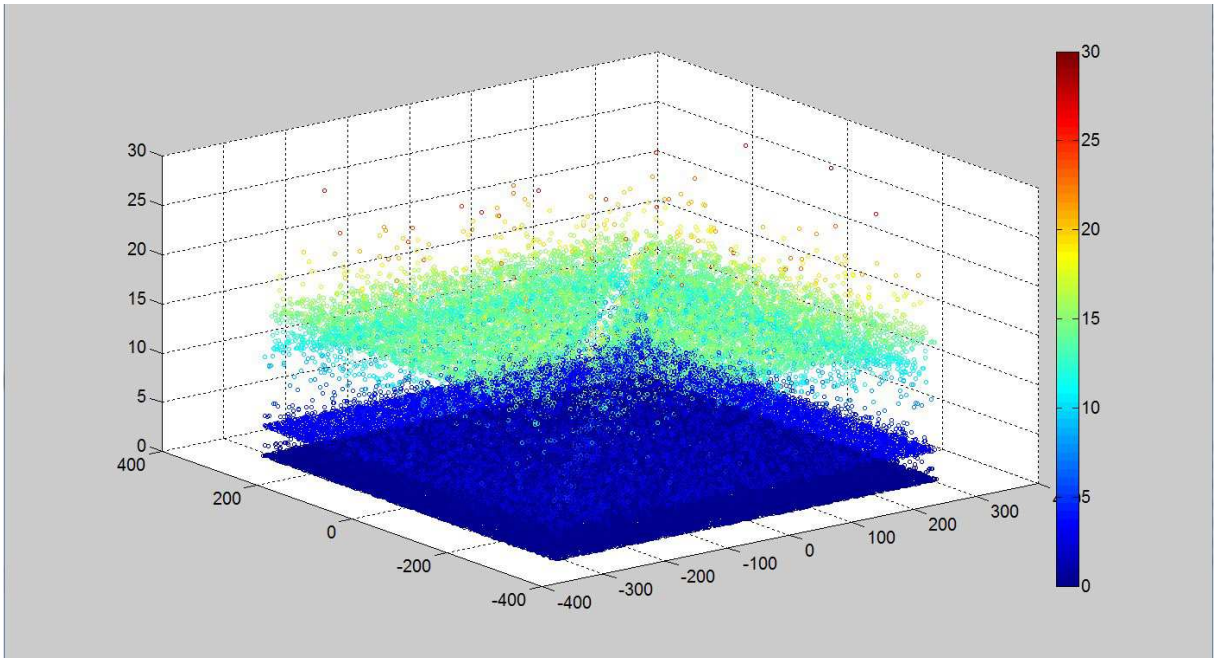

Figure 5.14. Operand-related error rate of FPU_ADD at voltage 0.9V

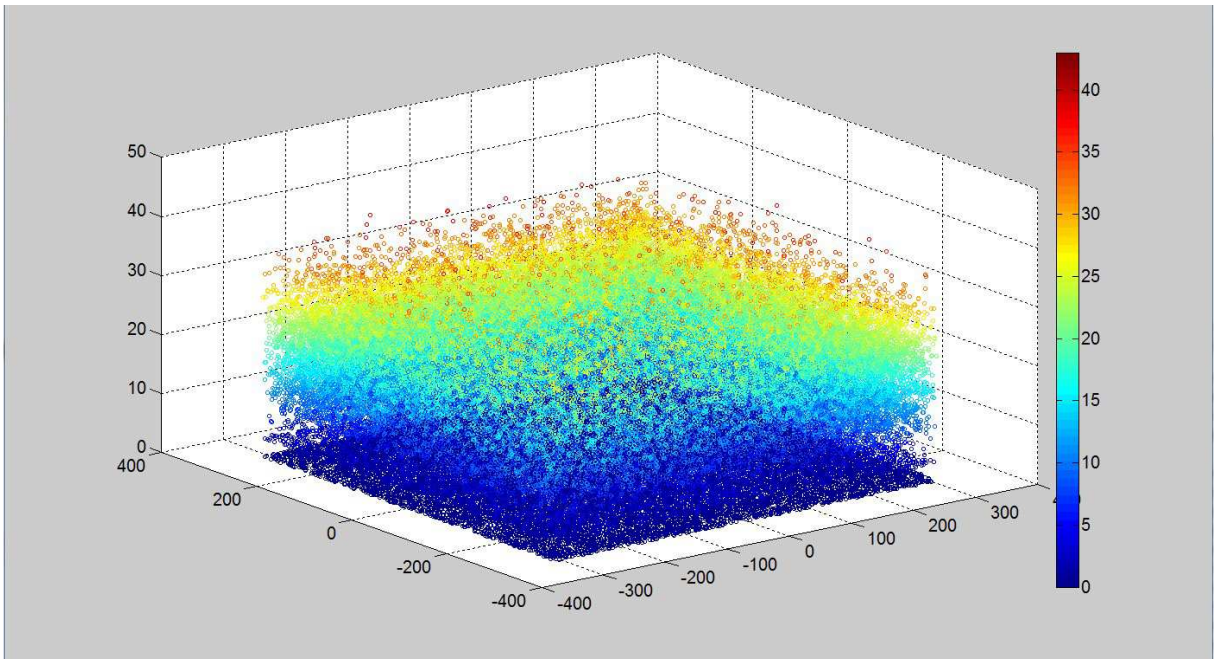

Figure 5.15. Operand-related error rate of FPU_ADD at voltage 0.8V

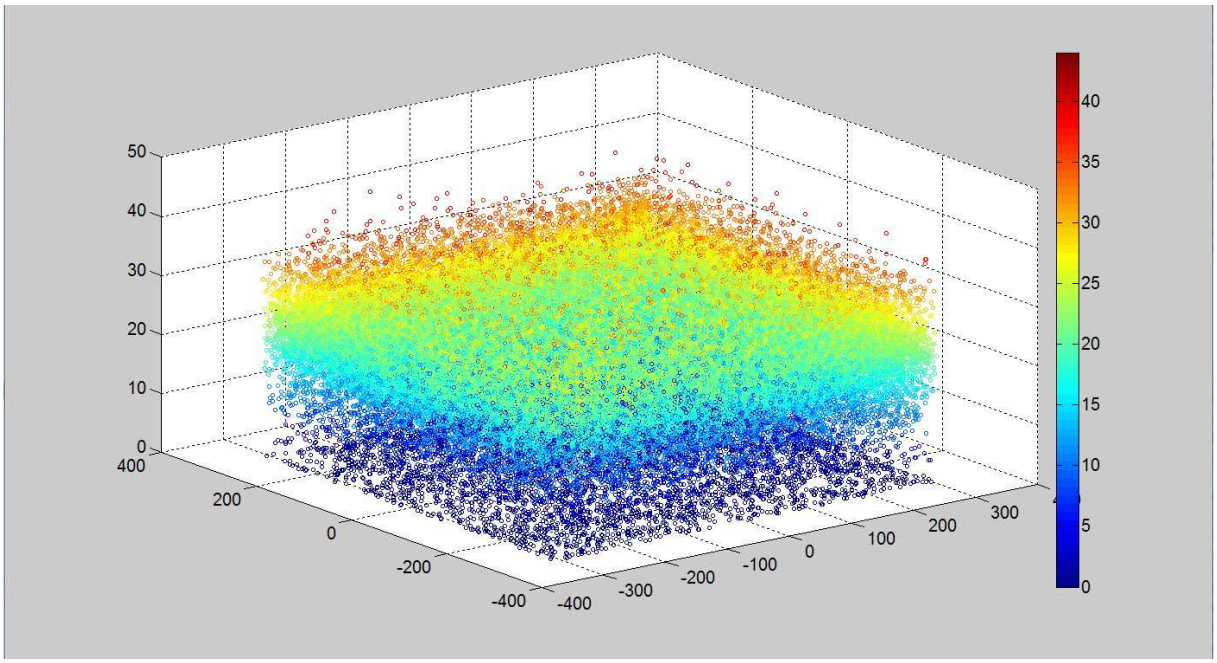

Figure 5.16. Operand-related error rate of FPU_ADD at voltage 0.7V

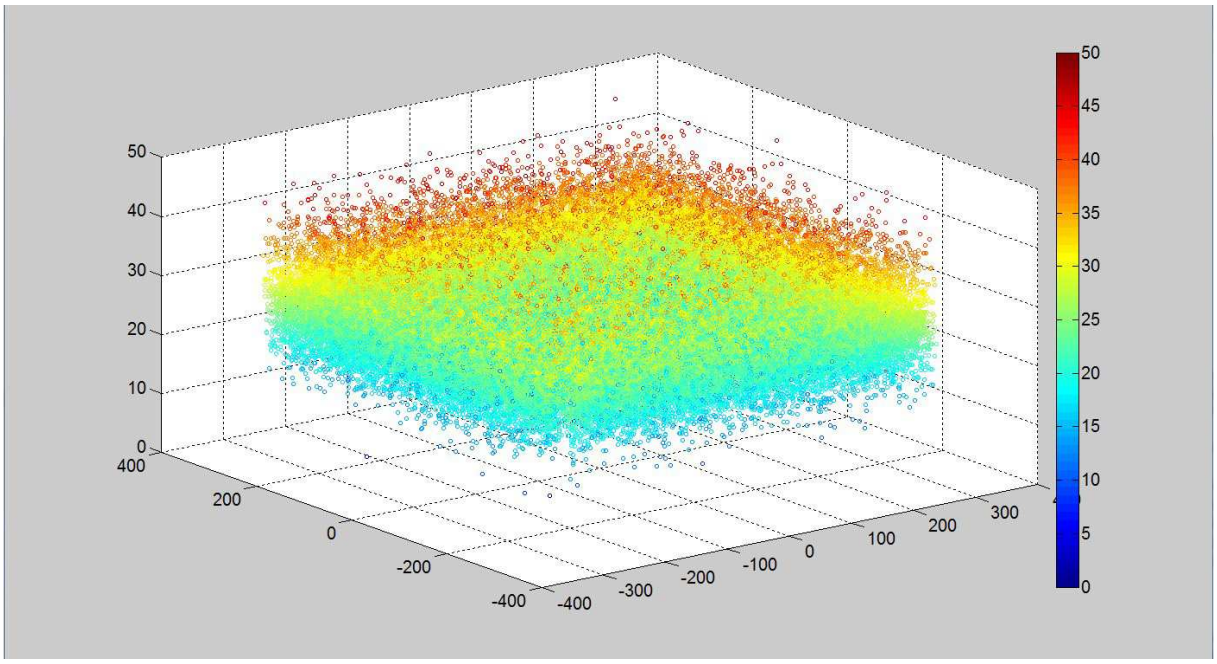

Figure 5.17. Operand-related error rate of FPU_ADD at voltage 0.6V

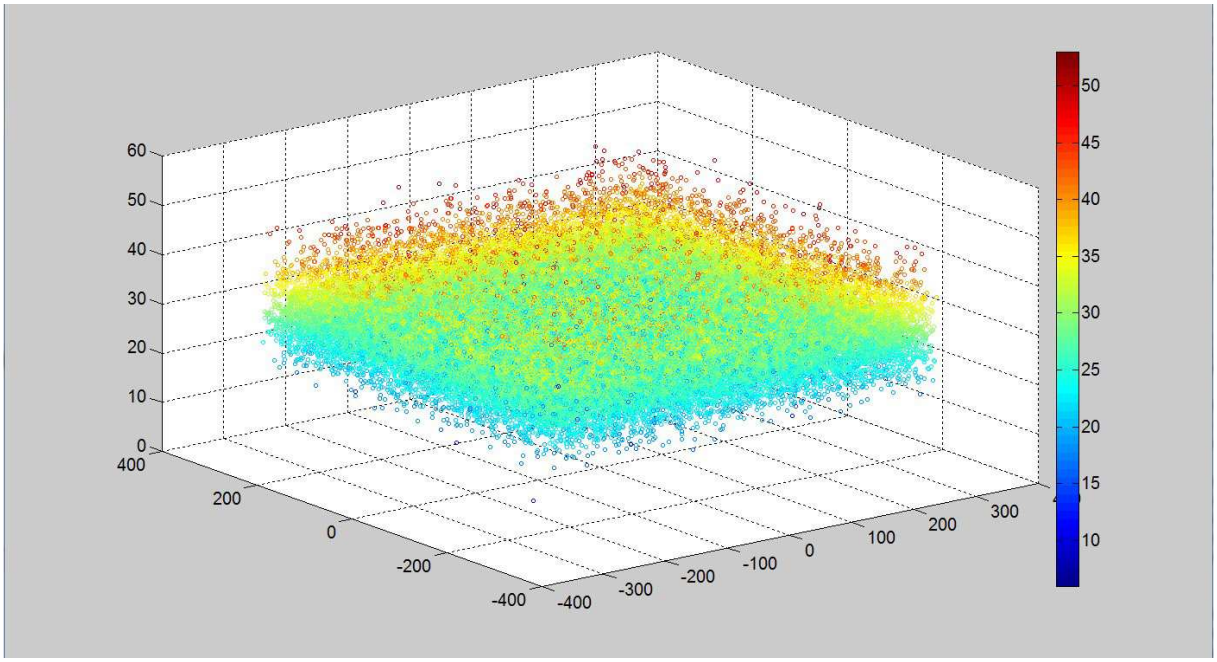

Figure 5.18. Operand-related error rate of FPU_ADD at voltage 0.5V

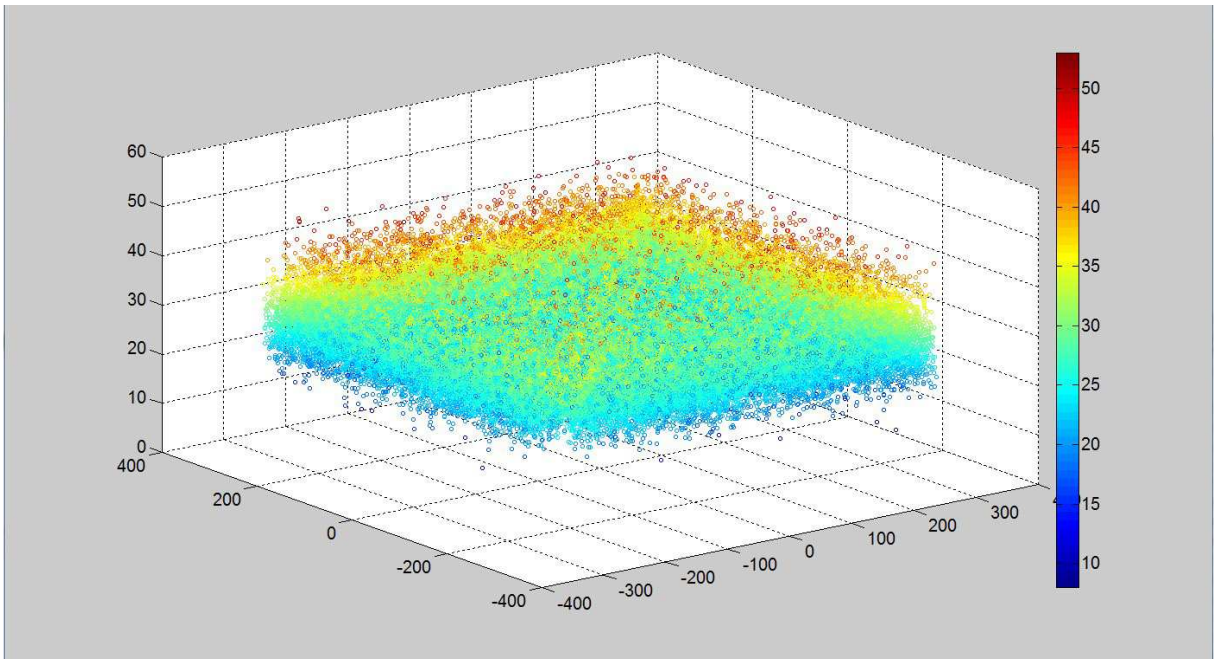

Figure 5.19. Operand-related error rate of FPU_ADD at voltage 0.4V

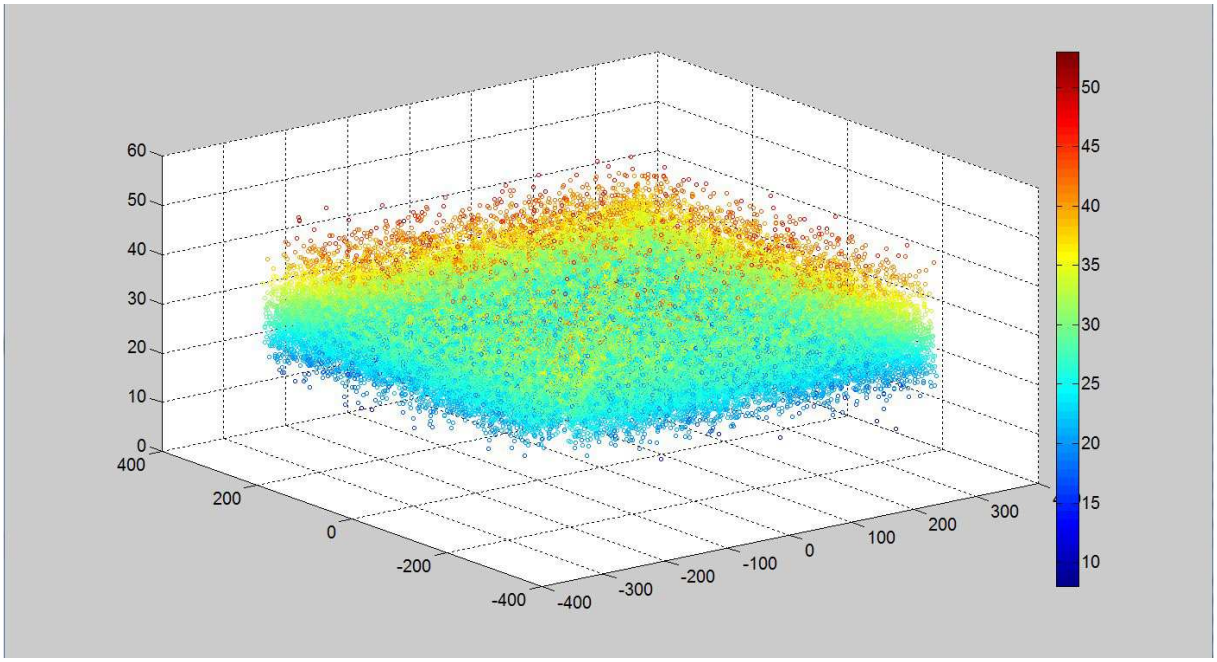

Figure 5.20. Operand-related error rate of FPU_ADD at voltage 0.3V

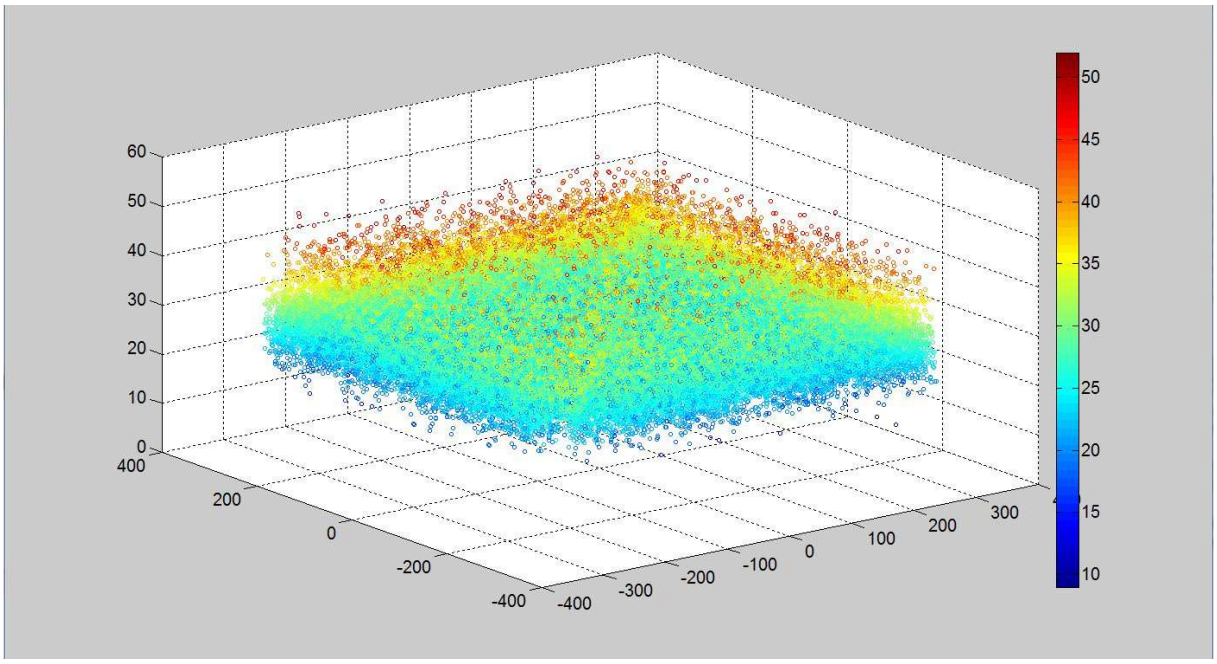

Figure 5.21. Operand-related error rate of FPU_ADD at voltage 0.2V

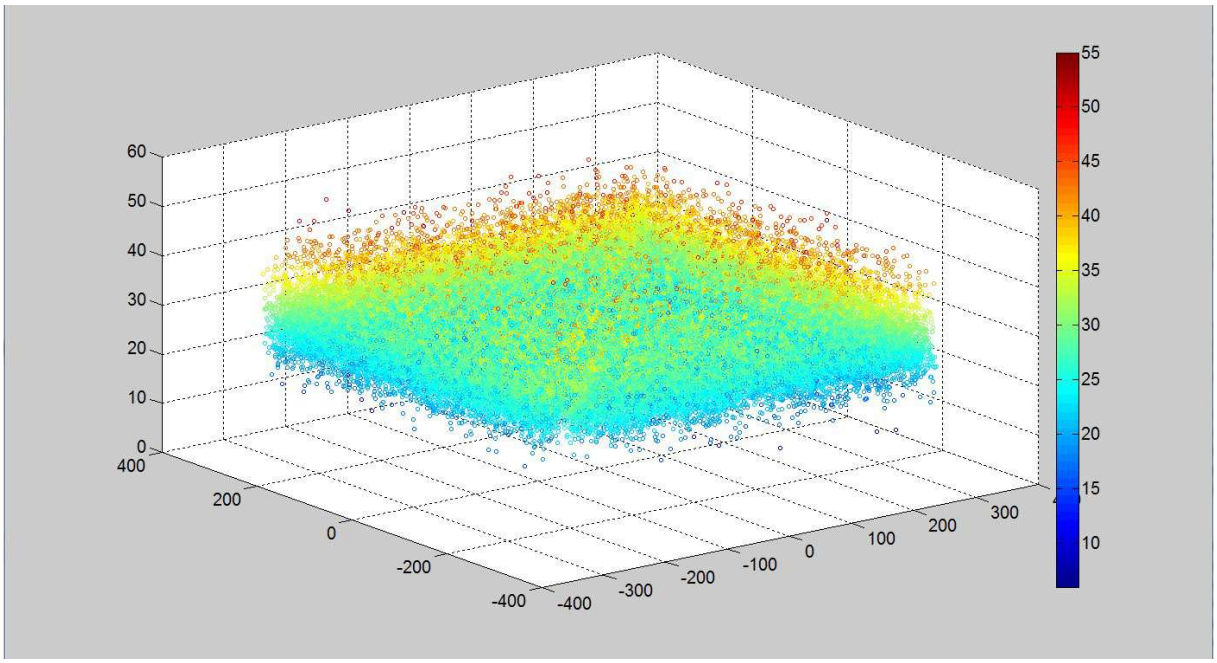

Figure 5.22. Operand-related error rate of FPU_ADD at voltage 0.1V

Let's look at voltage level 1.1V first. We can see most of the dots are lying on the ground. These are the correct results. There are two layers above. On the top layer, there is a gap along the diagonal. It indicates an opportunity for tolerating low voltages, when two operands have similar magnitude. This is in accordance with our understanding. When performing floating-point addition, it is necessary to align the magnitude of two operands, so they can be added correctly. If two operands are close in magnitude, it may take less time and the timing slack will be larger.

This gap also exists in voltage level of 1.0V to 0.7V. However, starting from 0.8V to lower voltages, the number of erroneous bits is generally high, we may not want to operate at these voltage levels. When voltage is below 0.6V, the number of wrong bits rises to approximately 30, which is consistent with 50% error rate we saw before.

Next we show the operand-related error rate of EXU_ALU in Figure 5.23 to Figure 5.33(the two axes are the two operands in linear space). We can see that, from voltage level 1.1V to 0.5V the results are the same. There is no any error until voltage level 0.4V, which is in accordance with the overall error rate presented before. At 0.4V, some minor timing errors start to appear, but most of the results are still correct. At 0.3V, more bits are erroneous, and at 0.2V and 0.1V, about half of the bits are flipped. Again, this indicates that lowering the supply voltage of EXU_ALU to 50% is safe for all input vectors we have tested.

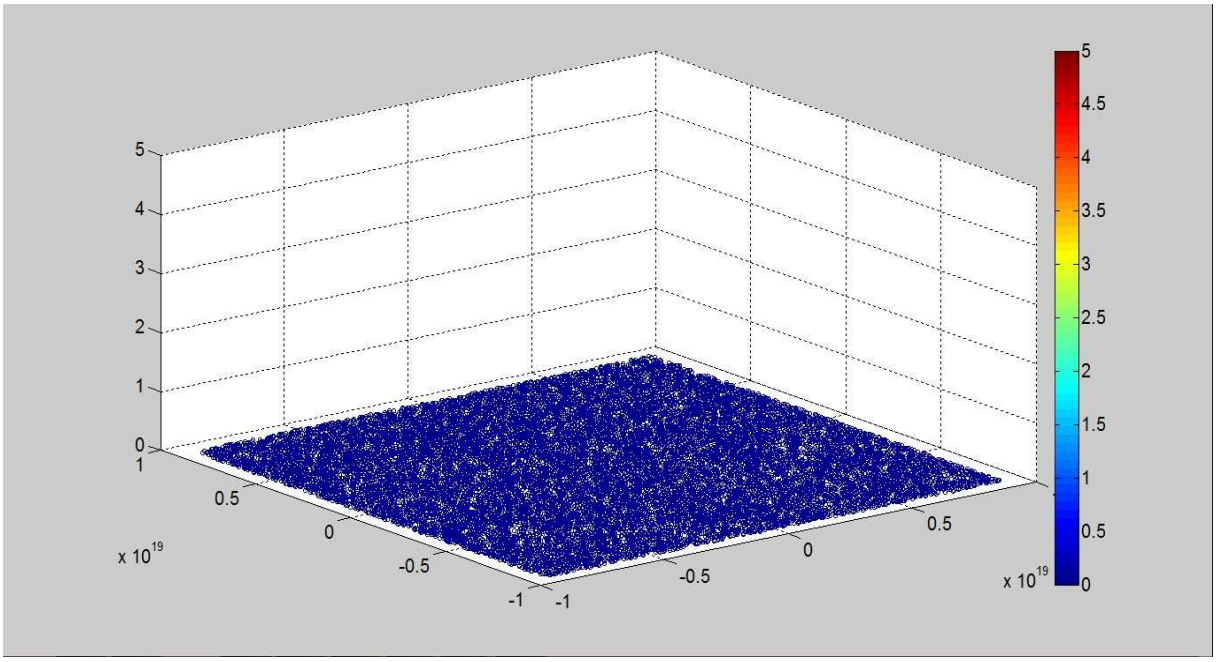

Figure 5.23. Operand-related error rate of EXU_ALU at voltage 1.1V

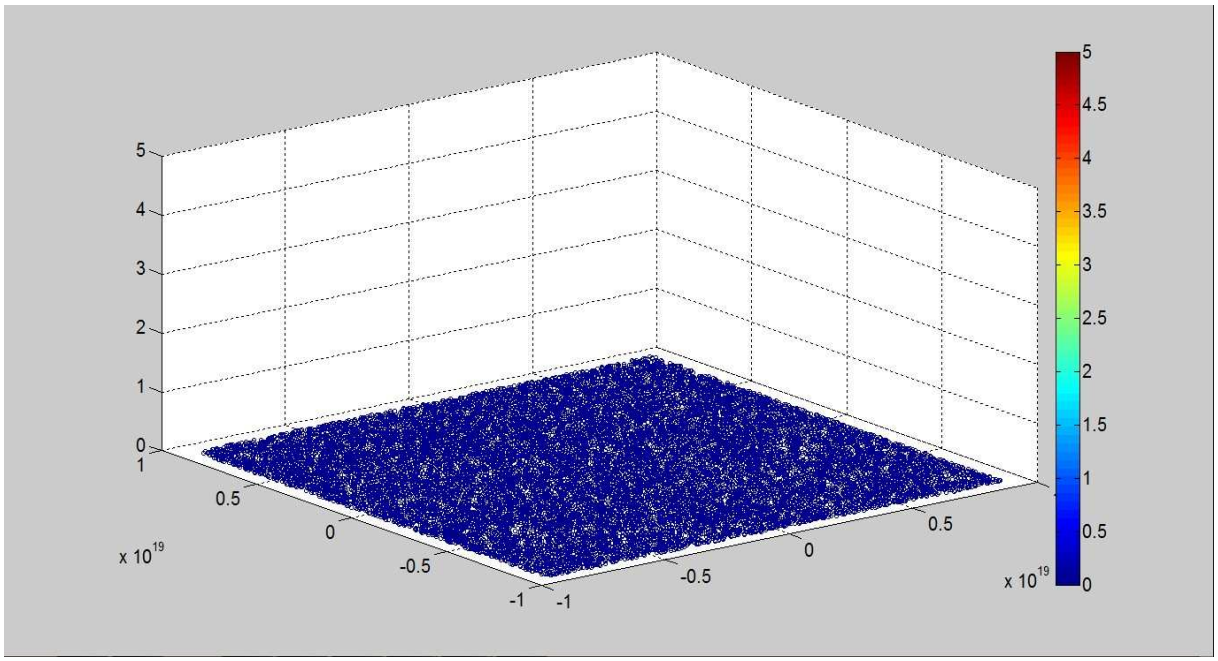

Figure 5.24. Operand-related error rate of EXU_ALU at voltage 1.0V

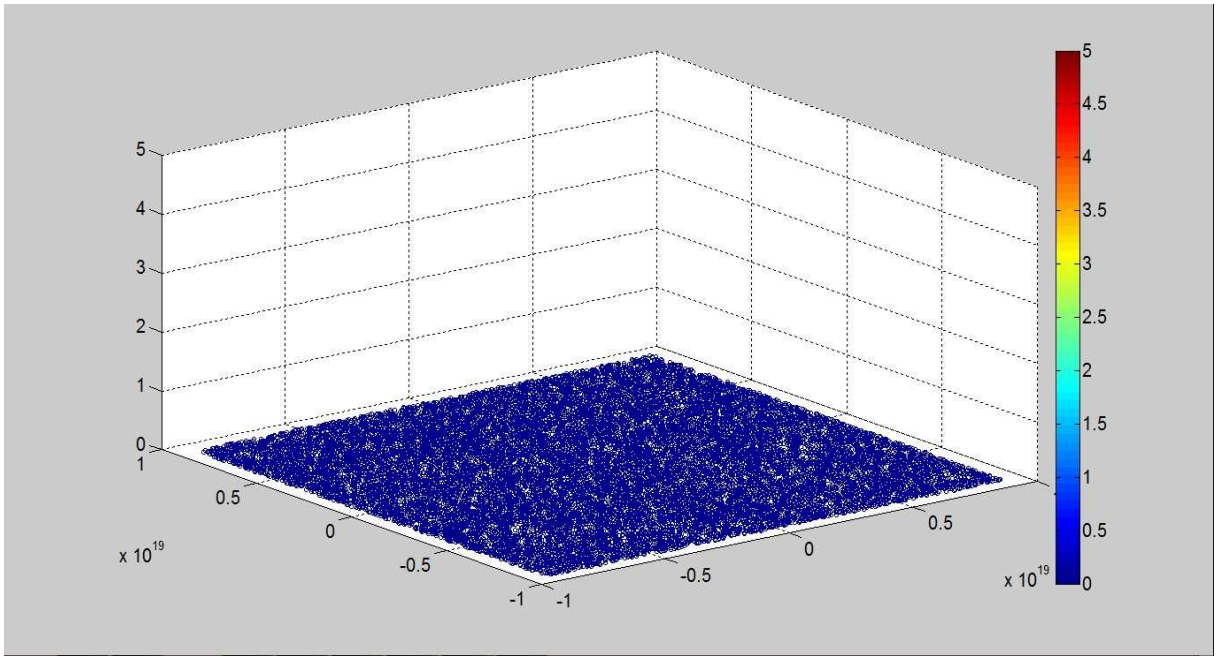

Figure 5.25. Operand-related error rate of EXU_ALU at voltage 0.9V

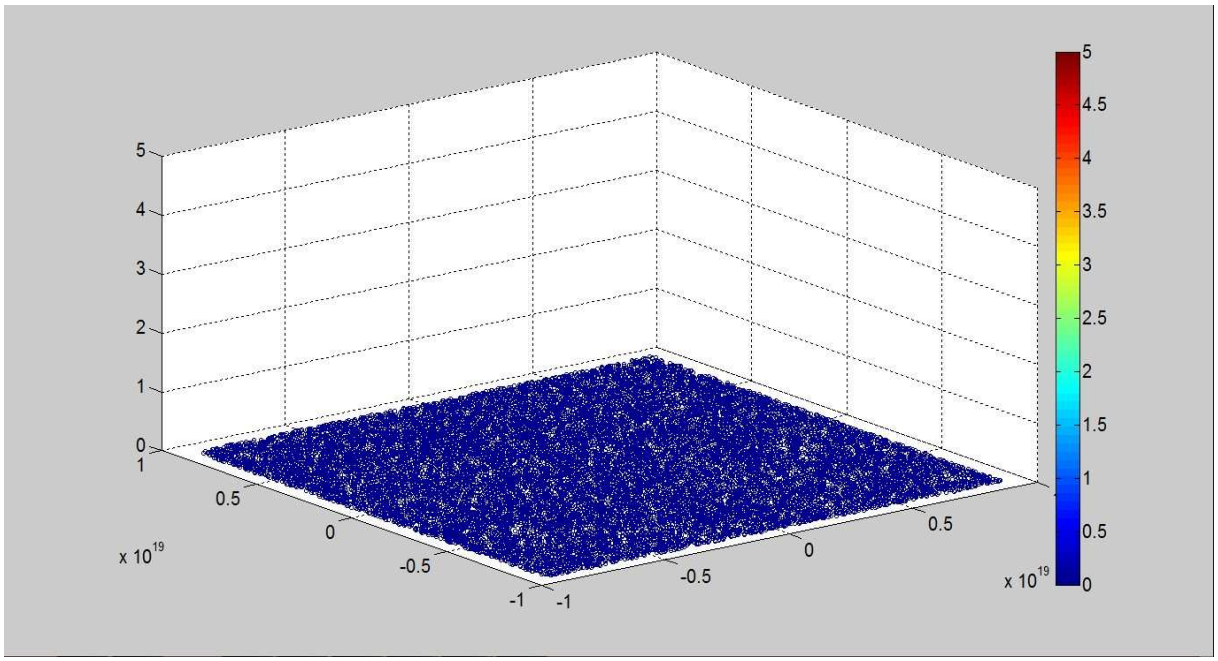

Figure 5.26. Operand-related error rate of EXU_ALU at voltage 0.8V

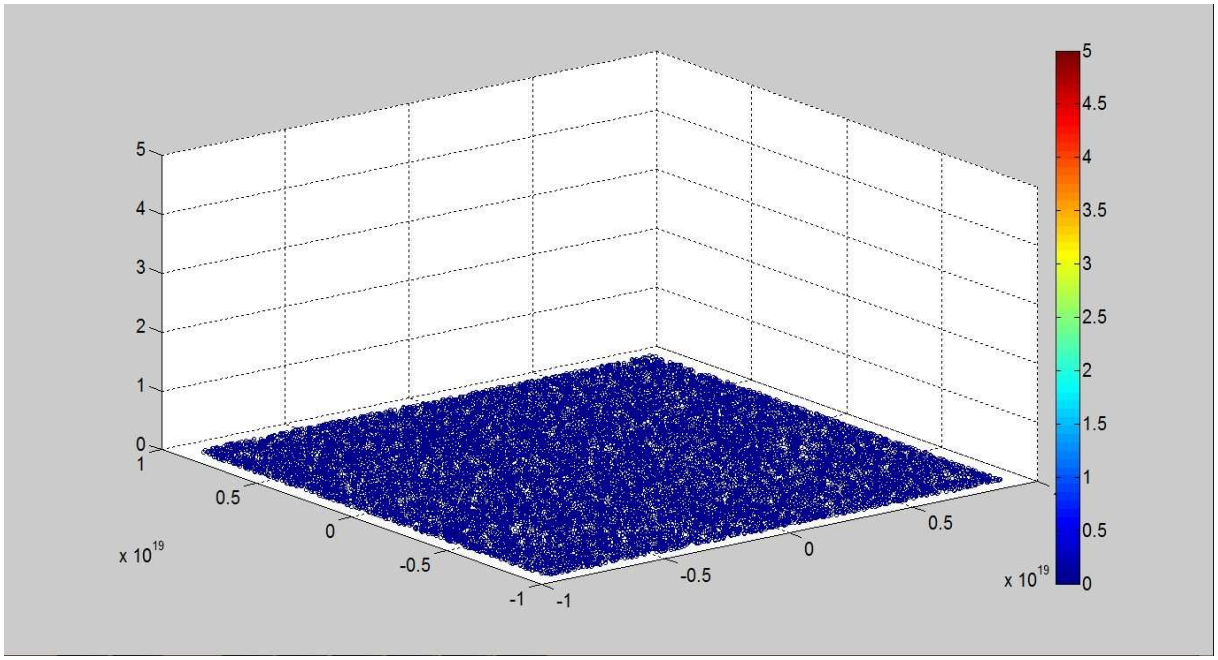

Figure 5.27. Operand-related error rate of EXU_ALU at voltage 0.7V

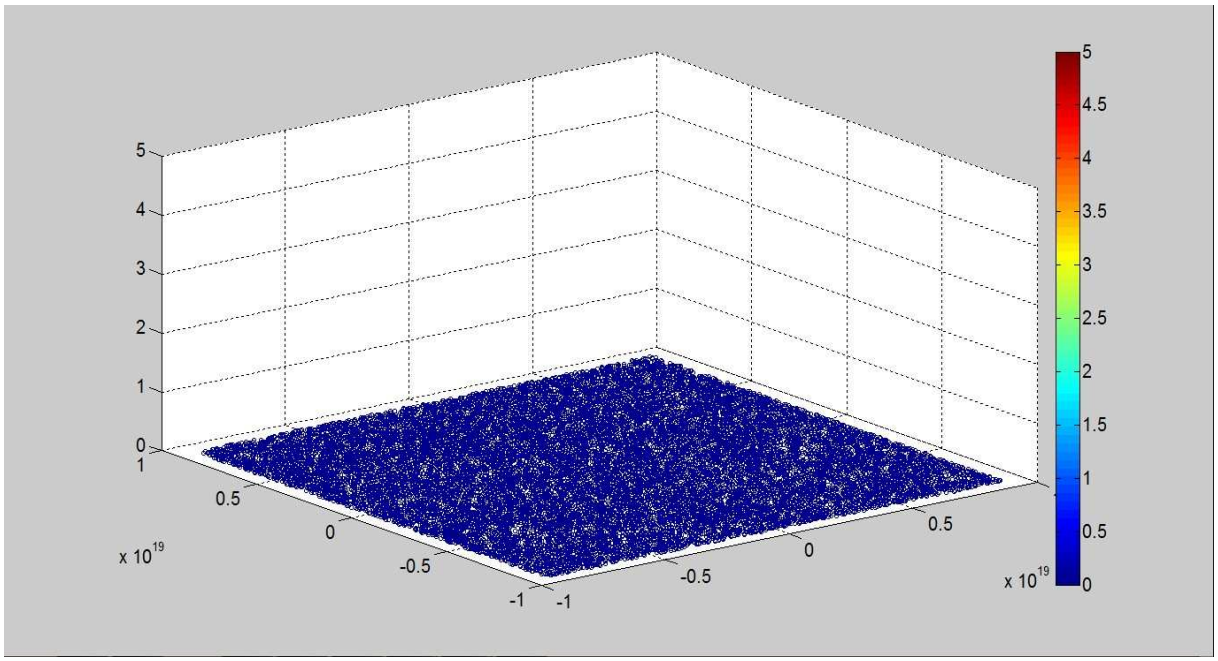

Figure 5.28. Operand-related error rate of EXU_ALU at voltage 0.6V

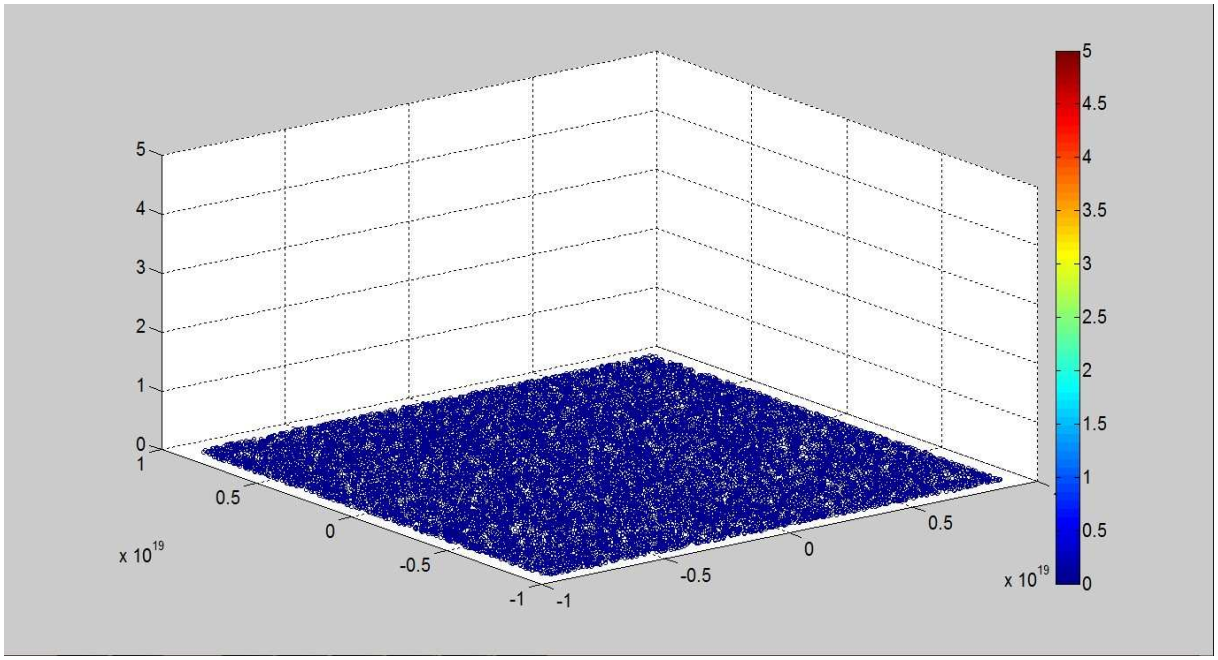

Figure 5.29. Operand-related error rate of EXU_ALU at voltage 0.5V

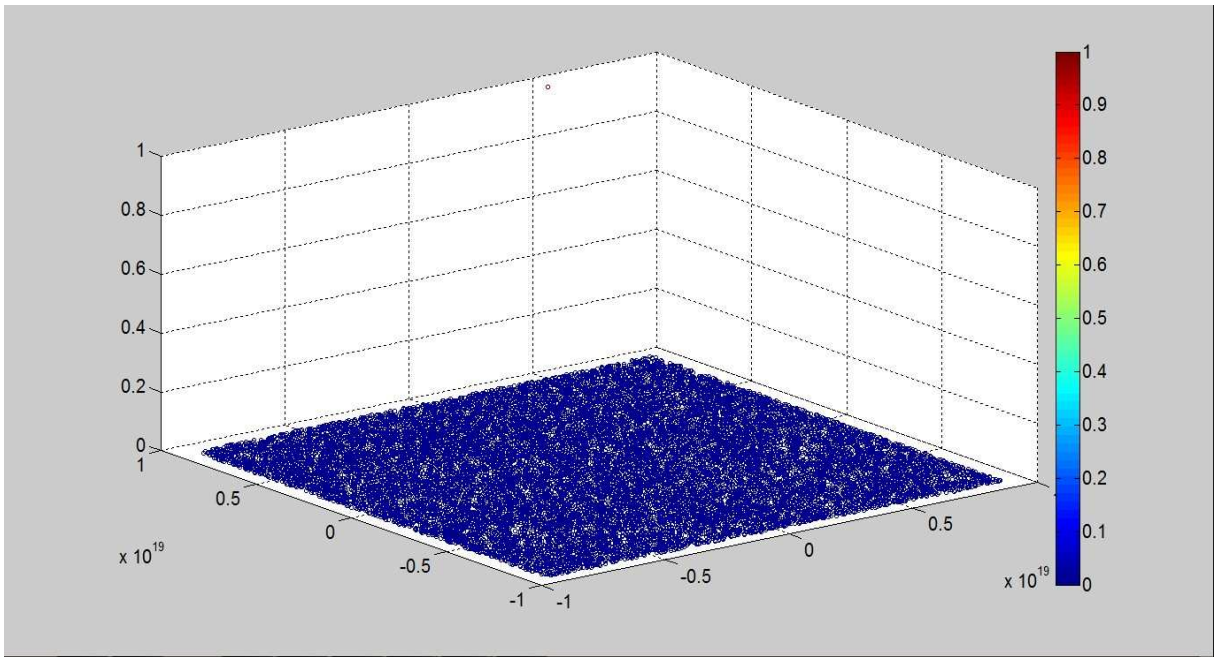

Figure 5.30. Operand-related error rate of EXU_ALU at voltage 0.4V

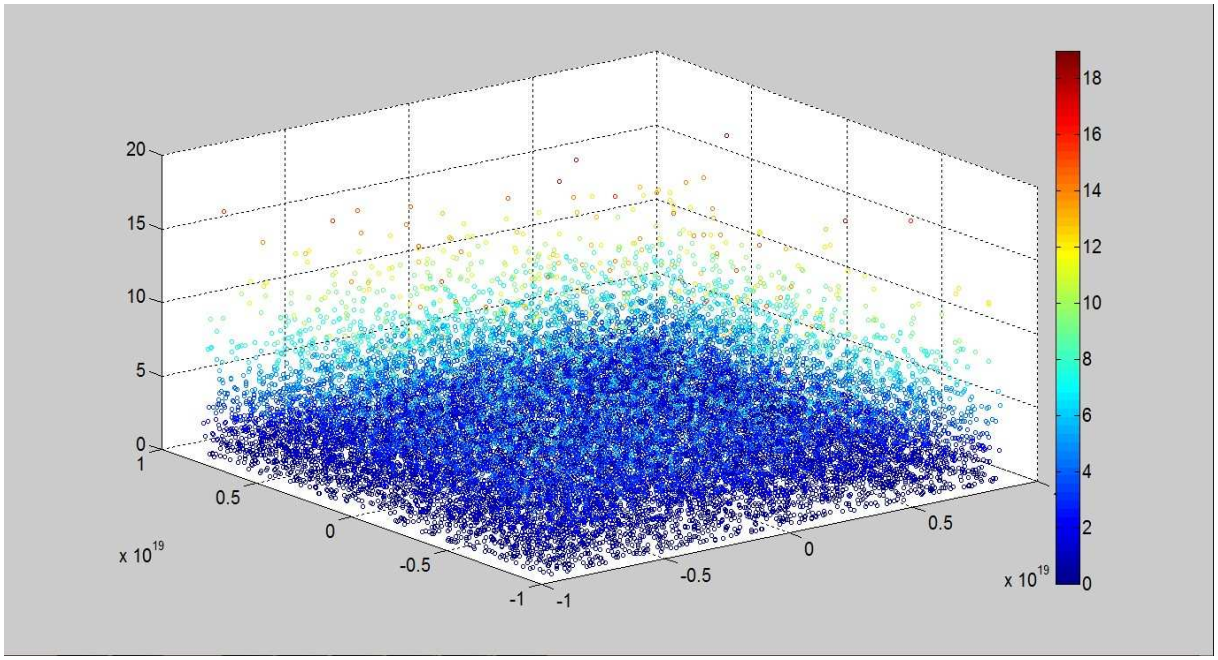

Figure 5.31. Operand-related error rate of EXU_ALU at voltage 0.3V

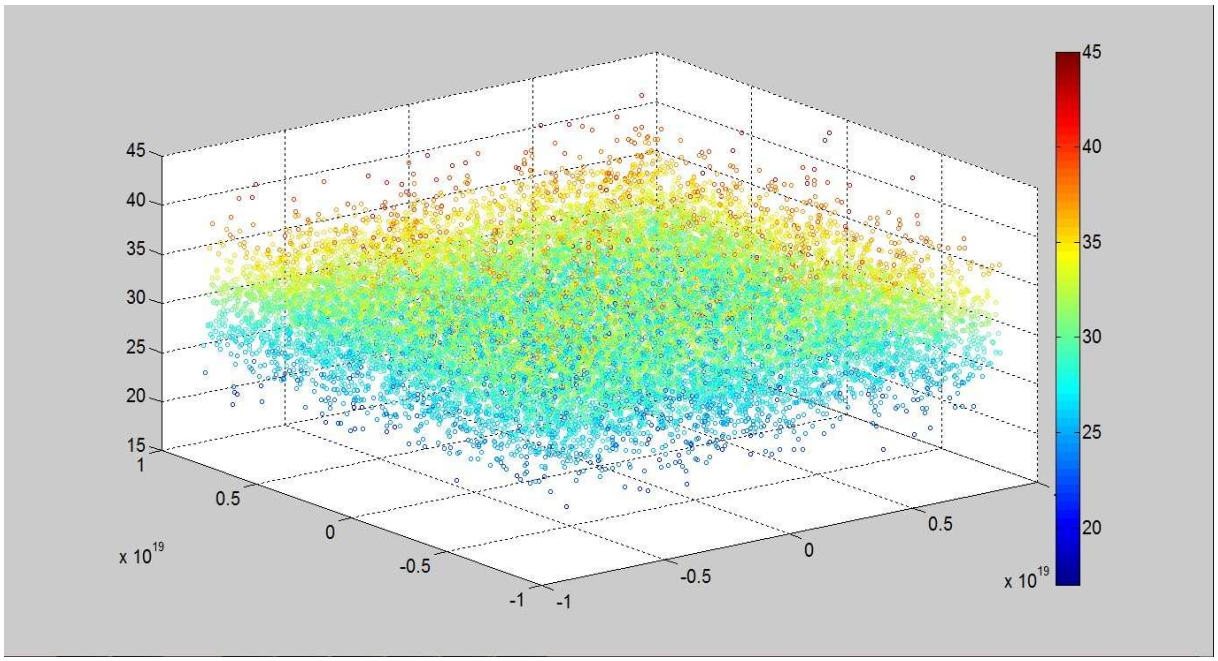

Figure 5.32. Operand-related error rate of EXU_ALU at voltage 0.2V

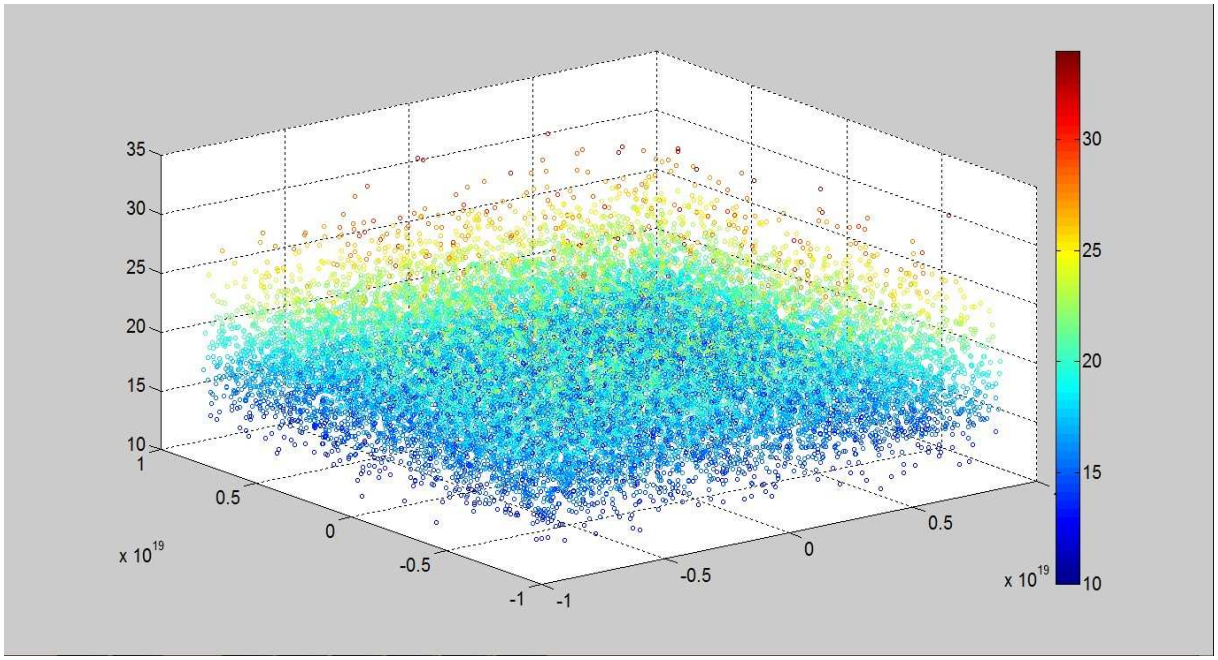

Figure 5.33. Operand-related error rate of EXU_ALU at voltage 0.1V

We show the operand-related error rate of FPU_MUL in Figure 5.34 to Figure 5.44. Since the input vectors we use to test FPU_MUL are much fewer than the ones used to test the other two execution units, we see the dots are sparse in the 3D space. It is difficult to draw reliable conclusion from these graphs, but we can validate that the number of erroneous bits increases with the decreasing of voltages. Notice that at voltage levels 0.4V and below, we can see some results have no erroneous bit, even thought the supply voltage is already smaller than the threshold voltage. This is not because that the circuit can still function. On the contrary, the circuit ceases to switch and the output stays at 0. If the correct result happens to be 0, we will observe a correct output.

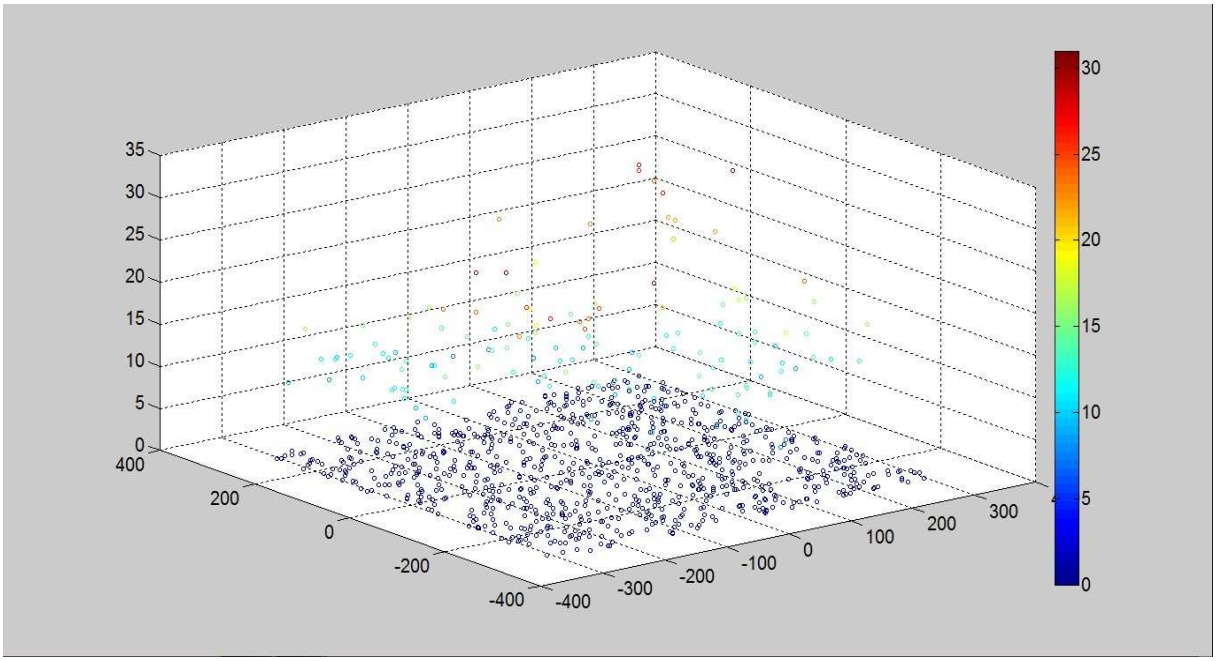

Figure 5.34. Operand-related error rate of FPU_MUL at voltage 1.1V

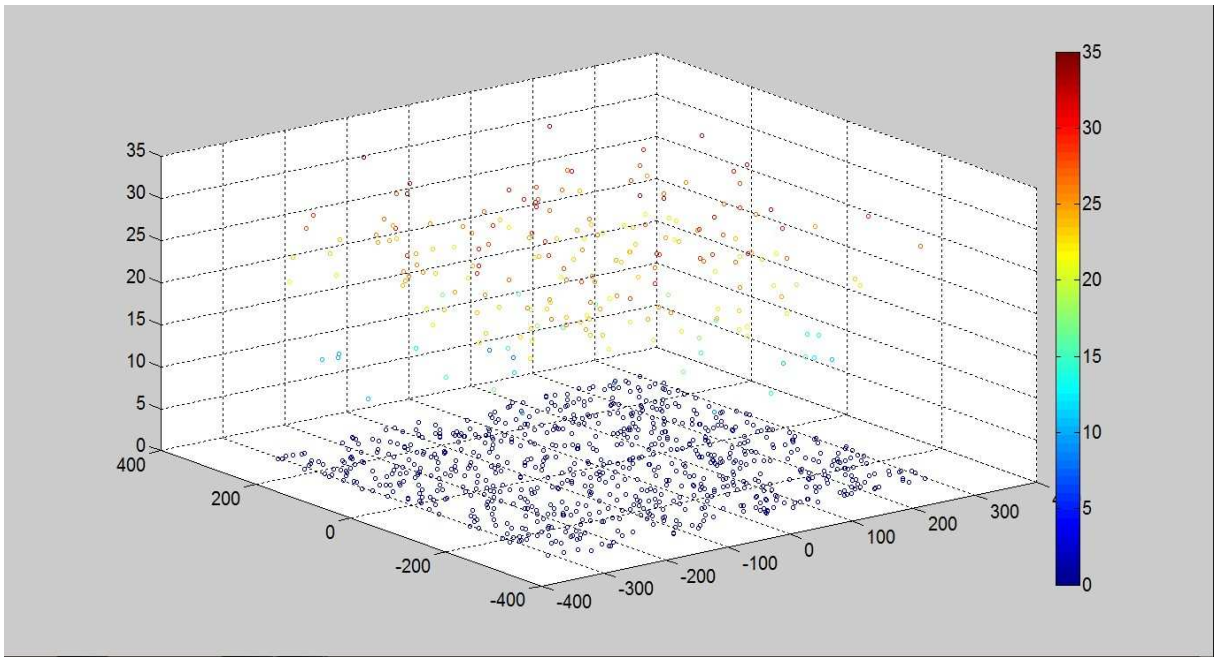

Figure 5.35. Operand-related error rate of FPU_MUL at voltage 1.0V

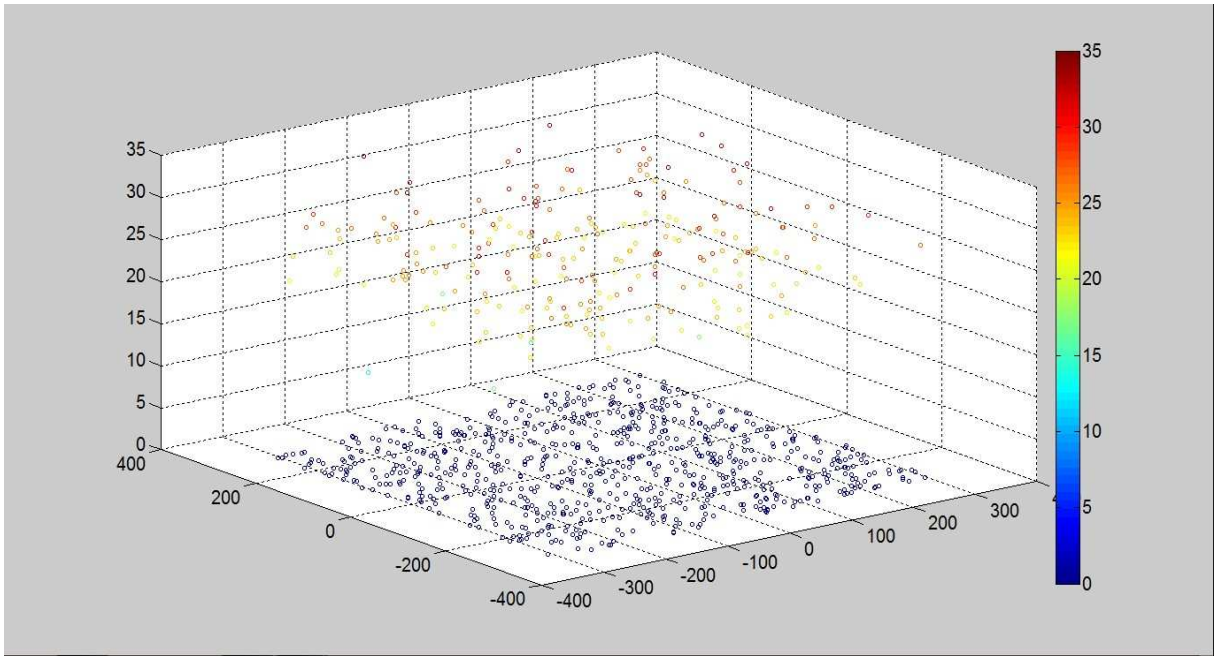

Figure 5.36. Operand-related error rate of FPU_MUL at voltage 0.9V

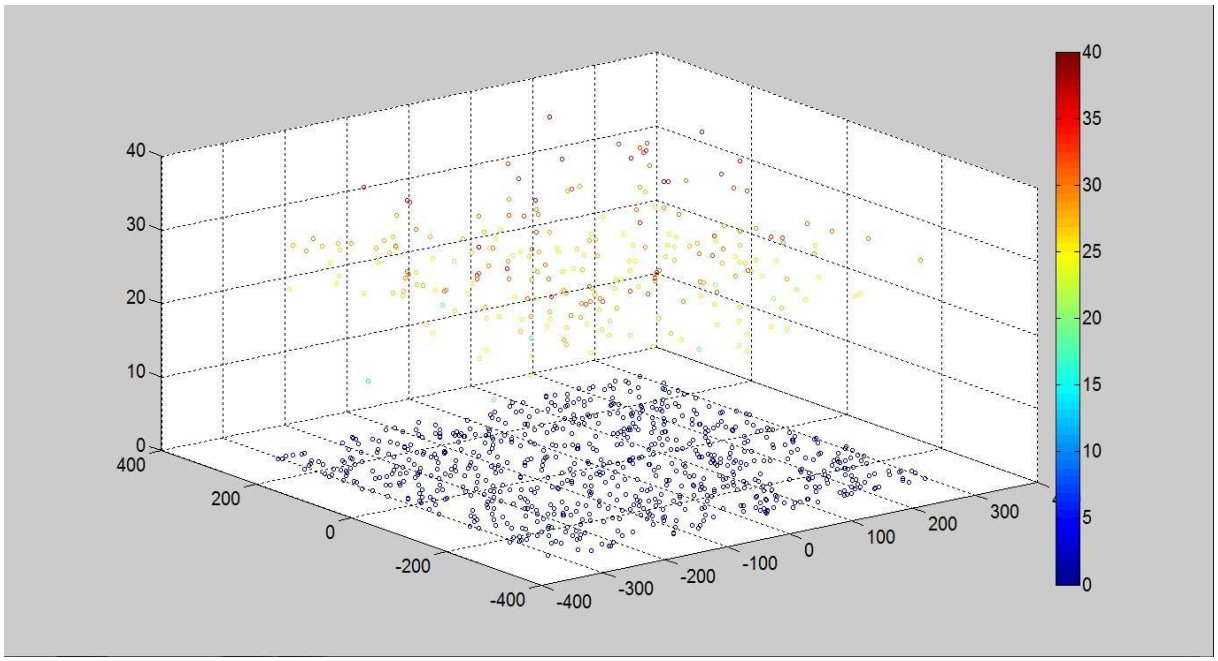

Figure 5.37. Operand-related error rate of FPU_MUL at voltage 0.8V

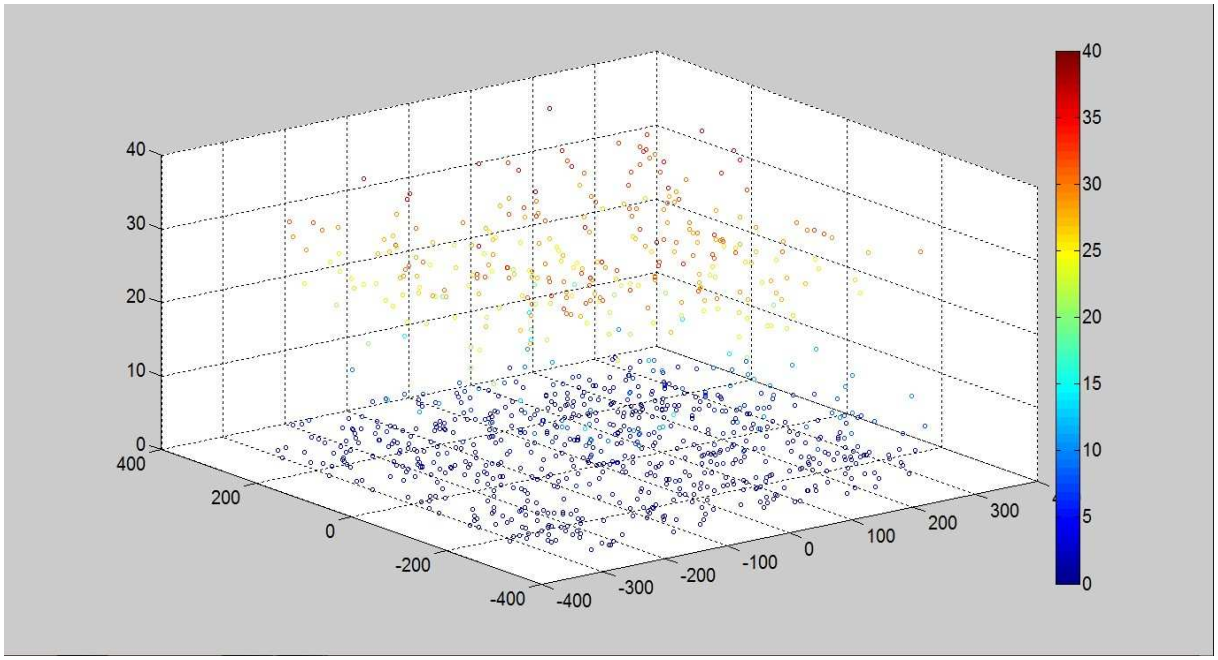

Figure 5.38. Operand-related error rate of FPU_MUL at voltage 0.7V

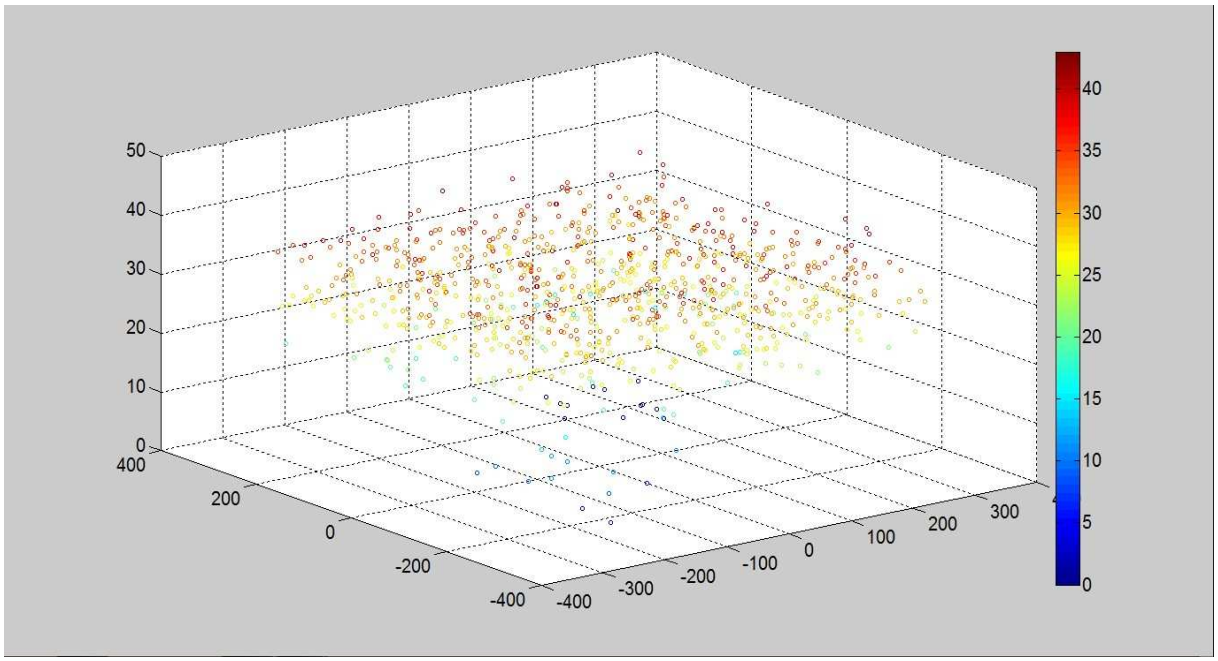

Figure 5.39. Operand-related error rate of FPU_MUL at voltage 0.6V

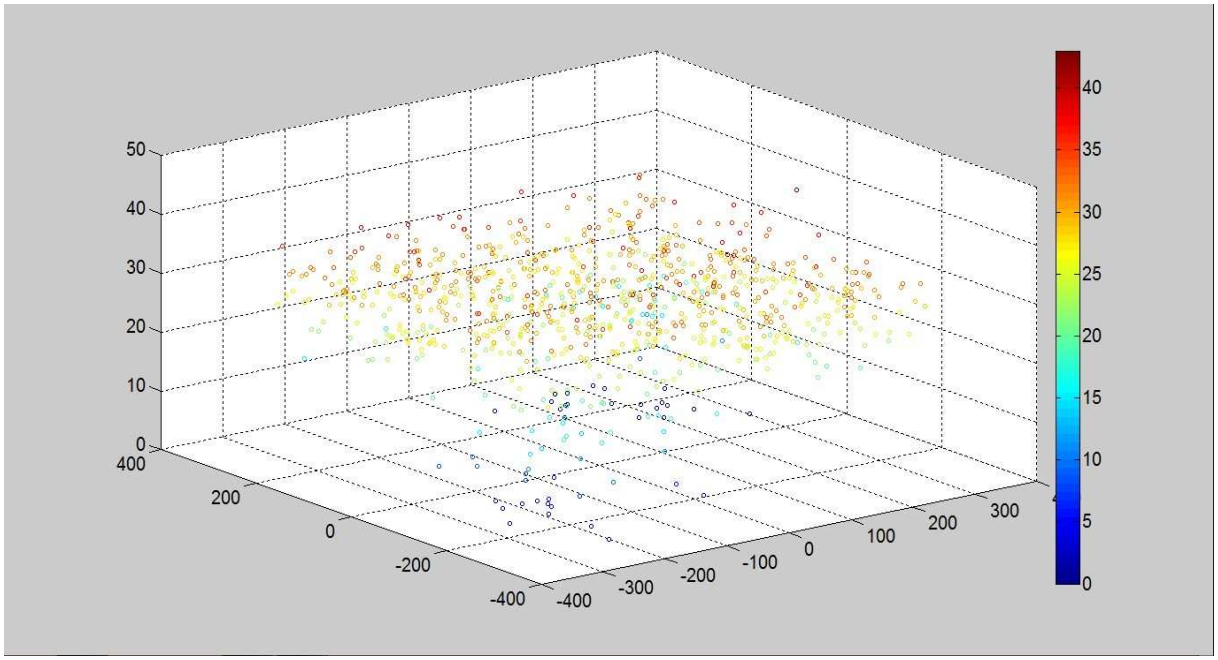

Figure 5.40. Operand-related error rate of FPU_MUL at voltage 0.5V

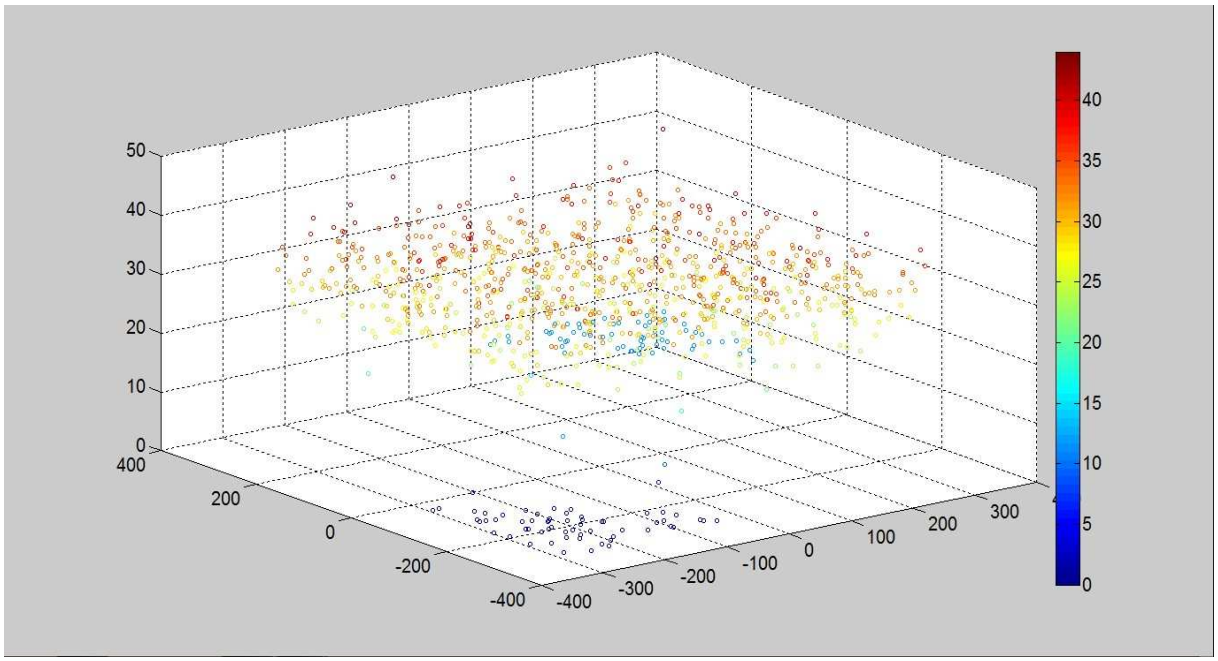

Figure 5.41. Operand-related error rate of FPU_MUL at voltage 0.4V

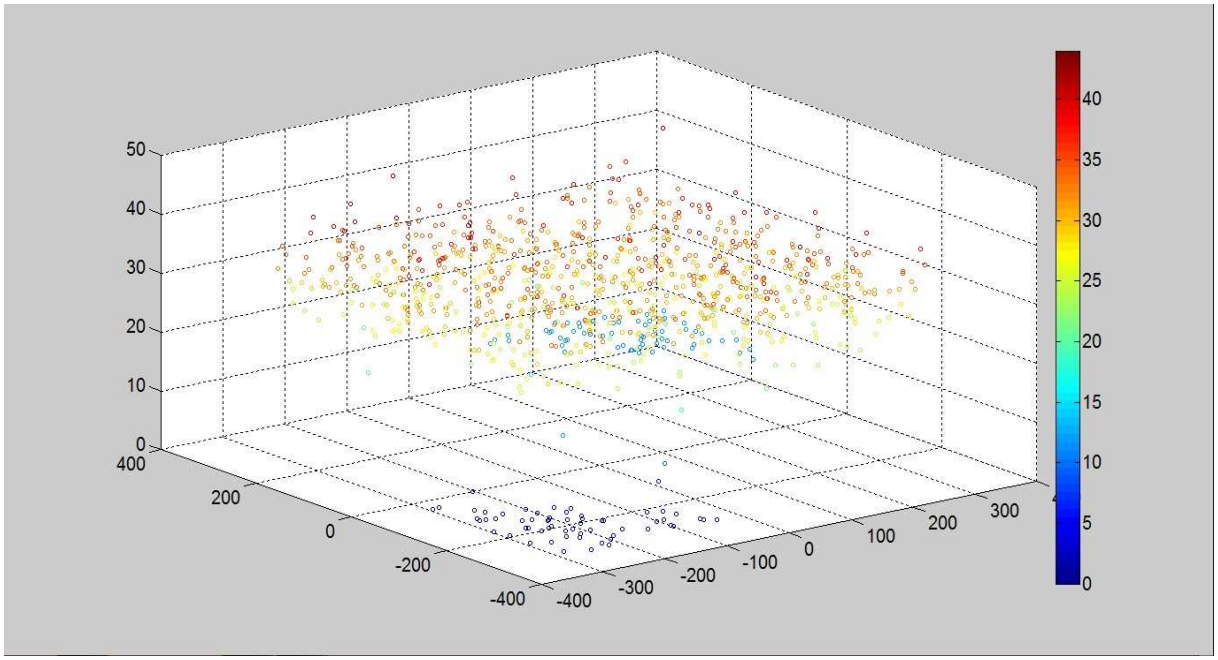

Figure 5.42. Operand-related error rate of FPU_MUL at voltage 0.3V

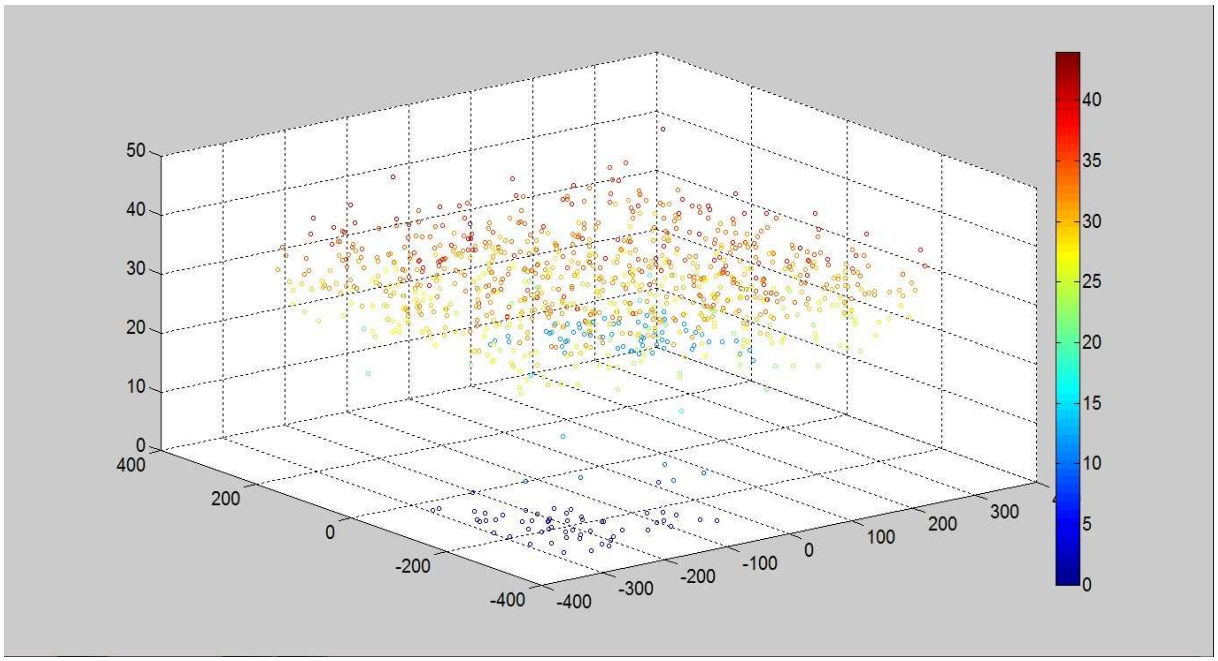

Figure 5.43. Operand-related error rate of FPU_MUL at voltage 0.2V

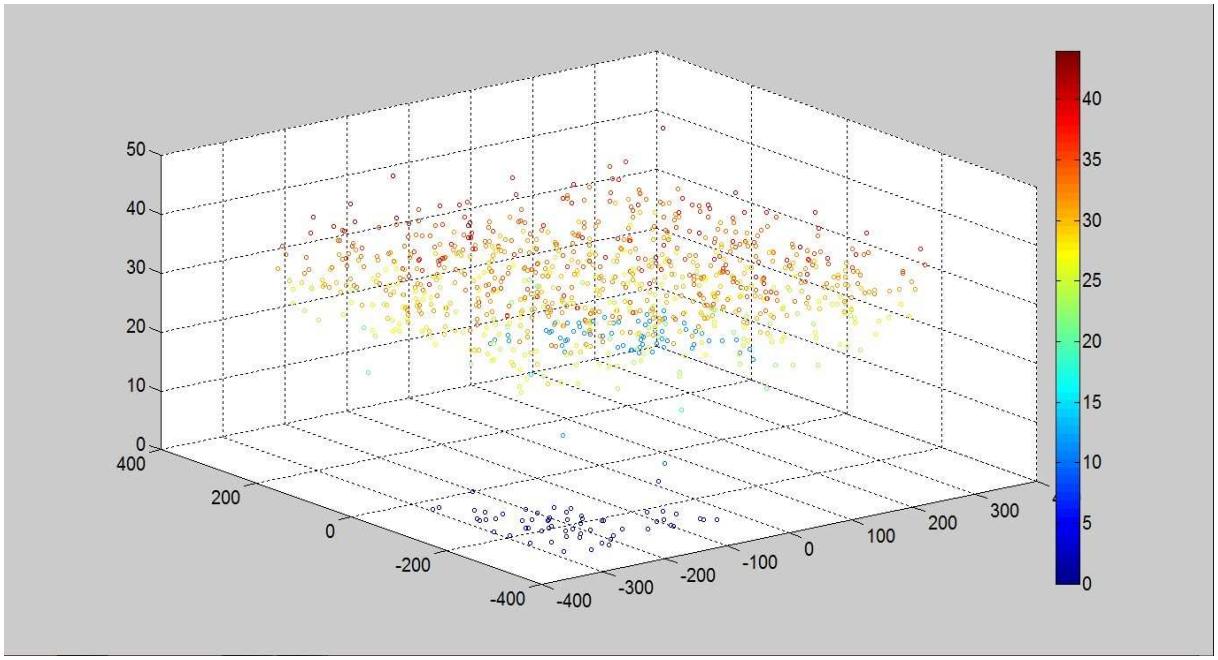

Figure 5.44. Operand-related error rate of FPU_MUL at voltage 0.1V

5.1.4. Relative Error

We define the fourth metric *Relative error* as:

$$\text{Relative error} = \left| \frac{\text{observed value} - \text{correct value}}{\text{correct value}} \right| \tag{5.1}$$

It shows how much the observed value is diverted from correct one. This indicates the actual impact on software. The results of FPU_ADD are shown in Figure 5.45 to Figure 5.51. From Figure 5.45 to Figure 5.49 show the relative error is in linear space, for voltage level between 1.1V and 0.7V. The relative error increases very quickly below voltage of 0.7V, so in Figure 5.50 to Figure 5.51 it is shown in logarithmic space, for voltage level 0.6V and 0.5V. Below 0.5V, the observed value is always Inf, and we omit figures for these voltage levels.

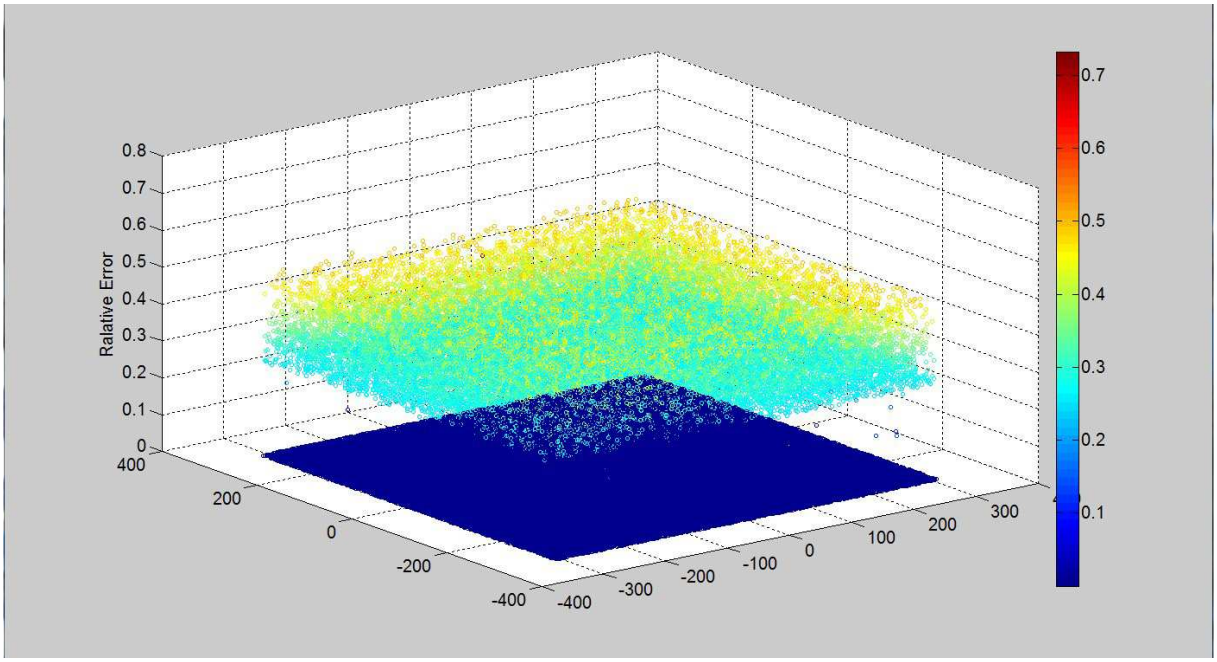

Figure 5.45. Relative error of FPU_ADD at voltage 1.1V in linear space

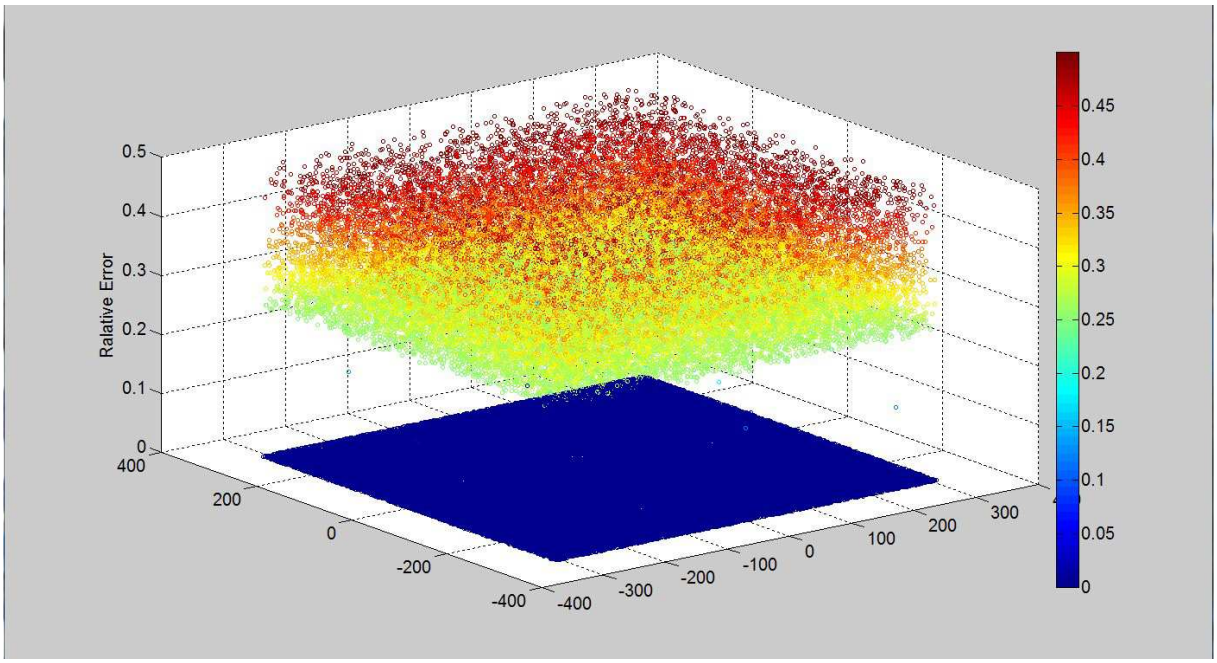

Figure 5.46. Relative error of FPU_ADD at voltage 1.0V in linear space

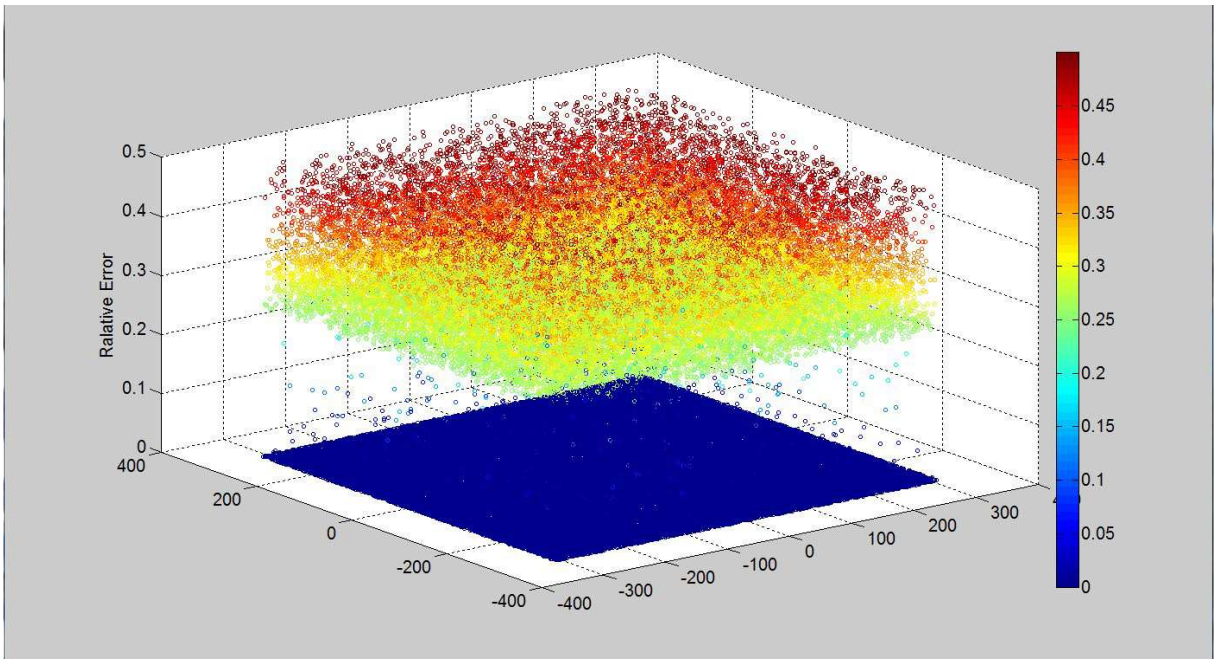

Figure 5.47. Relative error of FPU_ADD at voltage 0.9V in linear space

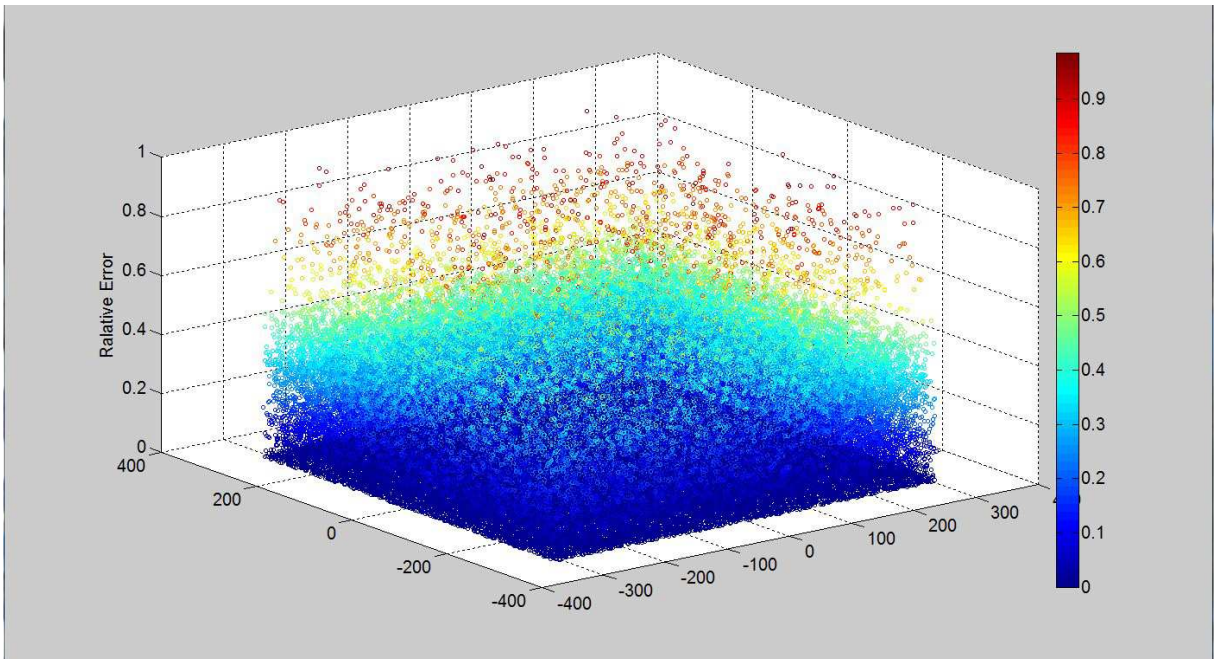

Figure 5.48. Relative error of FPU_ADD at voltage 0.8V in linear space

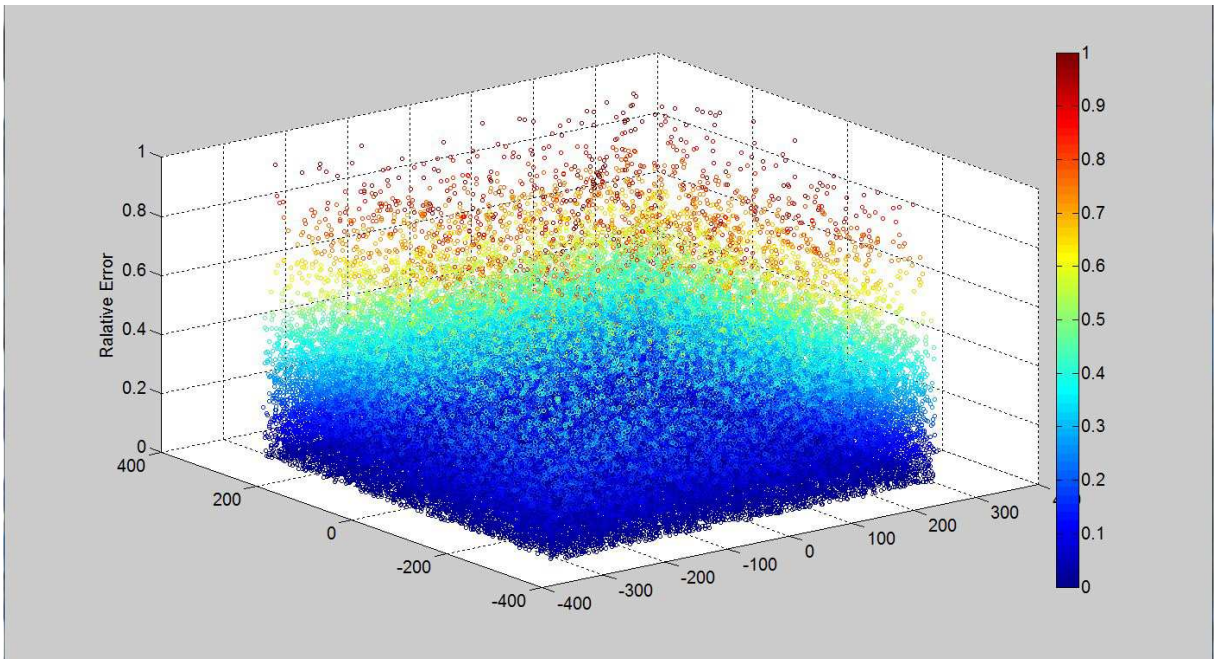

Figure 5.49. Relative error of FPU_ADD at voltage 0.7V in linear space

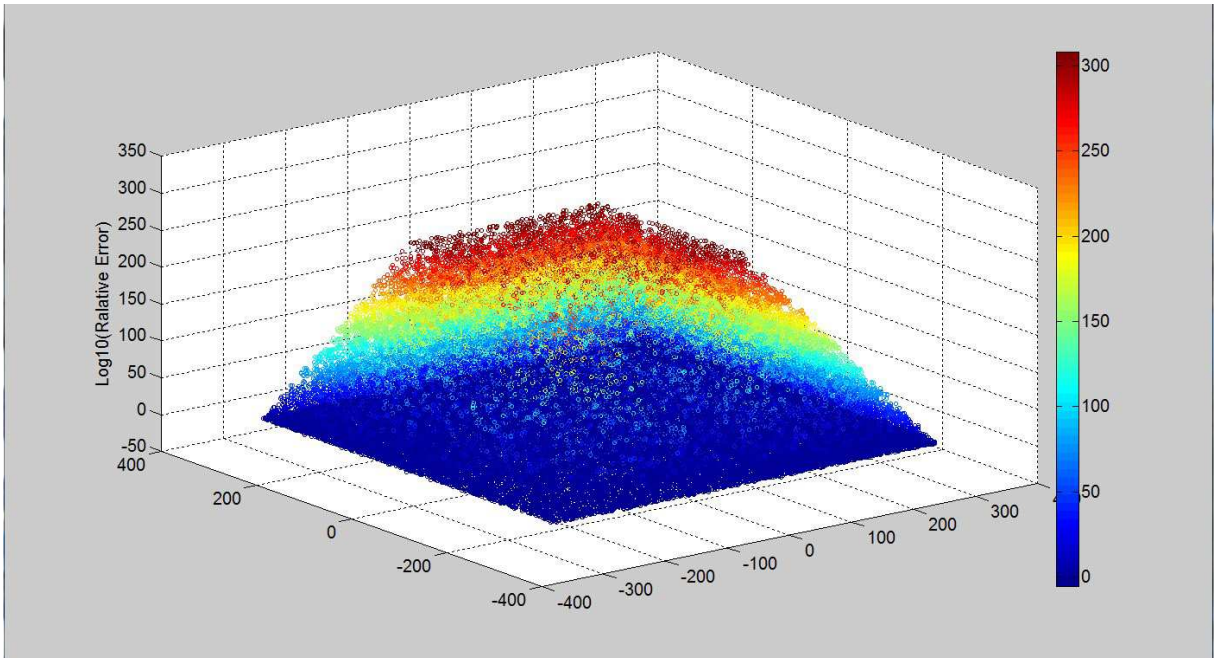

Figure 5.50. Relative error of FPU_ADD at voltage 0.6V in logarithmic space

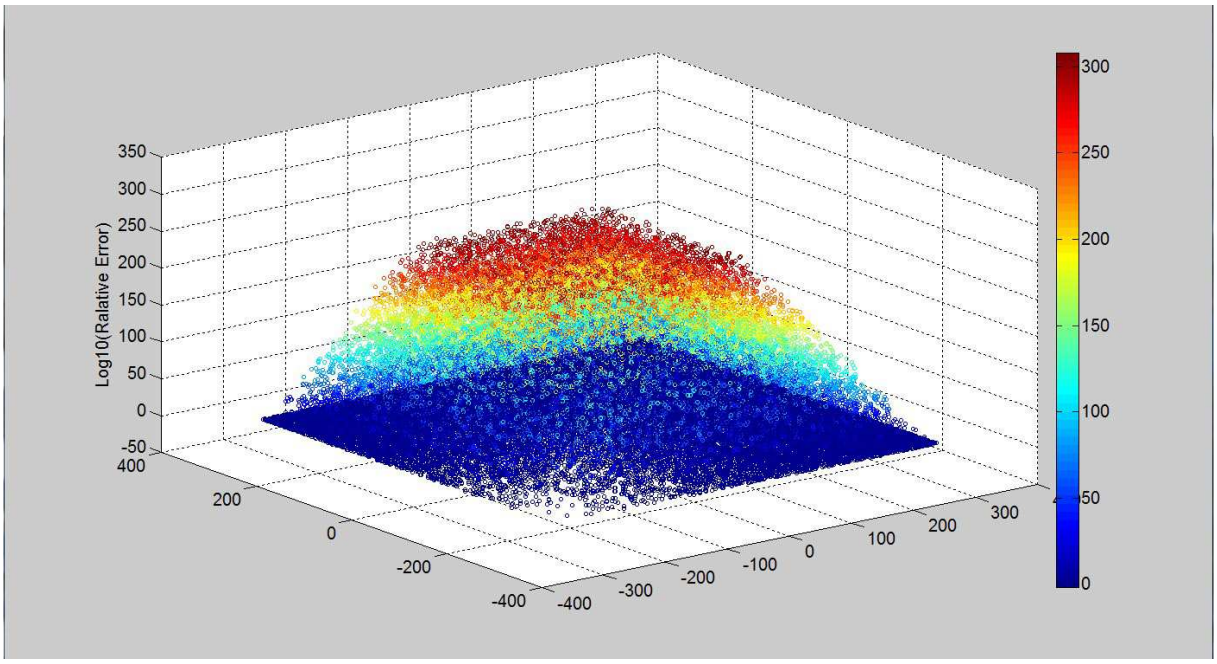

Figure 5.51. Relative error of FPU_ADD at voltage 0.5V in logarithmic space

The results show that voltage 0.7V is an important threshold. Voltage levels above it will have relative errors smaller than 1. However, when voltage goes below it, we will soon have massive errors of a magnitude which can only be measured in logarithmic space. Such massive error will surely bring down the output quality.

We show the relative error of EXU_ALU in Figure 5.52 to Figure 5.62. For voltage levels 0.5V and above, the result is always correct. At voltage levels 0.4V and below, there are some timing errors. The maximum of them can be as large as 10^{14}. However, such massive error is very rare, and most of the results are still correct or only have minor deviation.

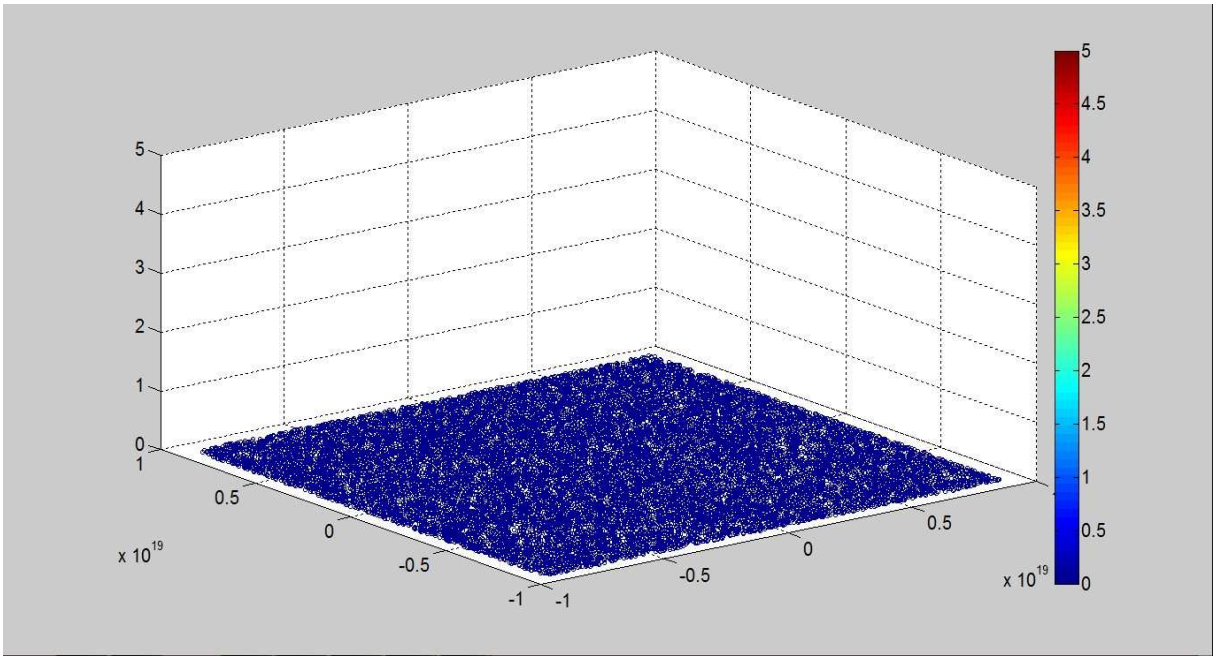

Figure 5.52. Relative error of EXU_ALU at voltage 1.1V in linear space

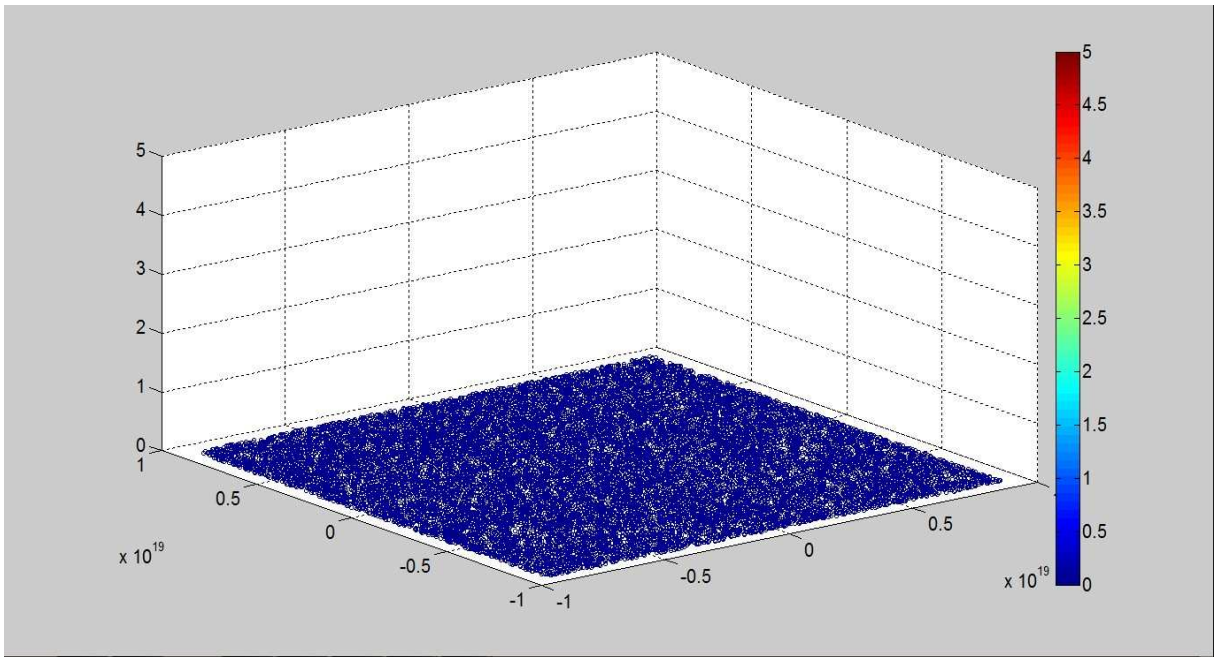

Figure 5.53. Relative error of EXU_ALU at voltage 1.0V in linear space

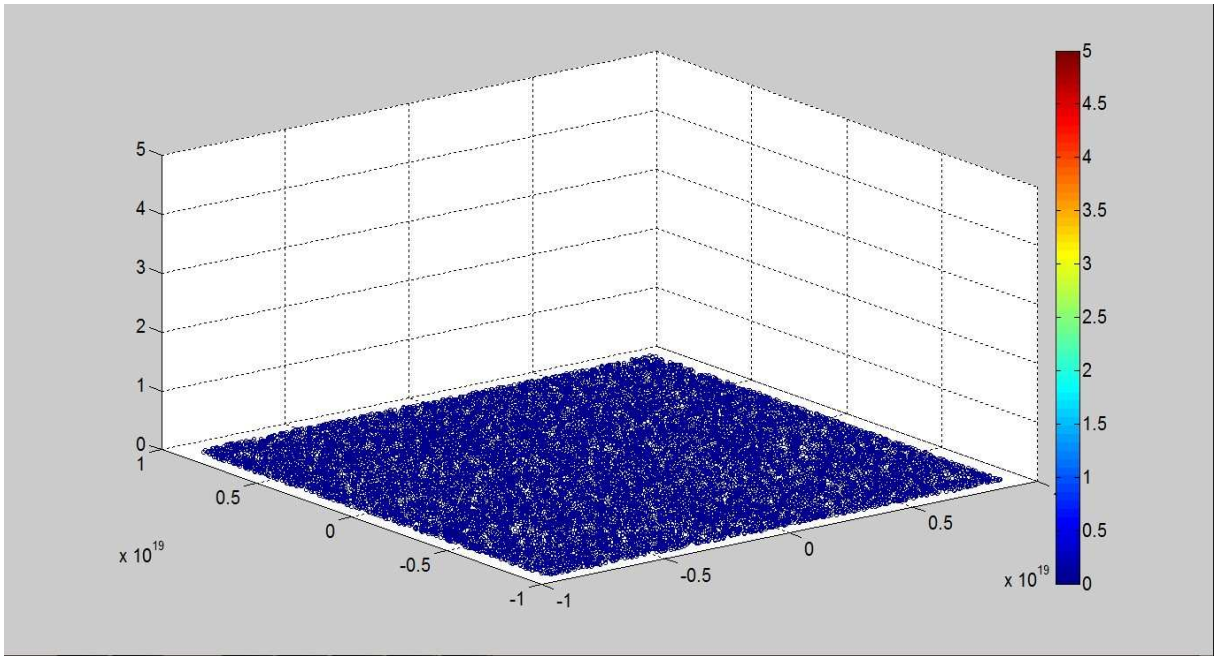

Figure 5.54. Relative error of EXU_ALU at voltage 0.9V in linear space

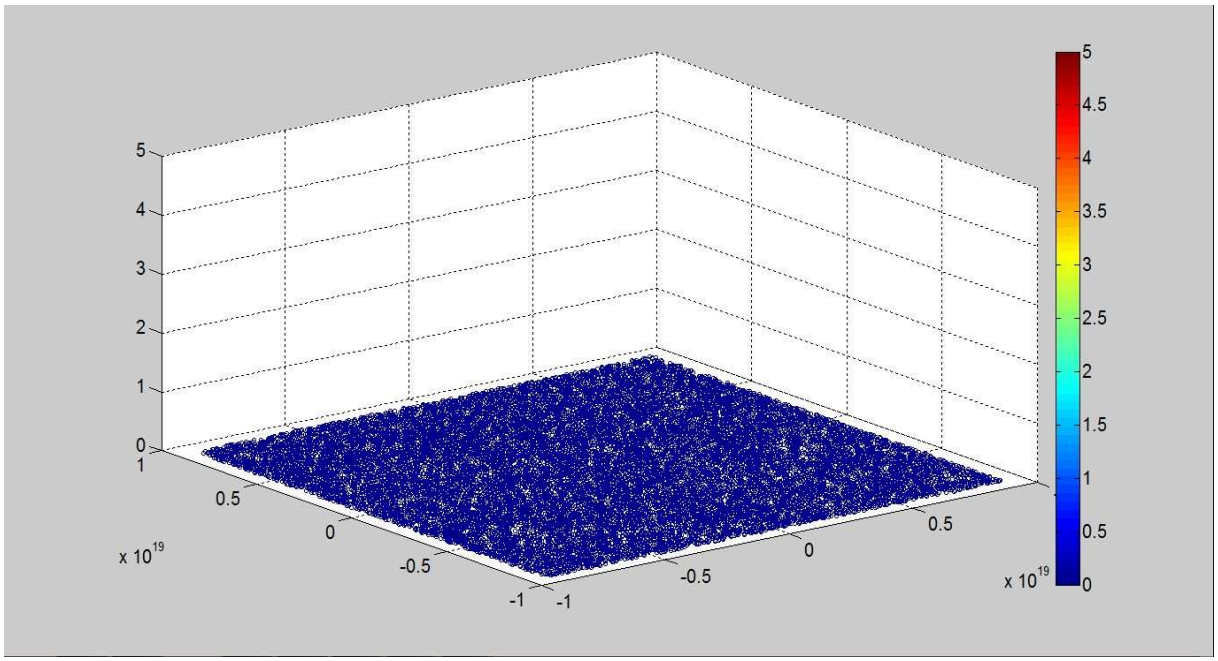

Figure 5.55. Relative error of EXU_ALU at voltage 0.8V in linear space

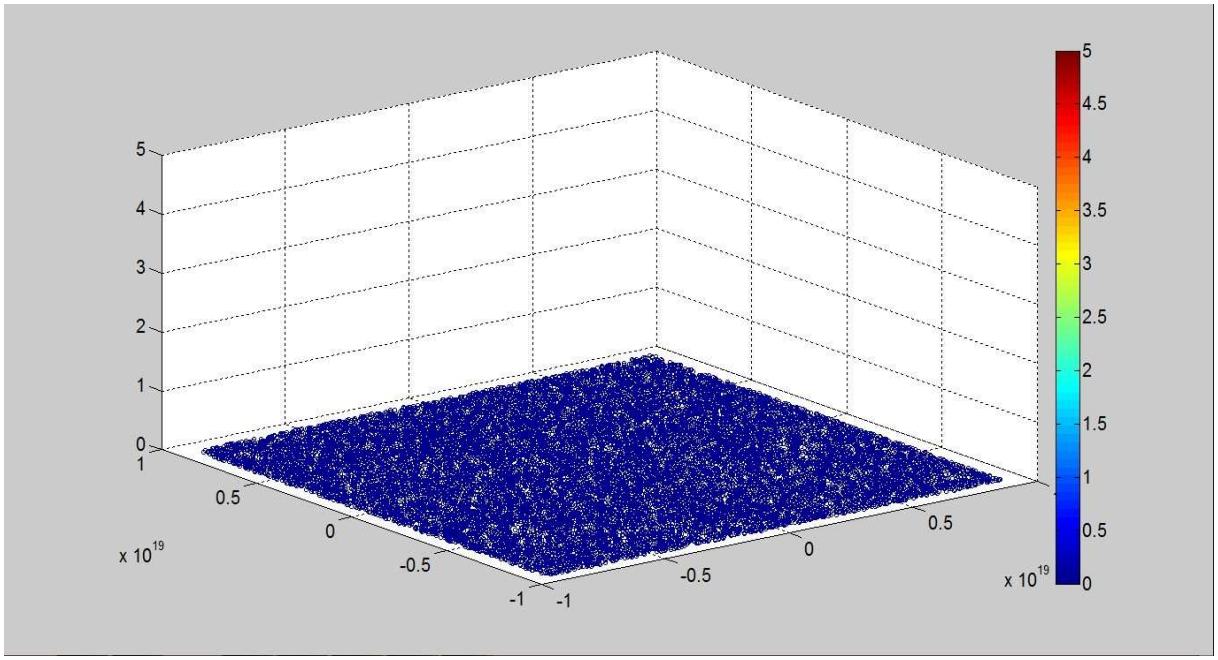

Figure 5.56. Relative error of EXU_ALU at voltage 0.7V in linear space

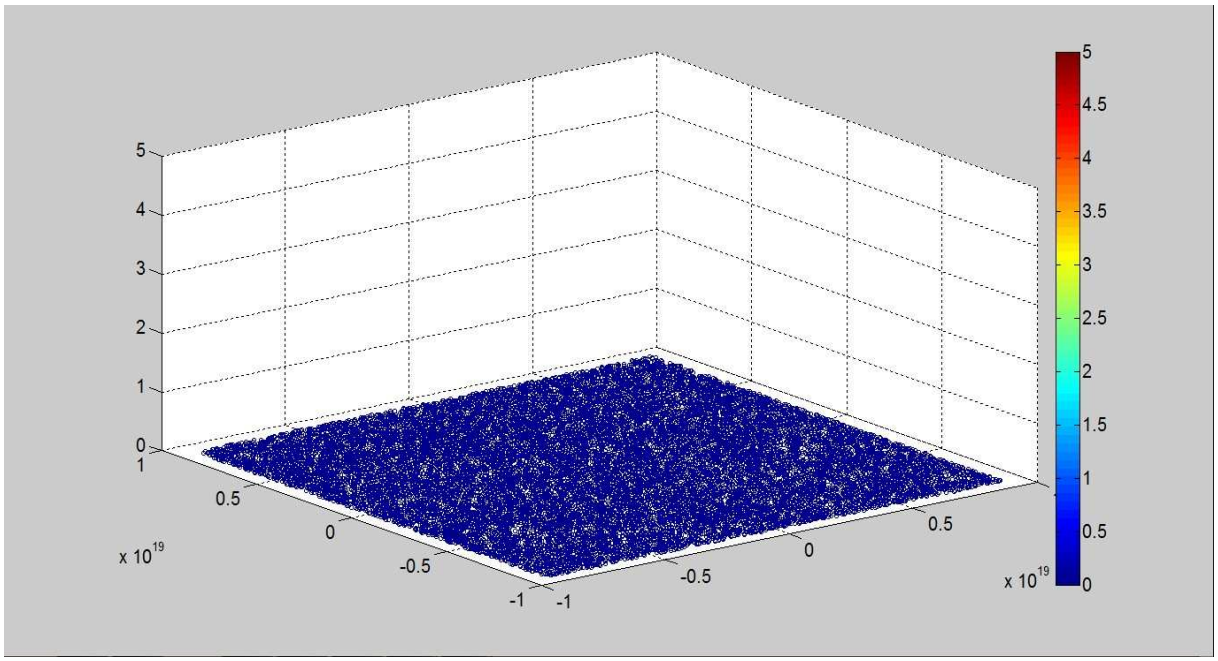

Figure 5.57. Relative error of EXU_ALU at voltage 0.6V in linear space

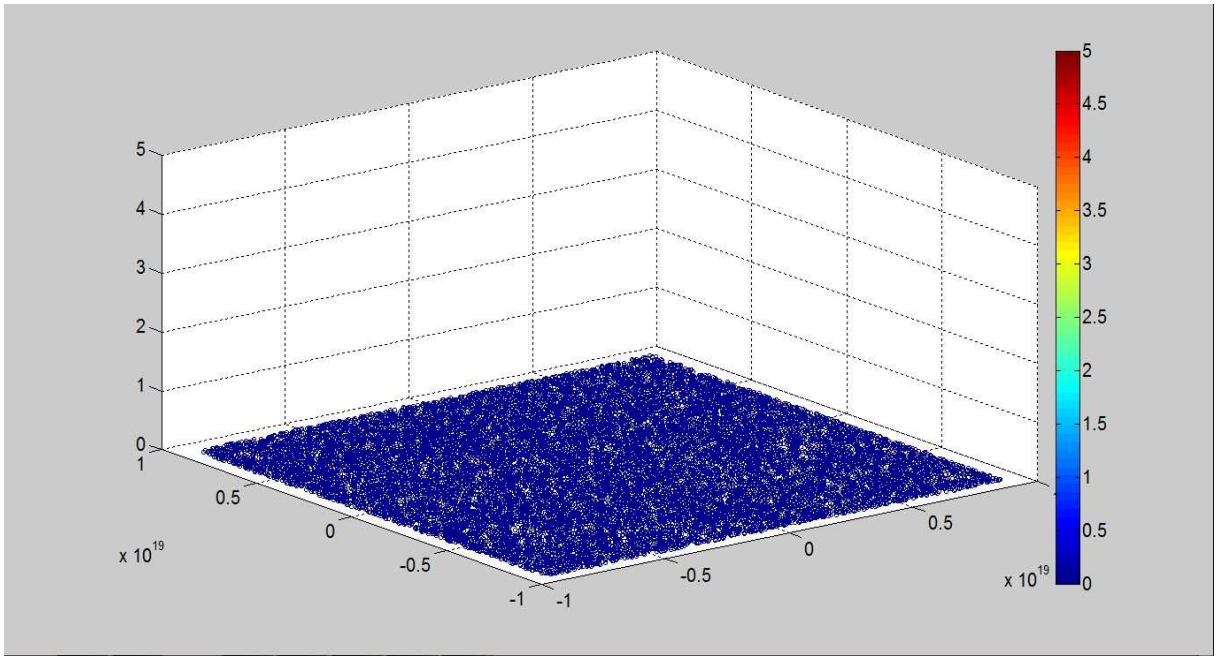

Figure 5.58. Relative error of EXU_ALU at voltage 0.5V in linear space

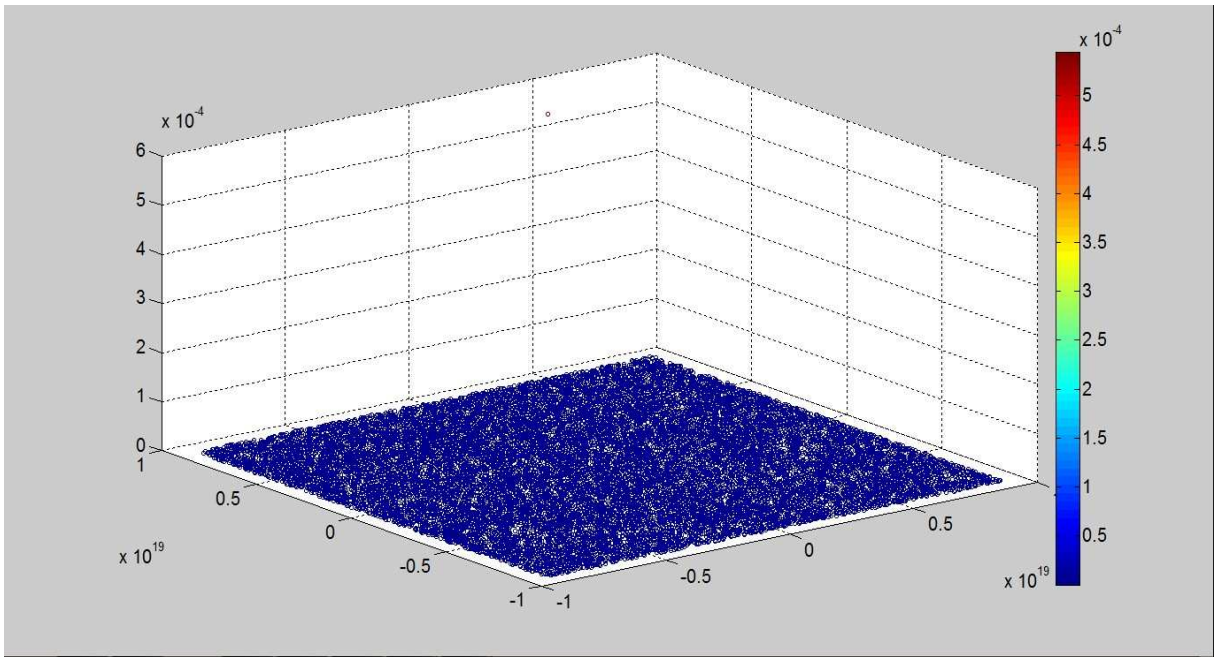

Figure 5.59. Relative error of EXU_ALU at voltage 0.4V in linear space

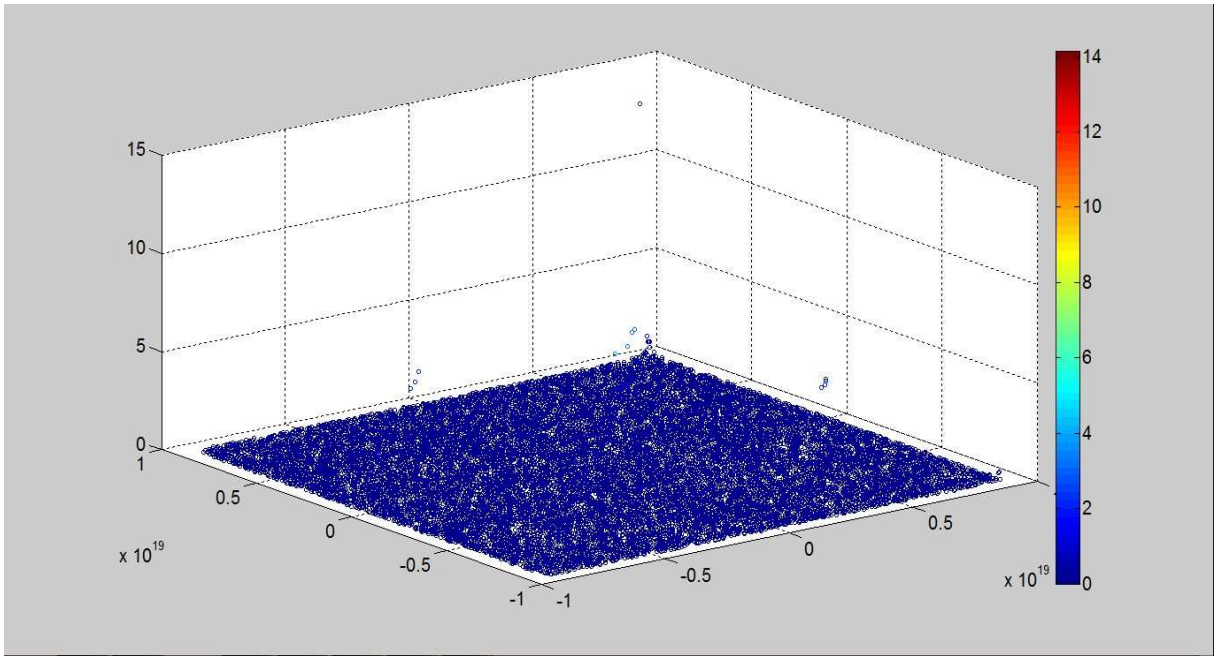

Figure 5.60. Relative error of EXU_ALU at voltage 0.3V in linear space

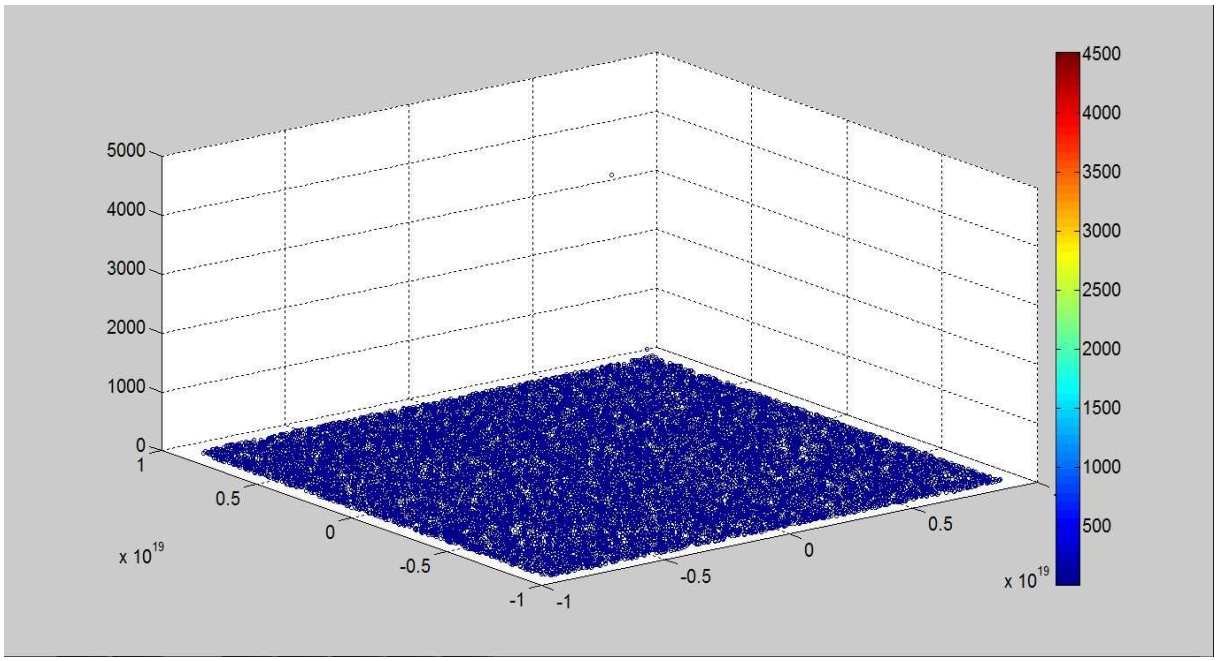

Figure 5.61. Relative error of EXU_ALU at voltage 0.2V in linear space

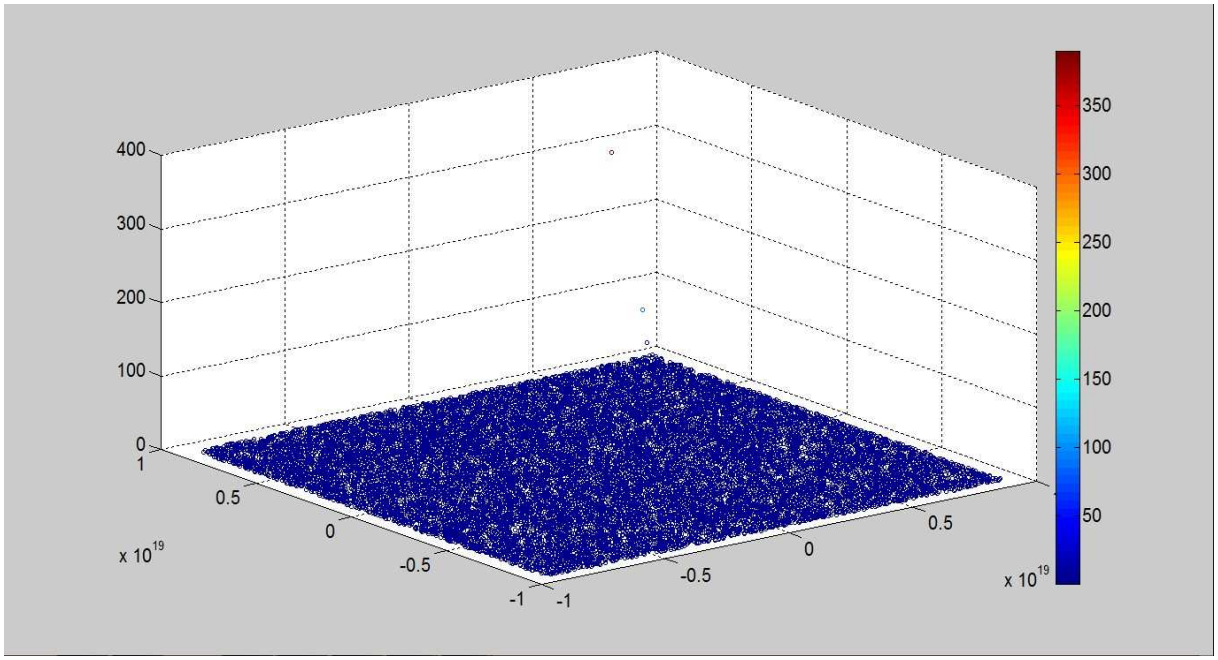

Figure 5.62. Relative error of EXU_ALU at voltage 0.1V in linear space

Then we show the relative error of FPU_MUL in Figure 5.63 to Figure 5.73. Notice that in voltage levels 1.1V to 0.9V, the relative error is shown in linear space. From 0.8V, the relative error becomes huge, and we use log10(relative error) to show them. However, for voltage 0.4V and below, the relative error is shown in linear space again, and it is always 1. It is because the output we get in these operating conditions is always 0, and it makes the relative error always 1.

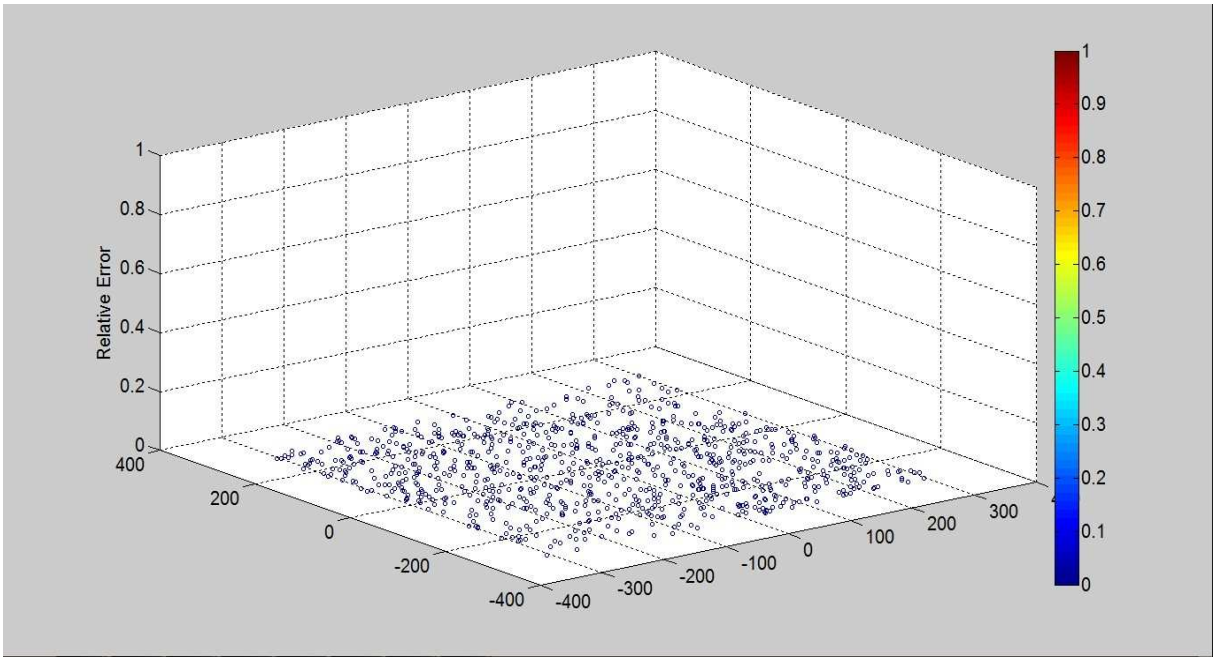

Figure 5.63. Relative error of FPU_MUL at voltage 1.1V in linear space

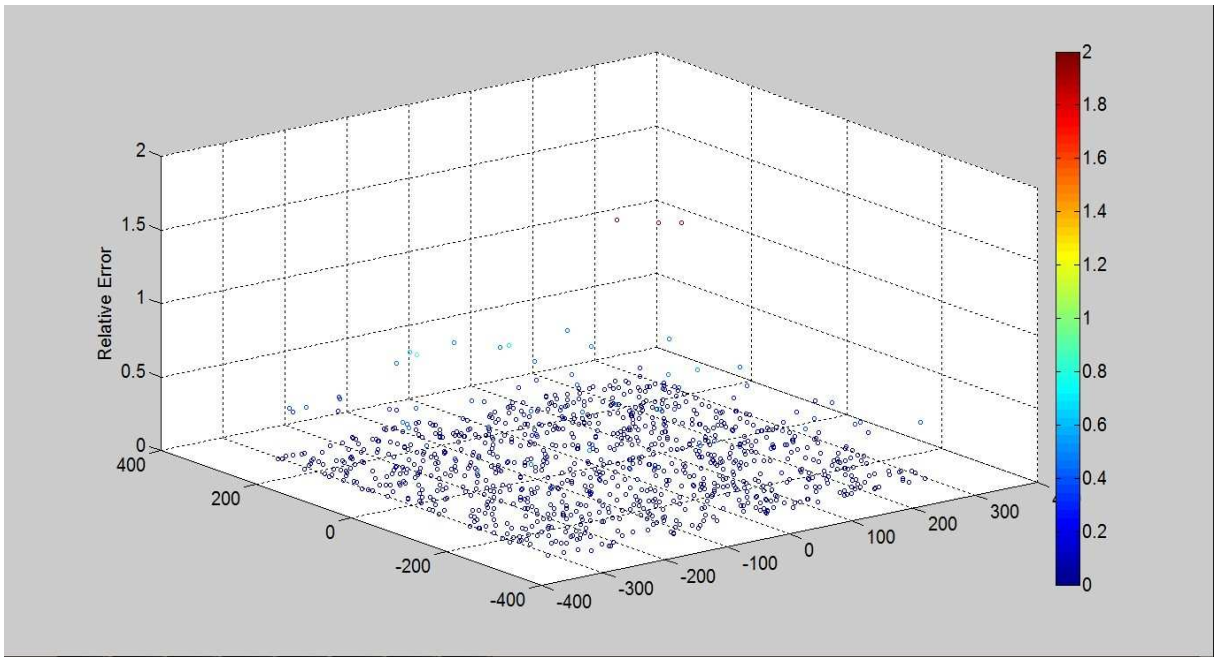

Figure 5.64. Relative error of FPU_MUL at voltage 1.0V in linear space

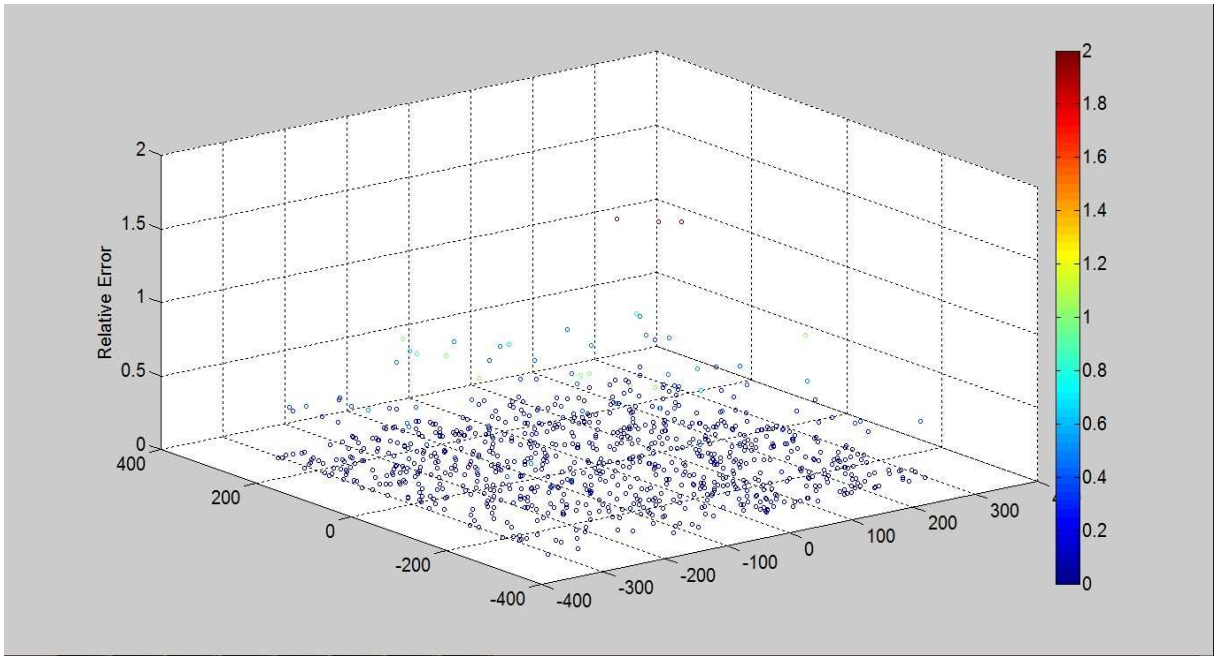

Figure 5.65. Relative error of FPU_MUL at voltage 0.9V in linear space

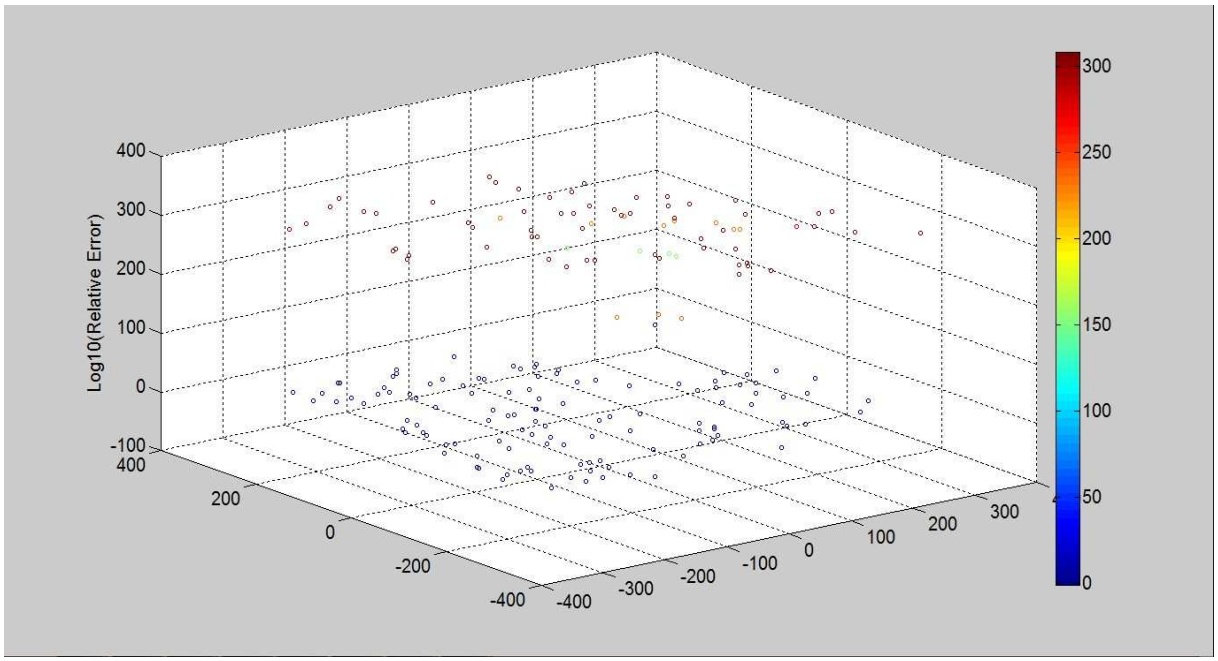

Figure 5.66. Relative error of FPU_MUL at voltage 0.8V in logarithmic space

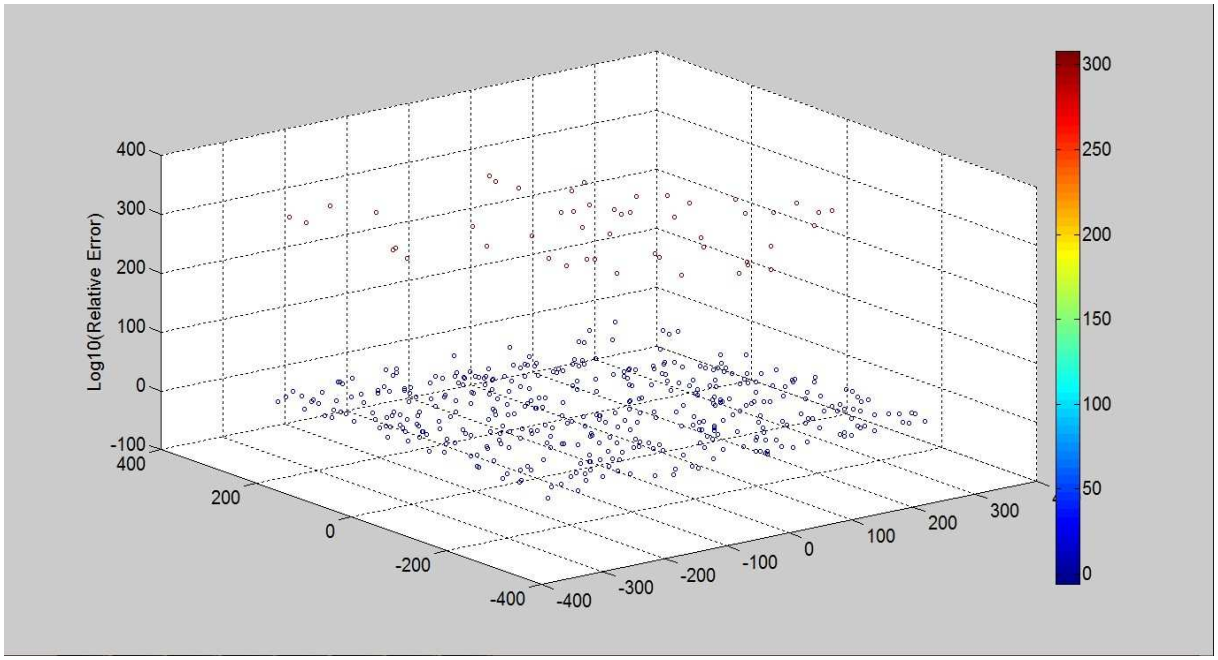

Figure 5.67. Relative error of FPU_MUL at voltage 0.7V in logarithmic space

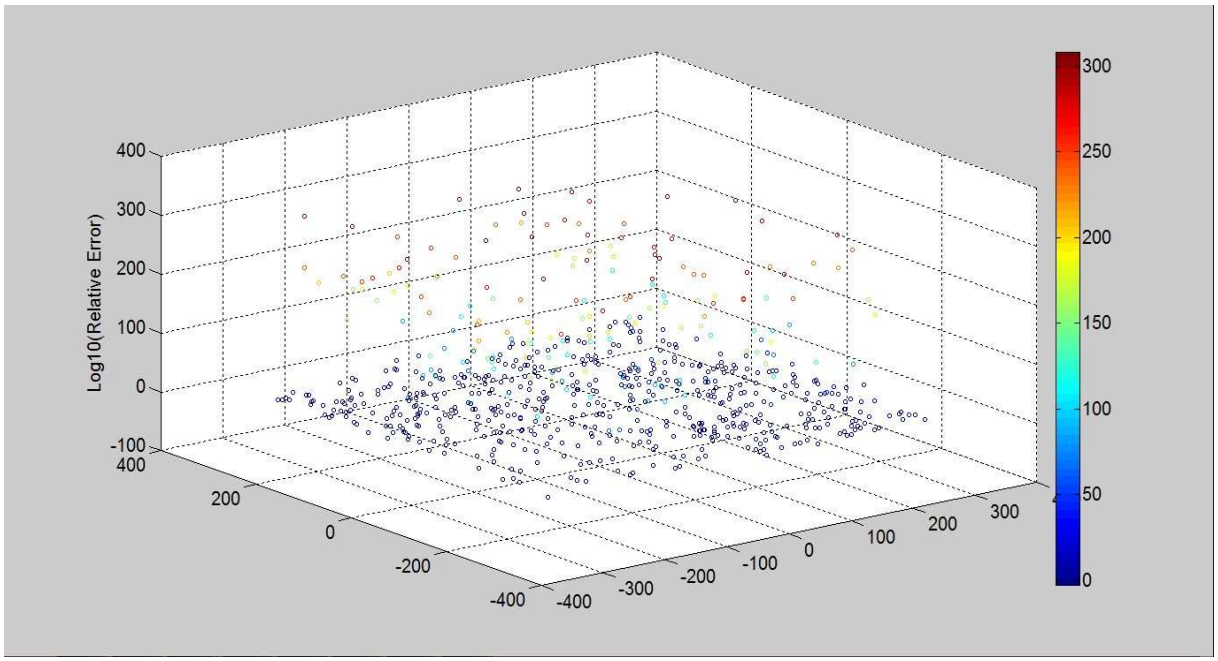

Figure 5.68. Relative error of FPU_MUL at voltage 0.6V in logarithmic space

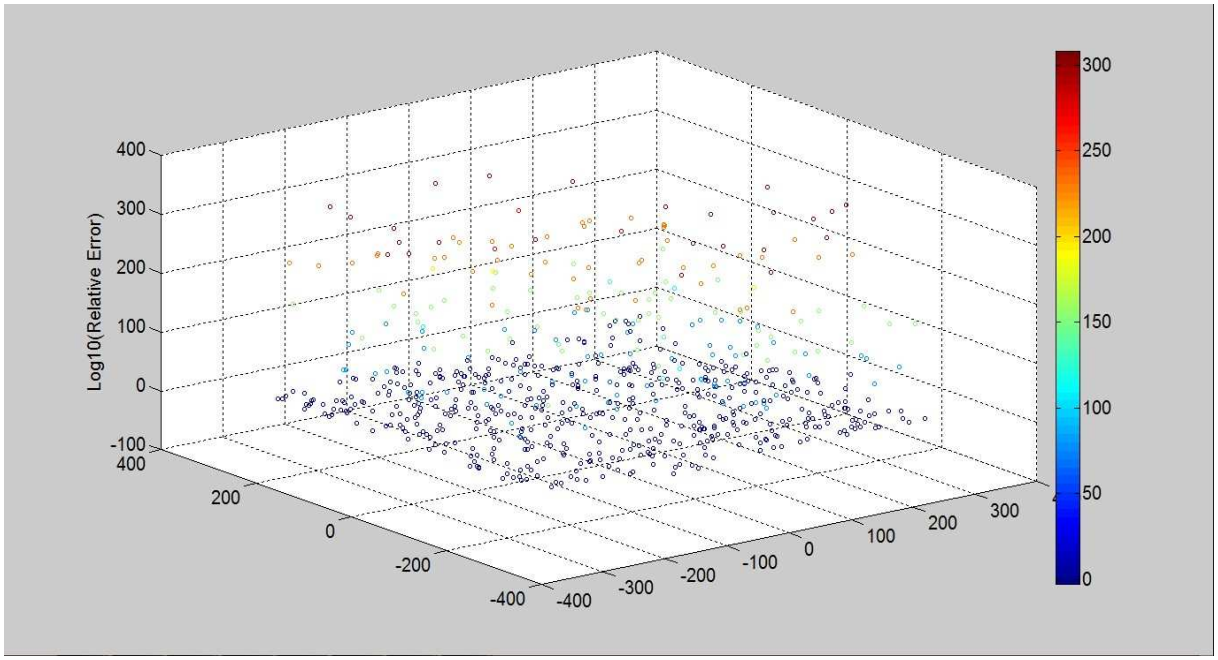

Figure 5.69. Relative error of FPU_MUL at voltage 0.5V in logarithmic space

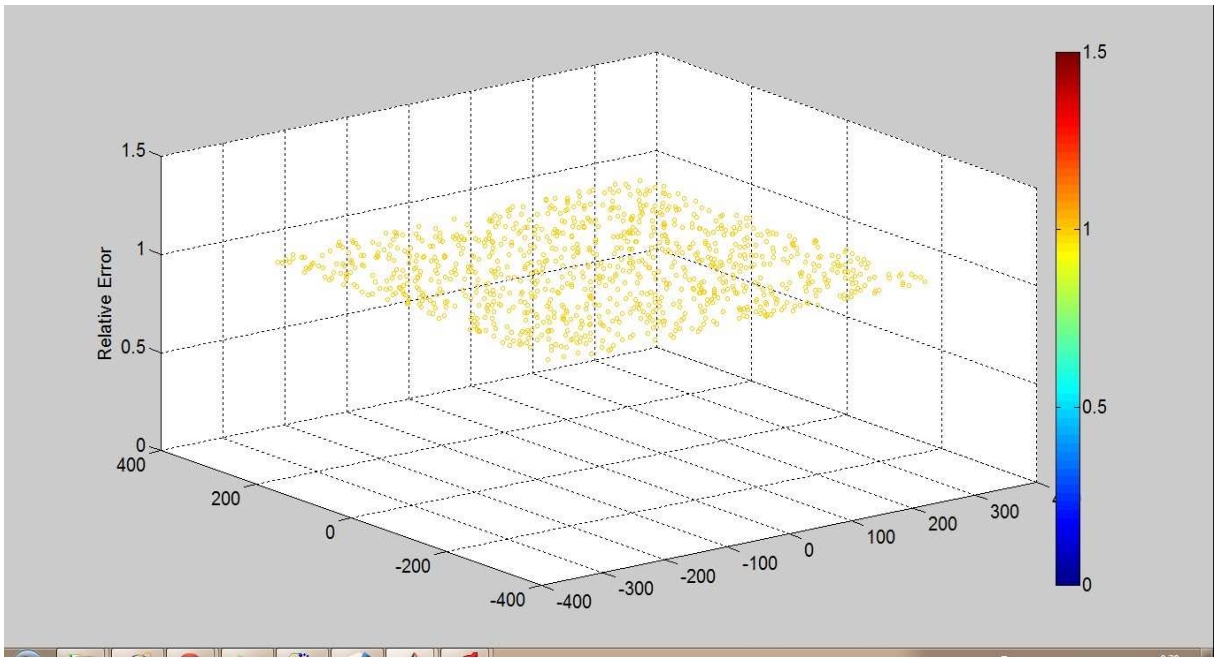

Figure 5.70. Relative error of FPU_MUL at voltage 0.4V in linear space

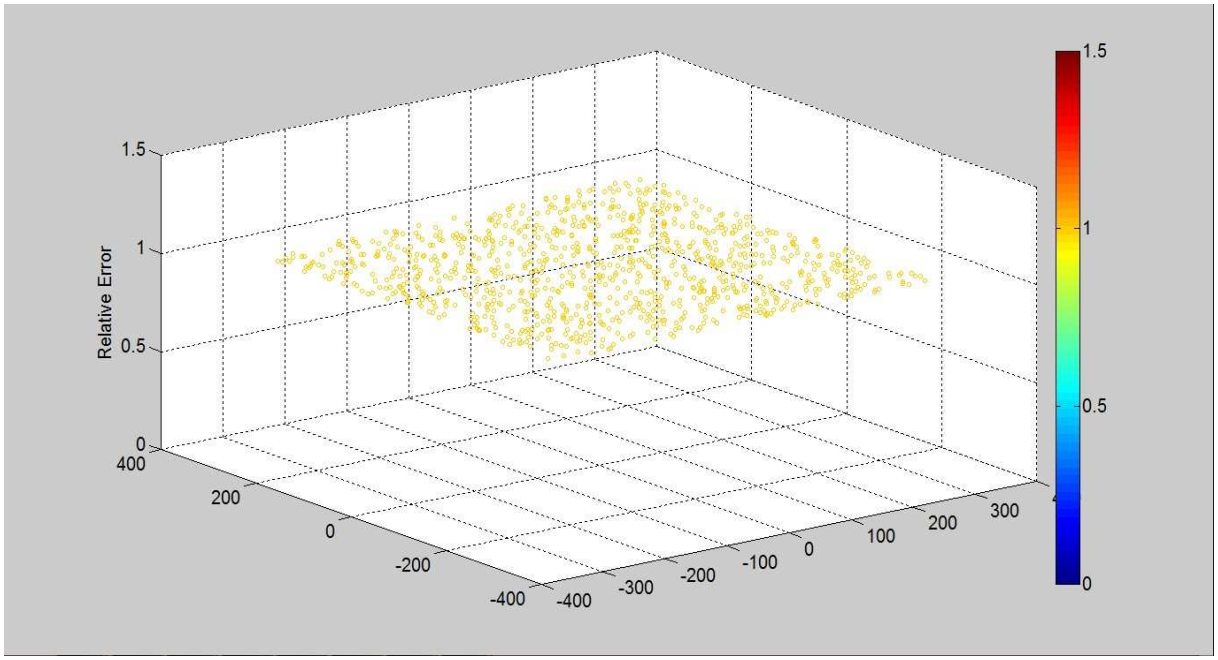

Figure 5.71. Relative error of FPU_MUL at voltage 0.3V in linear space

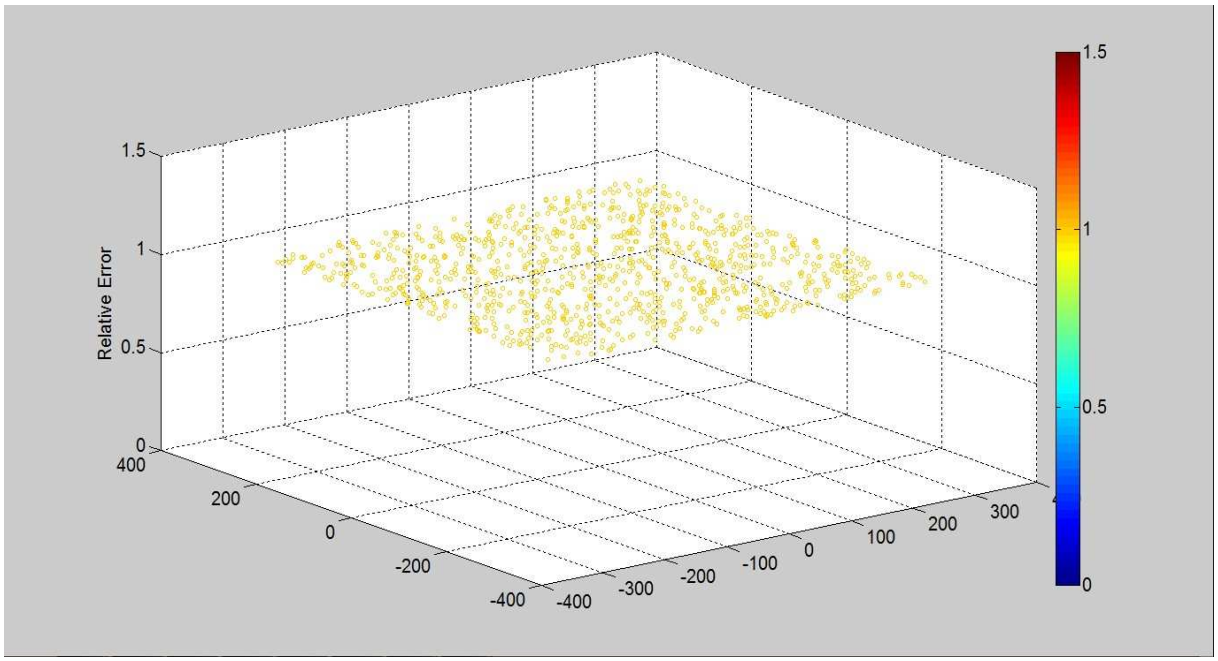

Figure 5.72. Relative error of FPU_MUL at voltage 0.2V in linear space

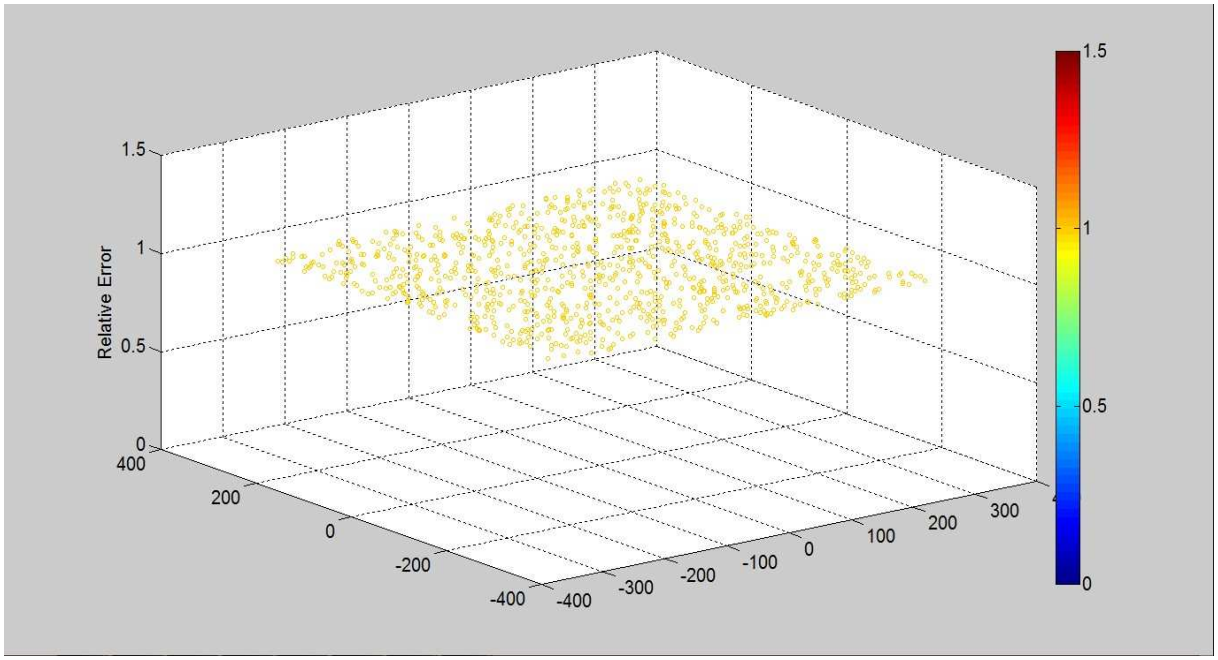

Figure 5.73. Relative error of FPU_MUL at voltage 0.1V in linear space

5.2. Error Tolerance of JPEG

PSNR of JPEG decompression with injected errors is shown in Figure 5.74 [**33**]. The results are given in lines regarding the bit positions of error injection.

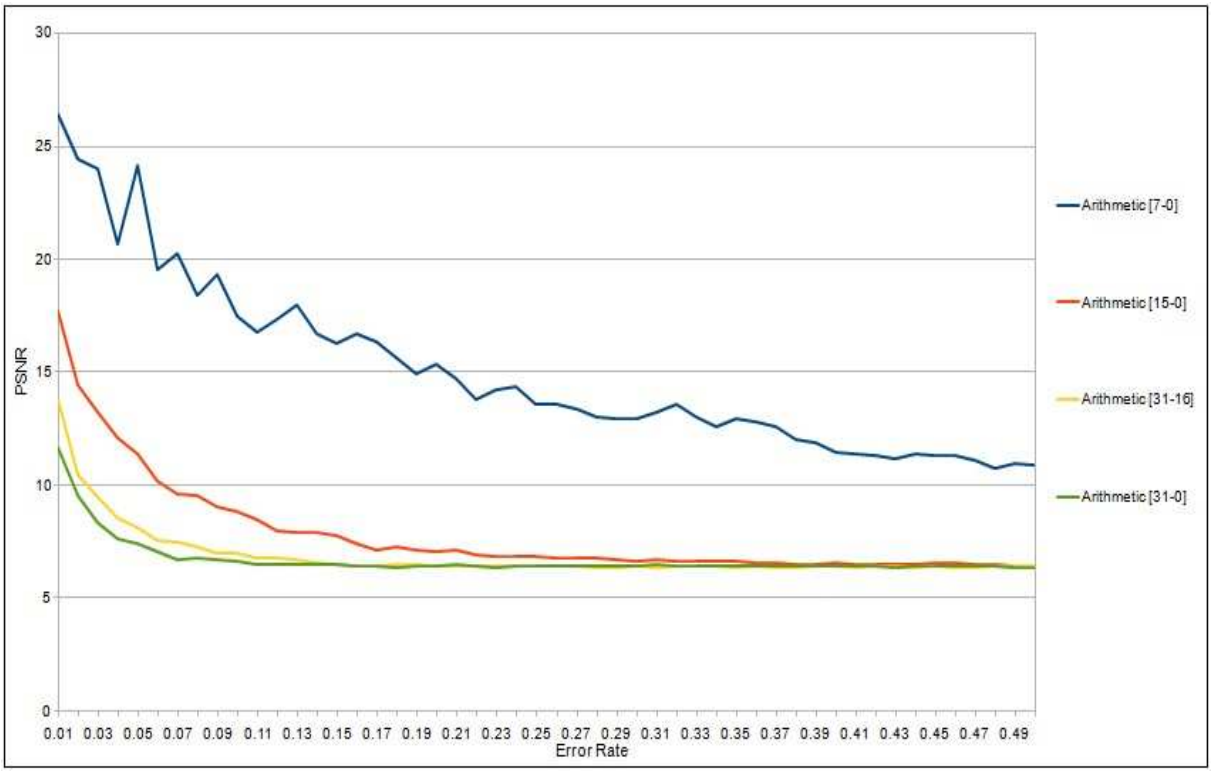

Figure 5.74. PSNR of error injected JPEG decompression

Generally speaking, the results show little resilience to errors. Even at error rate of 0.1, PSNR degrades drastically and the picture is blurring. As shown by the dot lines at 0.1 error rate, PSNR decreases by at least 33%.

Comparing the four lines, we can see that errors injected at lower bits have smaller impacts on the output quality. This implies that, if we treat the bits differently, specifically

by supplying high order bits with higher voltage, the impact on the output quality will be more limited. This is a accuracy-power trade-off at smaller granularity.

We have omit the result for error rate above 50% to make the graph clearer. Error rate larger than 50% will produce images with similar quality. There will be a little increase when error rate approaches 100%. In this case, all bits are flipped and it gives better quality than randomly flip some bits, but this does not make the picture usable.

CHAPTER 6

Discussion

The results in the previous chapter shows pessimistic vision for error resilience. For JPEG decompression, any error rate above 0.1% will make objects in the image become hard to recognize. However, we also notice that for some particular execution unit (EXU_ALU), there exists great tolerance for low supply voltage. We can safely decrease the voltage to 50% without encountering any timing errors. This will bring us 75% saving of dynamic power and 50% saving of static power in the EXU_ALU. Only at voltage level of 0.2V and 0.1V, the quality degradation will become a serious problem for the user, as indicated by the two vertical lines in Figure 6.1.

Since JPEG decompression only involves integer operations, EXU_ALU accounts for a great percentage in overall dynamic power consumption, and it means significant processor power saving for this application as well. Our study is limited to only one multimedia application by now. We have reason to believe that with more extensive study, other applications that are more resilient will be identified.

The future work of this project involves three aspects.

First, we will optimize the simulation flow and continue to characterize other execution units. The problem with mixed-signal simulation is its performance. Every time a gate is simulated in analog, the information will be passed to circuit level digital simulator but never recorded. Next time the gate is invoked, it has to be simulated again, even though the same simulation had just been done and the information had been obtained.

Considering there are many gates of the same type in one unit, and these gates might switch frequently, this is a huge waste of simulation time. For example, when we simulate FPU_ADD with 10,000 test vectors, each job takes 4 to 5 days to finish. Since we have 10 permutations for each voltage level, and we sweep through 12 voltage levels, the whole simulation time is more than 2 weeks. To solve this problem, we are trying to develop other simulation methods. The new method will be done in two phases. In phase one, we will run some SPICE-like simulations with certain operating conditions we give as input, and the simulations will create a new library with low voltage characteristics. In phase two, the circuit simulator will refer to the library for all necessary information and run as a pure digital simulator. The analog simulation is done for each gate only once, no

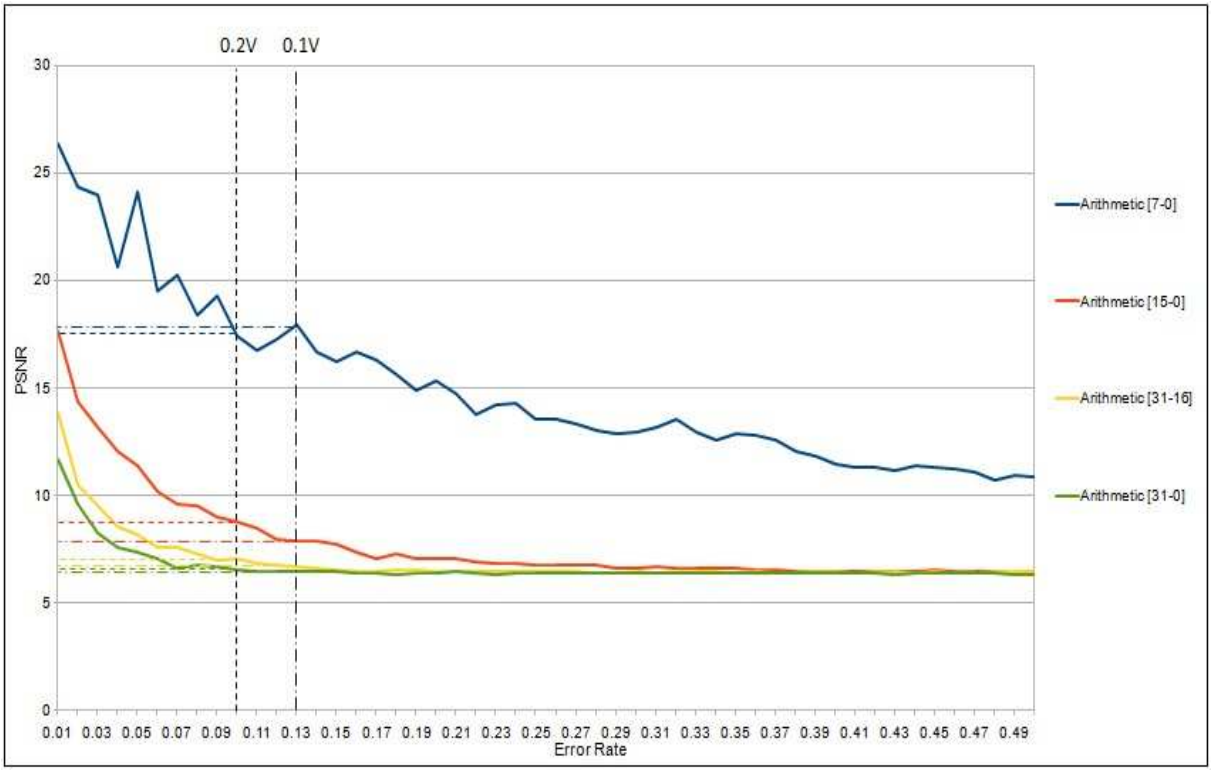

Figure 6.1. PSNR of error injected JPEG decompression

matter how many input vectors we have or how complex the circuit is. This will greatly improve the simulation performance.

Second, we will develop programming language constructs that denote the reliability guarantees required by different sections of the code or data. These constructs specify which variables and code segments are allowable to hardware errors, and their tolerance margins. In turn, the compiler maps these constructs to specialized instructions that direct the core to steer the computation to a functional and storage unit with a specific reliability level, by changing its operating voltage. These reliability constraints could be estimated manually by the user through experimentation or other methods, or automatically by modified quality-of-service profiler tools.

Third, on the hardware end, to meet the required fidelity constraints set by the software layer, dynamically scale voltage to minimize the power consumption at a given reliability level, based on experimental models of hardware behavior at each voltage. With execution units characterization and the proper hardware design, the system as a whole can guarantee the reliability levels required by the software.

CHAPTER 7

Conclusion

Using mixed-signal simulations at netlist level, we have characterized below-nominal supply voltage behaviours of three execution units. We calculate the error rate of their execution output to indicate the tolerance to low supply voltage. The results are distinct for these three execution units. For floating-point units, timing errors appear right after the supply voltage is lowered, while for integer addition and logical operations, the unit continues to operated correctly until the supply voltage is close to the threshold voltage of the MOS gates. With this characterization, we can tune the supply voltage specifically to each execution unit, according to the power and fidelity requirements. Thus, by striking a balance between computational accuracy and supply voltage, and through software/hardware cooperation, we anticipate Elastic Fidelity will successfully tackle the ongoing power crisis in processor design.

References

[1] Horowitz, Mark, Elad Alon, Dinesh Patil, Samuel Naffziger, Rajesh Kumar, and Kerry Bernstein. "Scaling, power, and the future of CMOS." In *Electron Devices Meeting, 2005. IEDM Technical Digest. IEEE International*, pp. 7-pp. IEEE, 2005.

[2] Glanz, J. A. M. E. S. "Google details, and defends, its use of electricity." *The New York Times* (2011).

[3] Jeong, Kwangok, Andrew B. Kahng, and Kambiz Samadi. "Impact of guardband reduction on design outcomes: A quantitative approach." *Semiconductor Manufacturing, IEEE Transactions on* 22, no. 4 (2009): 552-565.

[4] Kahng, Andrew B., Seokhyeong Kang, Rakesh Kumar, and John Sartori. "Designing a processor from the ground up to allow voltage/reliability tradeoffs." In *High Performance Computer Architecture (HPCA)*, 2010 IEEE 16th International Symposium on, pp. 1-11. IEEE, 2010.

[5] Patel, J. "Cmos process variations: A critical operation point hypothesis." In *Online Presentation*. 2008

[6] Rabaey, Jan M., Anantha P. Chandrakasan, and Borivoje Nikolić. *Digital Integrated Circuits*, 2/*E*. Prentice Hall, 2003.

[7] Yeo, Kiat-Seng, and Kaushik Roy. i*Low voltage, low power VLSI subsystems*. McGraw-Hill, 2005.

[8] Sun Microsystems. "OpenSPARC T1 Microarchitecture Specification", Part No. 819-6650-11, Revision B, February 2009. http://www.opensparc.net/opensparc-t1/index.html

[9] Li, Xuanhua, and Donald Yeung. "Application-level correctness and its impact on fault tolerance." In *High Performance Computer Architecture, 2007. HPCA 2007. IEEE 13th International Symposium on*, pp. 181-192. IEEE, 2007.

[10] Zadeh, Lotfi A. "Fuzzy logic, neural networks, and soft computing." *Communications of the ACM* 37, no. 3 (1994): 77-84.

[11] Zadeh, Lotfi A. "Some reflections on soft computing, granular computing and their roles in the conception, design and utilization of information/intelligent systems." *Soft Computing-A fusion of foundations, methodologies and applications* 2, no. 1 (1998): 23-25.

[12] Mallik, Arindam, and Gokhan Memik. "A case for clumsy packet processors." In *Proceedings of the 37th annual IEEE/ACM International Symposium on Microarchitecture*, pp. 147-156. IEEE Computer Society, 2004.

[13] Harizopoulos, Stavros, and Anastassia Ailamaki. "StagedDB: Designing database servers for modern hardware." *IEEE Data Eng. Bull* 28, no. 2 (2005): 11-16.

[14] Chakraborty, Koushik, Philip M. Wells, and Gurindar S. Sohi. "Computation spreading: employing hardware migration to specialize CMP cores on-the-fly." *ACM SIGOPS Operating Systems Review* 40, no. 5 (2006): 283-292.

[15] de Kruijf, Marc, Shuou Nomura, and Karthikeyan Sankaralingam. "A unified model for timing speculation: Evaluating the impact of technology scaling, CMOS design style, and fault recovery mechanism." In *Dependable Systems and Networks (DSN), 2010 IEEE/IFIP International Conference on* , pp. 487-496. IEEE, 2010.

[16] Firouzi, Farshad, Mostafa E. Salehi, Fan Wang, and Sied Mehdi Fakhraie. "An accurate model for soft error rate estimation considering dynamic voltage and frequency scaling effects." *Microelectronics Reliability* 51, no. 2 (2011): 460-467.

[17] Roberts, David, Todd Austin, David Blauww, Trevor Mudge, and Krisztin Flautner. "Error analysis for the support of robust voltage scaling." In *Quality of Electronic Design*, 2005. ISQED 2005. Sixth International Symposium on, pp. 65-70. IEEE, 2005.

[18] Burd, Thomas D., and Robert W. Brodersen. "Design issues for dynamic voltage scaling." In Low Power Electronics and Design, 2000. *ISLPED'00. Proceedings of the 2000 International Symposium on*, pp. 9-14. IEEE, 2000.

[19] Choudhury, Mihir R., and Kartik Mohanram. "Masking timing errors on speed-paths in logic circuits." In *Design, Automation & Test in Europe Conference & Exhibition, 2009. DATE'09.*, pp. 87-92. IEEE, 2009.

[20] Shafik, Rishad A., Bashir M. Al-Hashimi, and Krishnendu Chakrabarty. "Soft error-aware design optimization of low power and time-constrained embedded systems."

In *Proceedings of the Conference on Design, Automation and Test in Europe*, pp. 1462-1467. European Design and Automation Association, 2010.

[21] Ernst, Dan, Nam Sung Kim, Shidhartha Das, Sanjay Pant, Rajeev Rao, Toan Pham, Conrad Ziesler et al. "Razor: A low-power pipeline based on circuit-level timing speculation." In *Microarchitecture, 2003. MICRO-36. Proceedings. 36th Annual IEEE/ACM International Symposium on*, pp. 7-18. IEEE, 2003.

[22] Sarangi, Smruti, Brian Greskamp, Abhishek Tiwari, and Josep Torrellas. "EVAL: Utilizing processors with variation-induced timing errors." In *Microarchitecture, 2008. MICRO-41. 2008 41st IEEE/ACM International Symposium on*, pp. 423-434. IEEE, 2008.

[23] Greskamp, Brian, Lu Wan, Ulya R. Karpuzcu, Jeffrey J. Cook, Josep Torrellas, Deming Chen, and Craig Zilles. "Blueshift: Designing processors for timing speculation from the ground up." In *High Performance Computer Architecture, 2009. HPCA 2009. IEEE 15th International Symposium on*, pp. 213-224. IEEE, 2009.

[24] Carreira, Joo, Henrique Madeira, and Joo Gabriel Silva. "Xception: A technique for the experimental evaluation of dependability in modern computers." *Software Engineering, IEEE Transactions on 24, no.* 2 (1998): 125-136.

[25] Lee, Chunho, Miodrag Potkonjak, and William H. Mangione-Smith. "MediaBench: a tool for evaluating and synthesizing multimedia and communicatons systems." In *Proceedings of the 30th annual ACM/IEEE international symposium on Microarchitecture*, pp. 330-335. IEEE Computer Society, 1997.

[26] Mukherjee, Shubhendu S., Christopher Weaver, Joel Emer, Steven K. Reinhardt, and Todd Austin. "A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor." In *Microarchitecture, 2003. MICRO-36. Proceedings. 36th Annual IEEE/ACM International Symposium on*, pp. 29-40. IEEE, 2003.

[27] Shivakumar, Premkishore, Michael Kistler, Stephen W. Keckler, Doug Burger, and Lorenzo Alvisi. "Modeling the effect of technology trends on the soft error rate of combinational logic." In *Dependable Systems and Networks, 2002. DSN 2002. Proceedings. International Conference on*, pp. 389-398. IEEE, 2002.

[28] Shye, Alex, Benjamin Scholbrock, and Gokhan Memik. "Into the wild: studying real user activity patterns to guide power optimizations for mobile architectures." In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 168-178. ACM, 2009.

[29] Shye, Alex, Yan Pan, Ben Scholbrock, J. Scott Miller, Gokhan Memik, Peter A. Dinda, and Robert P. Dick. "Power to the people: Leveraging human physiological traits to control microprocessor frequency." In *Microarchitecture, 2008. MICRO-41. 2008 41st IEEE/ACM International Symposium on*, pp. 188-199. IEEE, 2008.

[30] Sampson, Adrian, Werner Dietl, Emily Fortuna, Danushen Gnanapragasam, Luis Ceze, and Dan Grossman. "EnerJ: Approximate data types for safe and general low-power computation." *ACM SIGPLAN Notices* 46, no. 6 (2011): 164-174.

[31] Esmaeilzadeh, Hadi, Adrian Sampson, Luis Ceze, and Doug Burger. "Architecture support for disciplined approximate programming." In *Proceedings of the seventeenth international conference on Architectural Support for Programming Languages and Operating Systems*, pp. 301-312. ACM, 2012.

[32] Krimer, Evgeni, Patrick Chiang, and Mattan Erez. "Lane decoupling for improving the timing-error resiliency of wide-SIMD architectures." In *Proceedings of the 39th International Symposium on Computer Architecture*, pp. 237-248. IEEE Press, 2012.

[33] Personal communication with George Tziantzioulis, Northwestern University, 2011-2012.