

Low Power Probabilistic Floating Point Multiplier Design

AMAN GUPTA^{#¶}, SATYAM MANDAVALLI[‡], VINCENT J. MOONEY^{§*&¶}, KECK-VOON LING^{§¶}, ARINDAM BASU^{§¶}, HENRY JOHAN^{#¶} AND BUDIANTO TANDIANUS^{#¶}

[‡]International Institute of Information Technology, Hyderabad, India

[§]School of Electrical and Electronic Engineering, Nanyang Technological University (NTU), Singapore

[¶]School of Computer Engineering, Nanyang Technological University (NTU), Singapore

[&]School of Electrical and Computer Engineering, Georgia Institute of Technology, Georgia, USA

[¶]NTU-Rice Institute for Sustainable and Applied Infodynamics, Nanyang Technological University, Singapore

aman@students.iit.ac.in, satyam@iit.ac.in, mooney@ece.gatech.edu, {vjmoooney, ekvling, arindam.basu, henryjohan, budi0010}@ntu.edu.sg

Abstract— We present a low power probabilistic floating point multiplier. Probabilistic computation has been shown to be a technique for achieving energy efficient designs. As best known to the authors, this is the first attempt to use probabilistic digital logic to attain low power in a floating point multiplier. To validate the approach, probabilistic multiplications are introduced in a ray tracing algorithm used in computer graphics applications. It is then shown that energy savings of around 31% can be achieved in a ray tracing algorithm’s floating point multipliers with negligible degradation in the perceptual quality of the generated image.

Keywords- *probabilistic computation; floating point multiplication;*

I. INTRODUCTION

The floating point format provides a wide dynamic range number representation as compared to other number formats. This range comes at the cost of including a power (and area) hungry floating point unit in an architecture. The use of floating point in embedded systems can be limited because of its huge power utilization when implemented in hardware. But since there are applications which require a wide range of numbers, it becomes inevitable to include a floating point unit in the architecture. Hence, there is a need for designing low power circuits for floating point operations to bring down the overall power expenditure. Among the most extensively used floating point operations, floating point multiplication is typically the most frequent power consuming operation. For this reason, the focus of this paper is to attain low power floating point multiplication. In particular, we aim to explore a tradeoff traditionally underutilized, namely, the tradeoff between power (energy) and the probability of correct computation. The idea of probabilistic computation [1, 2] is to devote energy in a circuit such that more energy is invested in more significant calculations and less energy is invested in less significant calculations. Energy investment is reduced by operating the less significant circuitry at a lower supply voltage. But, as the technology scales down, it is predicted in [3] that thermal noise is going to have a considerable effect in the correct functioning of circuits. Hence, reducing voltage will lead to a decrease in the noise immunity of a circuit. This means that at a lower supply voltage, noise will have a greater impact which may lead to erroneous results. Thus, there is a need for modeling the effect of device noise for future technology nodes, which becomes all the more essential when using voltage scaling. This leads to generation of tradeoffs between energy and accuracy, as lower energy means lowering of supply voltage which in turn will lead to more erroneous calculations. Such an attempt to develop energy efficient signal processing using probabilistic computations was presented in [2]. Of course, it should be mentioned here that other sources of circuit error, such as single event upsets or random transistor

parametric variations, may also be able to be handled by the approach presented in this paper. Applications which generate data for human perception can produce “reasonably good” results without requiring exact computations. By “reasonably good” we mean that the perceived quality of the output is almost the same as the output generated from exact computations. The primary idea is to exploit the dynamics of human perceptual cognition to facilitate energy savings.

One application which extensively uses floating point numbers is ray tracing [10]. Ray tracing is a technique for generating photorealistic images by simulating the behavior of viewing and light rays as in an actual physical environment. The algorithm requires a large dynamic range of numbers and uses floating point representation for this purpose. A major share of the total arithmetic operations required by a typical ray tracing algorithm is composed of single precision floating point multiplications. Since this application uses floating point operations to generate data for human perception, we conduct experiments on the ray tracing algorithm to demonstrate the validity of the use of probabilistic multiplications to achieve low power in floating point calculations.

This paper proposes a probabilistic design of a floating point multiplier with the aim to attain large energy savings. The usability of the design is then harnessed in a graphics application with little to no observable effect on the visual quality of the generated images. The rest of the paper is organized as follows. Section II mentions some of the work done previously to achieve low power circuits for floating point multiplication. Section III presents the general floating point architecture. Section IV explains our floating point multiplier architecture and the low power techniques we use. The experimental method is explained in Section V, and the results are shown in Section VI. Section VII provides a discussion of the results while Section VIII concludes the paper.

II. PRIOR WORK

Efforts have been made previously to reduce the power consumed by floating point multipliers at the cost of having some calculation errors. This was achieved mostly by reducing the number of bits of the operands or by truncating hardware in the floating point multiplier. The results of floating point bit width reduction on various benchmarks are presented in [4]. It has also been shown in [4] that most of the benchmarks are unaffected by not rounding the multiplication result. Hence, removing the rounding unit led to further power reduction. Truncating hardware in the floating point multiplier was shown in [5, 6]. Use of probabilistic computation in a ripple carry adder and in a six bit integer array multiplier to achieve low power in a Fast Fourier Transform was demonstrated in [2]. The implementation of this method in a Discrete Fourier Transform technique for energy efficient design was shown in

[7]. Voltage assignment to minimize energy usage of a probabilistic four bit integer array multiplier was done in [11]. As best known to the authors of this paper, we present here the first attempt to implement probabilistic computing in a floating point multiplier.

III. BACKGROUND

A single precision floating point number, according to the IEEE 754 Standard [8], is represented as a 32-bit number with 23 mantissa bits, 8 exponent bits and 1 sign bit. The actual mantissa bit length is 24 bits including the ‘hidden 1’ before the binary point. The architecture of a single precision floating point multiplier is shown in Fig. 1. Multiplication of two floating point numbers requires addition of the 8-bit exponents and multiplication of the 24-bit mantissas of the operands. The sign bit of the result is calculated by an XOR operation on the sign bits of the operands. The result of the 24-bit mantissa multiplication is a 48-bit binary number which needs to be normalized. This is achieved by shifting the result to adjust the binary point. The exponent of the result is adjusted according to the shift. Normalization is followed by rounding the number to give an output result with a 24-bit mantissa. Out of these bits, the least significant 23 bits are stored as the mantissa of the output, and the most significant bit, which is the ‘hidden 1’, is not stored. These 23 bits, along with the 8 bits from the exponent sum and the calculated sign bit, form the 32-bit floating point result. The difference in the architecture we implement and a generic floating point multiplier, at a block level, is that the rounding unit is not implemented in the technique we propose. Hence, except for the Rounding Unit shown in Fig. 1 as the rectangle with dotted lines, the rest of the circuit of the implemented floating point multiplier remains the same as that of a generic floating point multiplier.

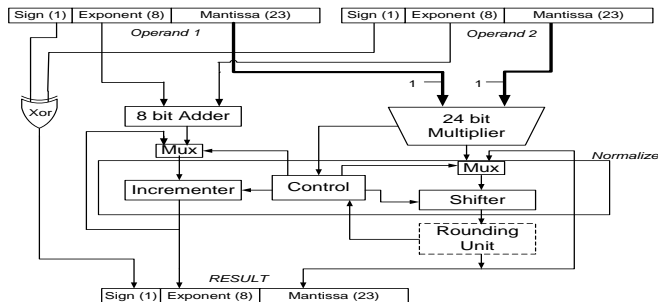


Figure 1. Floating Point Architecture

IV. PROBABILISTIC FLOATING POINT MULTIPLIER DESIGN

It has been shown in [4] that 81% of the total power consumption of a single precision floating point multiplier is from the 24-bit multiplier block, shown in Fig. 1. (From now until the end of the paper, please read ‘floating point multiplier’ to refer to a single precision or 32-bit floating point multiplier.) The rounding unit consumes around 18% of the total power [4]. Since we do not find rounding to be needed in this paper, around 99% of the floating point multiplier energy is consumed in the 24-bit multiplier block. The multiplier block consists of AND gates and full adders. About 5% of the 24-bit multiplier block energy is consumed by the AND gates used to calculate the partial products. We focus on applying low power techniques only in the full adders of the 24-bit multiplier block which consume 95% of the energy of the 24-bit multiplier block. Therefore, in this paper we take the energy savings

achieved by the low power techniques applied to the full adders to be approximately equivalent to the energy savings in the whole floating point multiplier. A 24-bit array multiplier is chosen to implement the multiplication since it is the most fundamental multiplier design. The structure of an array multiplier can be seen as full adders arranged in columns, each column leading to a significant bit of the result, as shown in Fig. 2. The full adder columns leading to calculation of more significant bits are termed as more significant columns while those leading to calculation of less significant bits are termed as less significant columns.

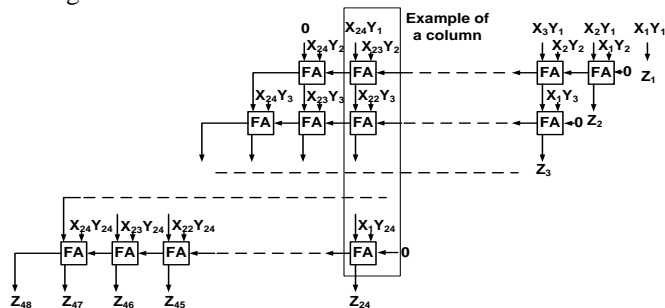


Figure 2. A 24-bit Array Multiplier Structure

We now explain two known techniques that can make a multiplier circuit run at a lower power at the cost of having lower precision and perhaps some errors at the output, and then we propose a new low power technique. Please note that we assume a scenario where a floating point unit is dynamically voltage scaled per application to minimize energy for a given application; therefore, we keep the full 24-bit multiplier hardware to be able to, if needed, use the full set of hardware for highest accuracy. The first technique, termed as the sleep technique, makes some of the less significant full adder columns of the array multiplier go into a sleep mode. The remaining full adder columns, which are not asleep, are operated at the nominal technology voltage. The larger the number of columns in sleep mode, the less is the energy consumed at the cost of decreased precision in the multiplication result. This scheme is analogous to hardware truncation [5, 6]. We do not use the word truncation because we just put the gates in a non-operational state and do not actually truncate them. Note that the results for the sleep scheme will be similar to the truncation techniques mentioned in Section II. The second known technique makes use of Biased Voltage Scaling (BIVOS). The method of BIVOS is introduced and explained in [2]. The full adder columns receive a biased supply voltage depending on the significance of the calculation performed by them, an approach which was first introduced in [11]. Full adders in the same column receive the same supply voltage [11]. This paper proposes a technique which builds on the sleep and BIVOS techniques. Starting from the least significant columns, some columns are turned to sleep mode while the rest of the columns are given a biased supply voltage. The less significant columns, which are not in sleep mode, are operated at a lower voltage as compared to the more significant columns. This BIVOS + Sleep technique is shown to out-perform either the sleep or the BIVOS technique alone.

We have assumed that the supply voltage can be chosen from one of five voltages, namely, 0.8V, 0.9V, 1.0V, 1.1V and 1.2V. Please note that we assume a System on Chip (SoC) scenario where voltage scaling is done for other SoC circuitry as well. Hence, the above mentioned five voltages are already

available. The voltage for each full adder column can be any one of the chosen supply voltages. A sequence of voltages which defines the supply voltage of each full adder column is termed as a ‘voltage profile’ of a multiplier. An example of a voltage profile, shown in Table I, operates the least significant 22 full adder columns in the sleep mode. The next two more significant columns are operated at 0.8V and so on.

TABLE I. EXAMPLE OF A VOLTAGE PROFILE

Columns (LSB-MSB)	Sleep(0V)	0.8V	0.9V	1.0V	1.1V	1.2V
	1-22	23-24	25-29	30-33	34-35	36-46

V. EXPERIMENTAL METHOD

The scheme of probabilistic computation presented in this paper uses scaling of the supply voltage. Our simulations use the Synopsys 90nm library. To perform the experiments, we have assumed that the supply voltage can be scaled to one of five voltages, namely, 0.8V, 0.9V, 1.0V, 1.1V and 1.2V. It should be mentioned here that the supply voltage can be reduced below 0.8V but due to large increase in the latency of the multiplier, we do not operate at voltages lower than 0.8V. Also, the value of the voltage can be any arbitrary value, e.g., 0.86, 0.92 but we have chosen voltages in steps of 0.1V to conduct the experiments.

A. Error Rate Characterization

Introduction of the low power techniques, discussed in Section IV, leads to errors at the output of the 24-bit multiplication. A single precision floating point multiplication requires a 24-bit multiplier. Simulating a 24-bit array multiplier in HSPICE for as few as 300 samples takes more than a day on a high end machine which has Intel Core 2 Quad CPU at frequency 2.66 GHz with 4 GB main memory. As introduction of probability is a statistical process, it is necessary to use a large number of samples for simulations. Also, the proposed technique involves voltage scaling and thermal noise which cannot be achieved in traditional digital simulation environments which are faster than HSPICE. Hence, there is a need for a simulator which can be used to quickly verify the applicability of the proposed techniques in the floating point architecture, with a reasonable accuracy. This motivated us to develop a simulator in the programming language C. We calculate the error rate of a 1-bit full adder from HSPICE and use that in a C simulator to calculate the error rate at each output bit of a 24-bit multiplier. This is explained in the following subsections.

1) *Error Rate Calculation for a 1-bit Full Adder Using HSPICE:* We perform HSPICE simulations with a 90nm technology library from Synopsys. The nominal supply voltage at this technology is 1.2V. The supply voltage of the full adders is varied over the mentioned five voltages. Deliberate injection of Gaussian noise at each full adder output is done to model the effect of thermal noise in the circuit as explained in [9]. The value of the root mean square (RMS) of noise is chosen so that there are no errors at the output of the full adder when operated at 1.2V even after injecting noise. The maximum noise RMS that does not cause any errors at 1.2V, obtained experimentally from HSPICE, comes out to be 0.15V. If the supply voltage is lower than 1.2V, there are errors at the output which increase as the voltage is lowered. This is how we predict a possible noisy future technology node (e.g., at 12nm); while of course no one can guarantee a prediction of the future, we nonetheless use

this model of randomly distributed uncorrelated errors due to thermal noise as a possible model of future probabilistic gates [1,2,3,7,9,11]. The full adder is simulated in HSPICE for the five stated supply voltages, at the given noise RMS. The full adder is fed by 100K random samples with uniform distribution of 1/0 at each input. The sum and the carry outputs are observed, and the probability of error for each output is calculated as explained in [9]. Error rate characterization of a full adder for all voltages is shown in Table II.

TABLE II. ERROR RATE CHARACTERIZATION OF A FULL ADDER

Supply Voltage	1.2V	1.1V	1.0V	0.9V	0.8V
Probability of error of sum	0.0	4E-5	5E-5	1.2E-4	3.9E-4
Probability of error of carry	0.0	0.0	2E-5	9E-5	3E-4

2) *Error Rate Calculation for a Multiplier Using a C Simulator:* The goal of the simulator is to generate error rates for each output of an array multiplier using the error rates of the full adder [9]. The process of obtaining error rates of a full adder from HSPICE was explained in the previous section. We state the timing assumptions that have been followed in the development of the simulator. It is assumed that all the new inputs to a gate arrive at the same time. Hence, before calculating the output of the current full adder in the circuit, all the gates whose outputs are the inputs of the current full adder under consideration have been simulated. The gates can be operated in sleep mode, which makes them non-operational. The outputs of such gates are considered to be logic 0.

The C simulator has a full adder model which takes as input three bits and generates a carry output and a sum output. Initially the correct outputs according to the inputs are calculated. Each full adder then introduces errors at its outputs in accordance with the supply voltage by using the probability of error values from the error rate characterization of the full adder. To model the effect of device noise, we introduce errors in the C simulator with the help of the inbuilt random number generator in C. A voltage profile, introduced in the previous section, along with the error rate characterization of the full adder for each voltage is given as input to the simulator. Each 24-bit input operand represents a mantissa of a floating point number. Since these numbers represent mantissas, the most significant bit, i.e., the 24th bit for both operands (where the LSB is the 1st bit) is fixed as 1 to model the ‘hidden 1’ of a normalized floating point mantissa. The least significant 23 bits are generated randomly with equal chances for each bit to be 1/0. A model of an array multiplier is developed in C using full adders which takes as input two 24-bit numbers and generates the 48-bit multiplication result. The binary operands are fed to the array multiplier model in the simulator.

Each full adder in the array multiplier is fed with inputs according to the input operands and the outputs of the previous gates, depending on the full adder’s position in the array structure of Fig. 2. The generated outputs are propagated to the next gates. Errors are generated at each full adder according to the supply voltage specified by the voltage profile, and the 48-bit output is generated. The output is then normalized. The least significant 24 bits of this normalized output are generally used for rounding. As the proposed floating point architecture does not use any rounding, these bits are not used past normalization. The most significant 24 bits of the 48-bit normalized output forms the mantissa of the probabilistic result. To calculate the error rate of each mantissa bit at the output, the probabilistic result needs to be compared with the

exact multiplication result. To do so, the 24-bit input operands are correctly multiplied and rounded, to calculate the exact multiplication result. The mantissa of the exact result is then compared with the mantissa of the probabilistic result. Please note that rounding is done only to calculate the exact result and not the probabilistic result. To generate the probability of error of each bit at the output, simulations are done for 100K samples. These probability values are then fed to the ray tracing application, where exact multiplications are replaced by probabilistic multiplications and an image is generated by using probabilistic multiplications.

To validate the error rate values generated by the C simulator, a comparison is done between a 6-bit array multiplier simulated in HSPICE and in the C simulator. The comparison was done for 50K samples for each of the three low power schemes mentioned in Section IV. Simulation of each of the low power schemes took two days on a high end machine. The average inaccuracy in the results generated by the C simulator is reported in Table III. The voltage profile used for each scheme is shown in Table IV. Note that the total number of full adder columns in a 6-bit multiplier is 10.

TABLE III. AVERAGE INACCURACY IN C SIMULATOR RESULTS

Low Power Scheme	SLEEP	BIVOS	BIVOS+SLEEP
Average Inaccuracy per Output Bit when Compared with HSPICE	4.13%	7.26%	5.29%

TABLE IV. VOLTAGE PROFILE FOR 6-BIT SIMULATIONS

		Sleep(0V)	0.8V	0.9V	1.0V	1.1V	1.2V
Columns (LSB - MSB)	SLEEP	1-3	-	-	-	-	4-10
	BIVOS	-	1-10	-	-	-	-
	BIVOS +SLEEP	1-3	4	5	6	7	8-10

B. Energy Characterization

Depending on the voltage profile, the circuit consumes different amounts of energy. The total energy consumed by a gate depends on the number of toggles that occur at the gate output and the energy consumed per output toggle. We calculate, from HSPICE, the energy consumed when the sum and the carry output of a full adder toggles. We then calculate the total number of toggles that occur in a multiplier using Verilog and hence find the total energy consumed in the multiplier. This is explained in the following sub sections.

1) *Energy Calculations for Sum and Carry Toggles Using HSPICE:* The total energy consumed by the full adder can be divided into the energy consumed for sum calculation and the energy consumed for carry calculation. Fig. 3 shows the transistor diagram we use for a 1-bit full adder with A, B, C, Sum and Carry as the three inputs and two outputs, respectively. We identify the part of the full adder circuit that calculates the sum output, shown in Fig. 3 with supply voltage named as ‘VDD_Sum’, and the part of the full adder circuit that calculates the carry output, shown in Fig. 3 with supply voltage named as ‘VDD_Carry’. We compute the energy consumed in each part when the full adder is fed with 50K samples in HSPICE. This provides the total energy consumed for sum calculation and the total energy consumed for carry calculation. The input samples for this experiment were generated using random number generation in C, and were fed as inputs in HSPICE, keeping uniform distribution of 1/0 at each input. During the sample generation, the input samples were analyzed to calculate the total number of toggles that will

be generated by these samples at the sum and the carry output of the full adder. We divide the total energy consumed for sum calculation by the total number of toggles for sum to get the energy consumed per sum toggle. Similarly, we calculate the energy consumed per carry toggle. These simulations are done for the five stated supply voltages, and a look up table for every supply voltage is generated as shown in Table V. Hence, for each supply voltage, the energy consumed per sum toggle, and the energy consumed per carry toggle are tabulated. We assume that energy is consumed only when any output toggles. For the 90nm process we use, leakage accounts for 0.1% of the energy consumed. For technologies with significant leakage, we would need to add static power to our model.

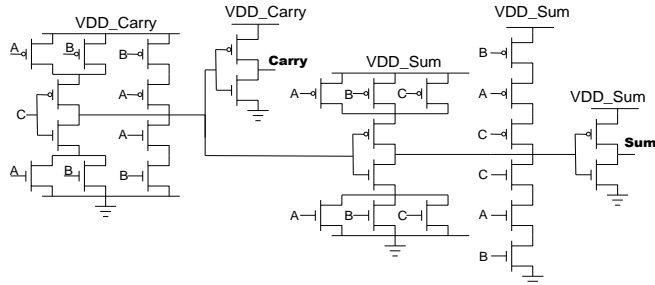


Figure 3. Full Adder Transistor Diagram

TABLE V. ENERGY CHARACTERIZATION OF A FULL ADDER

Supply Voltage	1.2V	1.1V	1.0V	0.9V	0.8V
Energy consumed per sum toggle (x E-14 Joules)	5.97	4.90	3.85	2.96	2.24
Energy consumed per carry toggle (x E-14 Joules)	7.02	5.80	4.6	3.51	2.68

2) Energy Calculations for a Multiplier Using Verilog:

The energy consumed per carry toggle and per sum toggle is calculated as explained in the previous section. To calculate the energy consumed by a larger circuit (e.g., an array multiplier) consisting of full adders, the average number of toggles for sum and carry output for each of the full adders in the circuit is required. A Verilog description of a 24-bit array multiplier was simulated in ALDEC Active-HDL 6.3 [12] and its Toggle Coverage feature was used to calculate the toggle rate. The number of toggles depends on the input to output delays of the full adder. Hence, for each input combination, the delays from each input to each output were calculated from HSPICE. This was done for all the five voltages considered, using the Synopsys 90nm library. Then, corresponding to each voltage, the delays were specified in the Verilog description. The toggle rate was calculated for each low power scheme, given a voltage profile, as the number of toggles is different for each scheme. Simulations were done for 50K random samples for each low power scheme with uniform distribution of 1/0 at each input bit. The calculated toggle rate was then multiplied with the energy per toggle values obtained from the energy characterization of the full adder, in accordance with the voltage profile, to find the total energy consumed by the circuit. To verify the energy values generated from toggle calculations, a 6-bit array multiplier was simulated in HSPICE and in Verilog for each of the low power schemes, for the same voltage profile shown in Table IV, and the results were compared. The energy consumed was calculated for 50K samples and the results are reported in Table VI. The final aim of the energy calculations is to compare the performance of the low power techniques, i.e., to know how much more/less

energy is consumed by a particular scheme as compared to the other schemes. Assume that simulation of two designs, say A and B, in HSPICE states that A consumes X% of the energy consumed by B. If the same designs were simulated using Verilog and the results show that A consumes X% of the energy consumed by B, then the comparisons are correct. Hence, instead of matching the actual energy values of A and B from HSPICE and Verilog, we calculate the energy consumed by A with respect to B from HSPICE and the energy consumed by A with respect to B from Verilog and compare that value. For comparisons of the low power schemes, we focus on relative accuracy rather than absolute accuracy, i.e., we compare all the schemes simulated in HSPICE with respect to one of the scheme simulated in HSPICE and calculate the percentage of energy consumed by each with respect to that one scheme. Similarly, we compare all the schemes simulated in Verilog with respect to one of the scheme simulated in Verilog and calculate the percentage of energy consumed by each with respect to that one scheme. This can be seen as normalization of the values with respect to that one scheme. The comparisons can be made with respect to any scheme. In Table VI, in the third column, comparisons are done for HSPICE and Verilog with respect to their corresponding Sleep techniques. As seen from Table VI, the inaccuracy in the Verilog results is under 5%.

TABLE VI. ENERGY CALCULATIONS

Low Power Scheme	Energy/sample (J) calculated from		Energy/sample calculated with respect to the corresponding Sleep scheme		Inaccuracy in Verilog results
	HSPICE	VERILOG	HSPICE	VERILOG	
SLEEP	1.92E-12	1.80E-12	100%	100%	-
BIVOS	9.22E-13	8.45E-13	48.03%	46.93%	2.29%
BIVOS+SLEEP	1.53E-12	1.37E-12	80.03%	76.45%	4.47%

C. Ray tracing

In the implemented ray tracing algorithm [10], single precision floating point multiplications account for 54% of the total number of arithmetic operations as shown in Fig. 4. Out of the total multiplications, around 72% of the multiplications were from determinant value calculations for matrices. These determinant calculations were of two types. Type 1 calculations were used in the algorithm to find the distance to the point of intersection. Type 2 calculations were used to find the interpolation parameters of the point of intersection. Introduction of errors in the type 1 multiplications leads to wrong distance calculations. If the distance calculated, using probabilistic multiplications, to the point of intersection of the ray, yields a result less than the exact distance, the ray falls short of the point of intersection. On the other hand, if the distance calculated is larger than the actual exact distance, the ray goes to a point beyond the actual intersection point. Both of these cases may lead to an increase in the number of rays in the simulation of light-object interaction. Due to the recursive nature of the algorithm, larger numbers of rays cause an increase in the total number of operations as compared to the case when exact calculations are done. The recursive nature can be seen by a loop formation in the flow chart of Fig. 4. Introduction of errors in the type 2 calculations only causes a displacement in the position of the point of intersection, which does not cause any increase in the number of arithmetic

operations executed by the algorithm. Type 2 operations account for 69% of the matrix calculations which is 50% of the total number of multiplications. Hence, type 2 multiplications which account for 50% of the total single precision floating point multiplications were made probabilistic according to the low power techniques explained in Section IV. The share of various operations shown in Fig. 4 is calculated by simulating the ray tracing algorithm for different graphics models such as Stanford Bunny, Stanford Dragon and Stanford Armadillo, and taking the average value for each operation. These models are courtesy of the Stanford Computer Graphics Laboratory.

Now we explain how the probability of error generated from the C simulator is injected in to the ray tracing algorithm. First, the correct multiplication result is calculated. This result is then converted into its 32-bit binary representation. Out of the 32 bits, the 23 mantissa bits are extracted and errors are introduced in the same manner as was done in Section V.A.2 in the C simulator. Errors are introduced only in the mantissa of the number while the exponent and sign bit calculations are error free. This erroneous 23-bit mantissa along with the 8-bit exponent and the sign bit of the correct result is converted into a floating point number. This number replaces the correct result in the ray tracing algorithm executed in C.

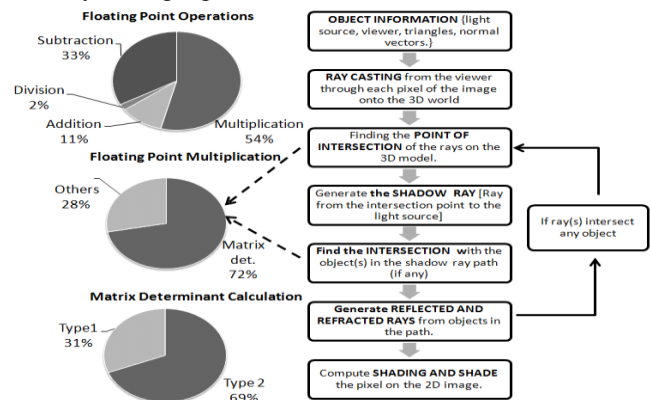


Figure 4. Ray Tracing Algorithm

VI. SIMULATION AND RESULTS

For the experiments, we first define an ideal case and a base case. The ideal case is taken as the one in which no errors are introduced into the multiplications of the ray tracing algorithm. Correctly rounded multiplication results are used to generate the output in the ideal case. The image generated from the ideal case does not have any deliberately introduced errors and is called the ideal image. To find out the actual reduction in the quality of the image generated from the low power schemes, we compare all the generated images with the ideal image to have a fair comparison. The base case is considered to be the one in which only rounding errors are introduced in the multiplier. All the full adders are operated at the nominal technology supply voltage of 1.2V. The energy consumed in this base case (note this is **not** the ideal case) is considered to be 100% energy. Energy consumed by application of low power schemes is compared with respect to the base case.

Peak Signal-to-Noise Ratio (PSNR) is used to represent the quality of the generated image. The graphics models used in the ray tracing algorithm for the experiment are the Stanford Bunny, Stanford Dragon and Stanford Armadillo. The results were calculated from colored images of resolution 200x200. The image generated “looks good” when the PSNR value is

47dB or greater. The energy consumed using sleep, BIVOS and the proposed BIVOS + Sleep technique, for a fixed image quality, is shown in Table VII. As mentioned, the energy consumed is with respect to the base case. For the BIVOS + Sleep case at 38% energy, the voltage profile for the full adder columns is shown in Table VIII. A column value of 1 in Table VIII refers to the least significant full adder column. The voltage profile was selected by simulating different voltage values for different full adder columns in the C simulator and choosing the one which met the requirements of the output image quality. Images of the graphics model after ray tracing for the ideal case and BIVOS + Sleep at 38% energy are shown in Fig. 5. Hence, the results show that an energy savings of 62% can be achieved in a floating point multiplier by application of the proposed BIVOS + Sleep technique. The proposed BIVOS + Sleep technique at 38% energy can be operated at the same clock frequency as the base case because the increase in the delay due to lowering of voltages is compensated by a reduction in the critical path as large number of lower significant columns are operated in sleep mode.

TABLE VII. IMAGE QUALITY VERSUS ENERGY CONSUMPTION

PSNR(dB) Relative to the Ideal Image	Energy Relative to the Base Case		
	BIVOS	Sleep	BIVOS + SLEEP
58 dB	80%	75%	66%
47 dB	55%	51%	38%

TABLE VIII. VOLTAGE PROFILE FOR BIVOS+SLEEP AT 38% ENERGY

Voltage	Sleep	0.8V	0.9V	1.0V	1.1V	1.2V
Column	1-22	23-24	25-29	30-33	34-35	36-46

To calculate the cost of voltage shifters, used for storing the results to memory, for bits operated at supply voltage less than 1.2V (for 90nm technology), we simulated the voltage shifter presented in [13] in HSPICE. The total energy consumed by voltage shifters was 0.15% of the energy consumed by the 24-bit multiplier in the base case. The percentage increase in area, due to voltage shifters, was around 0.014%. Hence, the cost of including voltage shifters, on the overall energy consumption and area of the proposed multiplier design, is negligible.

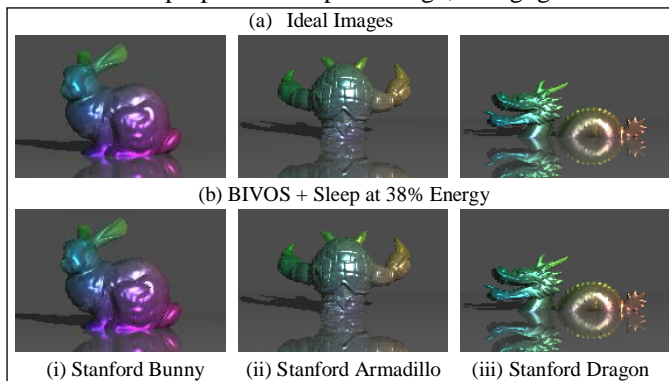


Figure 5. Images generated using Ray Tracing

VII. DISCUSSION OF RESULTS

One of the main reasons that this BIVOS + Sleep technique outperforms either the sleep or the BIVOS technique alone for a floating point multiplier is that, although the output of the 24-bit multiplication is 48 bits, the least significant 24 bits after normalization contribute merely to rounding and only the most significant 24 bits are stored as mantissa. Hence, one can afford large errors in the least significant columns which can be achieved by putting the full adders to sleep. This is where the

BIVOS scheme lags as BIVOS will still invest some amount of energy in the least significant columns. Moreover, with BIVOS, the critical path delay of the multiplier is greater than the clock period. The full adder columns just after the columns which are in sleep mode have a large amount of error because no calculation has been performed prior to that column. So, even if we operate the next few columns at the lowest available voltage, the new errors generated due to operation at a lower voltage are much less than the errors propagated from the previous columns in sleep mode. This is where the Sleep scheme lags as Sleep will operate the next columns at the highest voltage and will have almost the same error rate as the case when the columns are operated at a lower voltage.

VIII. CONCLUSION

The results show that with the proposed BIVOS + Sleep scheme, energy savings of around 62% can be achieved in a floating point multiplier. As the low power schemes are applied to 50% of the total multiplications of ray tracing algorithm, the overall application-level energy savings in the multiplications is equal to 31%. This is a significant improvement in energy savings achieved by the next best technique, sleep, for the same image quality. For our future work we plan to investigate alternative multiplier architectures such as Wallace trees.

REFERENCES

- [1] K. V. Palem, "Energy aware algorithm design via probabilistic computing: From algorithms and models to moore's law and novel (semiconductor) devices," Proceedings of the International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES'03), October 2003, pp. 113-116.
- [2] J. George, B. Marr, B. E. S. Akgul, and K. V. Palem, "Probabilistic arithmetic and energy efficient embedded signal processing," Proceedings of the 2006 International Conference on Compilers, Architecture and Synthesis for Embedded Systems, October 2006, pp. 158-168.
- [3] L. B. Kish, "End of Moore's law: Thermal (noise) death of integration in micro and nano electronics," Physics Letters A, 2002, vol. 305, no. 3-4, pp. 144-149.
- [4] J. Y. F. Tong, D. Nagle, Rob. A. Rutenbar, "Reducing power by optimizing the necessary precision/range of floating-point arithmetic," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, June 2000, vol.8, n.3, pp. 273-285.
- [5] M. J. Schulte, K. E. Wires and J. E. Stine, "Variable-Correction Truncated Floating Point Multipliers," Proceedings of the Thirty Fourth Asilomar Conference on Signals, Circuits and Systems, 2000, pp. 1344-1348.
- [6] K. E. Wires, M. J. Schulte and J. E. Stine, "Combined IEEE Compliant and Truncated Floating Point Multipliers for Reduced Power Dissipation," IEEE International Conference on Computer Design (ICCD), Austin, TX, September 2001, pp. 497-500.
- [7] L. N.B. Chakrapani, K. K. Muntimadugu, A. Lingamneni, J. George, K. V. Palem, "Highly Energy and Performance Efficient Embedded Computing Through Approximately Correct Arithmetic: A Mathematical Foundation and Preliminary Experimental Validation," Proceedings of CASES 2008, Atlanta, October 2008, pp. 187-196.
- [8] IEEE Standard for Floating-Point Arithmetic, IEEE Std 754-2008, pp. 1-58.
- [9] A. Singh, A. Basu, K.V. Ling and V. J. Mooney, "Modeling Multi-output Filtering Effects in CMOS," Proceedings of the VLSI Design and Test Conference (VLSI-DAT 2011), April 2011.
- [10] T. Whitted, "An improved illumination model for shaded display," Communications of the ACM, June 1980, v.23 n.6, pp. 343-349.
- [11] M. Lau, K. V. Ling, and Y. C. Chu, "Energy-Aware Probabilistic Multipliers: Design and Analysis," Proceedings of CASES 2009, October 2009, pp. 281-290.
- [12] Active Hdl 6.3 Manual, ALDEC (<http://www.aldec.com/activehdl/>)
- [13] Kyoung-Hoi Koo, Jin-Ho Seo, Myeong-Lyong Ko and Jae-Whui Kim, "A New Level-up Shifter for High Speed and Wide Range Interface in Ultra Deep Sub-Micron", IEEE International Symposium on Circuits and Systems, 2005, (ISCAS '05), vol. 2, pp. 1063- 1065, May 2005.