An Operand-Optimized Asynchronous IEEE 754 Double-Precision Floating-Point Adder

Basit Riaz Sheikh and Rajit Manohar
Computer Systems Laboratory
School of Electrical and Computer Engineering
Cornell University
Ithaca, NY 14853, U.S.A.
{basit,rajit}@csl.cornell.edu

***Abstract*—We present the design and implementation of an asynchronous high-performance IEEE 754 compliant double-precision floating-point adder (FPA). We provide a detailed breakdown of the power consumption of the FPA datapath, and use it to motivate a number of different data-dependent optimizations for energy-efficiency. Our baseline asynchronous FPA has a throughput of 2.15 GHz while consuming 69.3 pJ per operation in a 65nm bulk process. For the same set of nonzero operands, our optimizations improve the FPA's energy-efficiency to 30.2 pJ per operation while preserving average throughput, a 56.7% reduction in energy relative to the baseline design. To our knowledge, this is the first detailed design of a high-performance asynchronous double-precision floating-point adder.**

***Keywords*-Floating point arithmetic; asynchronous logic circuits; very-large-scale integration; pipeline processing**

I. Introduction

Efficient floating-point computation is important for a wide range of applications in science and engineering. Using computational techniques for conducting both theoretical and experimental research has become ubiquitous, and there is an insatiable demand for higher and higher performing VLSI systems. Today, this performance is limited by power constraints. The Top 500 supercomputer ranking now includes the energy-efficiency of the system as well as its performance. At the other end of the spectrum, embedded systems that have traditionally been considered low performance are demanding higher and higher throughput for the same power budget. Hence it is important that we develop *energy-efficient* floating-point hardware, not simply high performance floating-point hardware.

The IEEE 754 standard [1] for binary floating-point arithmetic provides a precise specification of a floating-point adder (FPA). This specification was determined after much debate, and it took several years before hardware vendors developed IEEE-compliant hardware. Part of the challenge was the belief that: (i) implementing most of the standard was sufficient; (ii) ignoring a few infrequently occuring cases led to more efficient hardware (e.g. [2]). Unfortunately ignoring certain aspects of the standard can lead to unexpected consequences in the context of numerical algorithms. Today, most floating-point hardware is IEEE-compliant or has an IEEE-compliant mode.

The observation that there are infrequently occuring cases that make the hardware difficult/slow leads to the natural question: can we design an energy-efficient *asynchronous* floating-point adder? An asynchronous circuit does not use a clock signal, and is not constrained to a global timing constraint. Perhaps we could design an IEEE-compliant floating-point adder that was a bit slower when certain infrequent cases occured. This could result in a significant energy reduction during normal operation. Self-timing would enable this flexibility at a very fine grain, allowing for operand-dependent performance.

We begin with a baseline asynchronous FPA that corresponds to a state-of-the-art high performance synchronous FPA design. We provide energy-consumption breakdown of a high-performance asynchronous FPA datapath, and use this to guide our optimizations for energy-efficiency. We present our operand-dependent optimization techniques to reduce the energy per operation of asynchronous floating-point addition, including some that result in poor throughput in pathological cases. It is these optimizations that are challenging in the synchronous context, because they increase the worst-case critical path making the common case slower even though on average they have negligible impact on throughput.

All our performance and energy evaluations use transistor-level simulation with estimated wire loads. We have found that our wire load estimates are conservative, and predicted energy and delay numbers have been about 10% higher than post-layout simulations (that include accurate parasitics) for a range of previous designs. In a 65nm bulk CMOS process (TT, 25°C, 1V), the baseline asynchronous FPA operates at a throughput of 2.15 GHz while consuming 69.3 pJ/op. With the same operand inputs, our optimized asynchronous FPA consumes only 30.2 pJ/op—a 56.7% reduction in energy relative to the baseline asynchronous FPA while preserving the average throughput.

II. Background and Related Work

A floating-point adder is used for the two most frequent floating-point operations: addition and subtraction. It requires much more circuitry to compute the correctly normalized and rounded sum compared to a simple integer adder. All the additional circuitry makes the FPA a complex, power-consuming structure. The following summarizes the key operations required to implement an IEEE-compliant FPA:

- The first step in the FPA datapath is to unpack the IEEE representation and analyze the sign, exponent, and significands bits of each input to determine if the inputs are standard normalized or are of one of the special types (NaN, Infinity, Denormal).
- The absolute difference of the two exponents is used as the shift amount for a variable right shifter which aligns the smaller of the operands.
- In parallel with the right align shifter, the guard, round, and sticky bits are computed to be used for rounding in latter stages of the FPA datapath.
- The next step is the addition or subtraction of two significands based on sign information.
- Most high-performance FPAs use a special-purpose circuit popularly known as a Leading-One-Predictor and Decoder (LOP/LOD) to predict the position of the leading one in parallel with the addition/subtraction step.
- The post addition steps include normalizing the significands. This may require either a left shift by a variable amount (using the predicted value from LOP), no shift (if the output is already normalized), or a right shift by one bit (in case of carry-out when the addition inputs have the same sign).
- The exponent is adjusted based on the shift amount during normalization. In parallel, the guard, round, and sticky bits are updated and are used, along with the rounding mode, to compute if any rounding is necessary. The sign of the sum is also computed.
- In case of rounding, the exponent and significand bits are updated appropriately.
- The final stage checks for a NaN, Infinity, or a Denormal outcome before outputting the correct result.

A. Asynchronous Arithmetic

The use of asynchrony to improve the performance of arithmetic circuits has been exploited by a number of different researchers. As early as 1946, von Neumann proposed using an asynchronous integer adder because the average-case delay for a ripple-carry adder is $O(\log N)$ where N is the number of bits in the input assuming that the input bits are independent, identically distributed (i.i.d.) random variables [7]. More recently it was shown that it is possible to design an asynchronous integer adder with an average-case latency of $O(\log \log N)$ for i.i.d. inputs [8] and that the design achieves the optimal asymptotic average-case latency for any input distribution [9]. There have been numerous papers on asynchronous adders with a variety of topologies (e.g. [5,10–12]).

To our knowledge, the work of Joel Noche et al. [13] is the only published work on FPA design using asynchronous circuits. Their work claims a full working single-precision floating-point unit (FPU). However, their FPU is completely non-pipelined, doesn't include any energy optimization techniques, and does not implement rounding logic. Their test vector included one addition of two arbitrary single-precision floating-point inputs for which they claim a completion time (latency) of 79 nanoseconds in a 0.35μm process at 3.3V.

B. Synchronous Floating-Point Adders

There is a large body of work on synchronous FPA design. Ercegovac and Lang [6] contains an overview of the different techniques used to optimize floating-point addition. Most of the earlier work on the FPA design has focused on improving FPA latency [17,19–21]. Oberman [21] proposes the use of two align shifters to improve the latency of their single-precision FPA with only one rounding mode. Seidel and Even [17] propose a two-path FPA design to reduce overall latency. The R-path in their design deals with cases of effective addition (or subtraction with exponent difference greater than 1) and N-path deals with effective subtraction with exponent difference less than or equal to 1. Both paths are in operation at the same time and use their own significand adders.

There is less work on low-power FPAs compared to low-latency FPA design. Pillai et al. [23] propose the partitioning of the floating-point datapath into three distinct, clock-gated datapaths for activity reduction. Only one of the three paths is active during any operational cycle in their FPA. In our proposed transistor-level optimized asynchronous FPA, we also use control-inhibited pipelines but instead of using clock-gating to turn off the pipelines (which may worsen clock skew especially for high performance FPAs in deep submicron technologies) we use local asynchronous conditional split pipelines which have no effect on overall throughput. Also, our design goes beyond pipeline inhibitions as explained in sections IV and V. The FPA design by Quinnell et al. [15] is one of the rare fully-implemented designs (65nm SOI) from academia. Although, they use standard-cell library as opposed to our custom transistor-level construction, their work provides us with a good baseline to analyze our throughput and power results.

Recent years have seen a number of contributions in the design of Fused-Multiply-Add (FMA) units [14–16,18]. In [16], the authors propose techniques to reduce the latency of a floating-point addition operation in an FMA. In terms of performance and power-efficiency, the P6 Binary Floating-Point Unit [14] represents the state-of-the-art. It supports an extremely aggressive cycle time of 13FO4s. Power saving is done by clock-gating pipeline stages not in use. Power simulations at 1.1V, 4GHz, and 100% utilization in a 65nm SOI process consumed $310mW$.

III. A Baseline Asynchronous FPA

To our knowledge, our baseline unit is the first fully-implemented (at the transistor-level) asynchronous double-precision floating-point adder of its kind. It supports all four rounding modes and is fully IEEE-754 compliant. Fig. 1 shows the block diagram of our FPA datapath, which is loosely based on recent high-performance FPA/FMAs. It uses standard state-of-the-art techniques such as leading one prediction and decoding, use of parallel prefix tree adder, and fast logarithmic

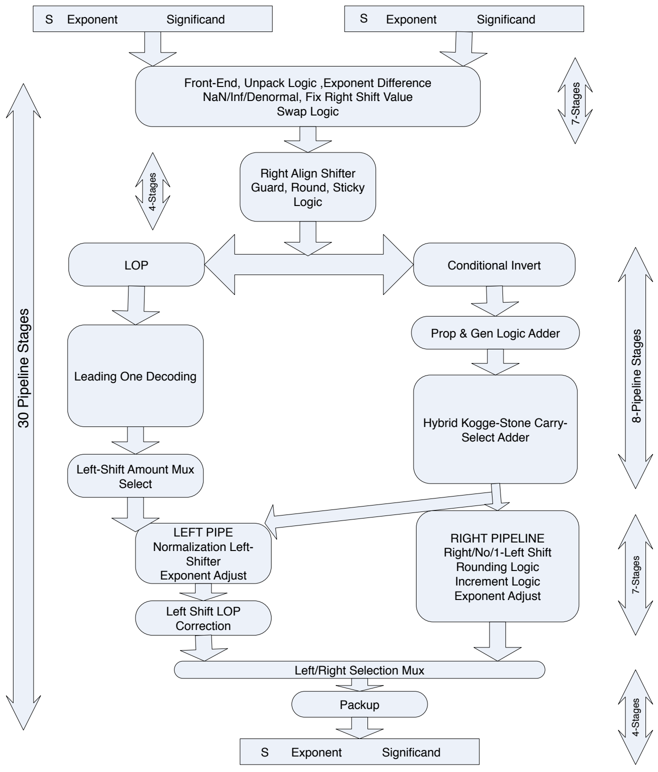
S Exponent Significand
S Exponent Significand
Front-End, Unpack Logic ,Exponent Difference NaN/Inf/Denormal, Fix Right Shift Value Swap Logic
7-Stages
Right Align Shifter Guard, Round, Sticky Logic
4-Stages
LOP
Conditional Invert
Prop & Gen Logic Adder
Leading One Decoding
Hybrid Kogge-Stone Carry-Select Adder
8-Pipeline Stages
30 Pipeline Stages
Left-Shift Amount Mux Select
LEFT PIPE Normalization Left-Shifter Exponent Adjust
RIGHT PIPELINE Right/No/1-Left Shift Rounding Logic Increment Logic Exponent Adjust
7-Stages
Left Shift LOP Correction
Left/Right Selection Mux
Packup
4-Stages
S Exponent Significand

Fig. 1. Asynchronous Baseline FPA Architecture

shifters to keep the throughput high. To reduce latency and overall complexity, the post-addition normalization datapath is separated in two paths. The *Left* path contains a variable left-shifter, whereas the *Right* path includes a single-position right or left shifter along with all rounding and increment logic. We equally weighed performance and power trade-offs in the choice of our circuits for various functional blocks of the FPA. The following subsections explain our choice of asynchronous pipelines, 56-bit significand adder and LOP/LOD functional block.

A. Fine-grain Asynchronous Pipelining

We use quasi-delay-insensitive (QDI) asynchronous circuits for our FPA design. Our baseline asynchronous FPA's datapath is highly pipelined (thirty pipeline stages) to maximize throughput. Unlike the standard synchronous pipelines, the forward latency of each asynchronous pipeline is only two logic transitions (the pull-down stack followed by the inverter), hence the thirty stage asynchronous pipeline depth results in acceptable FPA latency. The fine-grain asynchronous pipelines in our design contain only a small amount of logic (e.g. a two-bit full-adder). The actual computation is combined with data latching, which removes the overhead of explicit output registers. This pipeline style has been used in previous high-performance asynchronous designs, including a fully-implemented and fabricated asynchronous microprocessor [5].

We use pre-charge enable half-buffer (PCEHB) pipeline for all data computation [3]. It is a modified version of the original PCHB pipeline [4]. Our SPICE simulations show PCEHB pipelines to be faster and more energy-efficient than PCHB pipelines in a modern 65nm process. For simple buffers and copy tokens, we use a weak-conditioned half-buffer (WCHB) [4] pipeline stage, which is much smaller circuit than a PCEHB and hence is more energy-efficient for simple data buffering and copy operations.

B. Hybrid Kogge-Stone Carry-Select Adder

The 56-bit significand adder is on the critical path of the FPA and is the single largest functional block in the FPA datapath. Improvements in the adder design usually have the largest overall impact on the FPA, hence designers spend considerable time in optimizing their adder circuits for performance and power. Parallel prefix logic networks that use tree structures to compute the carry are usually preferred for any adder with a large number of input bits. Tree adders like Kogge-Stone [24], Brent-Kung [30], and Sklansky [30] can compute any N-bit sum with a worst-case latency of $O(\log N)$ stages. Many commercial chips use some form of these tree adders in their FPA implementations.

Our baseline asynchronous FPA uses a hybrid Kogge-Stone/carry-select adder. The rationale for this choice is that most high-performance floating-point adders use this topology. The adder is partitioned into eight-bit Kogge-Stone blocks that compute two speculative sum outputs (assuming the carry-in is either zero or one). The sum output is selected by the final stage based on the actual carry values. The choice of eight-bit Kogge-Stone sub-blocks was made for energy-efficiency as blocks with more bits would have resulted in higher energy due to long wiring tracks that have to run across the total width of the block. Most blocks in the adder use radix-4 arithmetic and 1of4 codes (like the adder in [5]) to minimize energy and latency.

Subtraction is done in the usual way by inverting the inputs and using a carry-in of one for the entire adder. The choice of significand to invert is important from the energy perspective. Since IEEE floating-point uses a sign-magnitude representation, a final negative result requires a second two's complement step. To avoid this, our asynchronous FPA always chooses to invert the smaller of the two significands.

C. Leading One Prediction and Decoding

Most modern FPA implementations use LOP/LOD logic to determine the shift amount for normalization in parallel with the significand adder. This reduces the latency of the FPA, because the shift amount is ready when the adder outputs are available.

Our LOP logic is inspired from the LOP scheme proposed by Bruguera et al. [22]. It subtracts the two significands using a signed digit representation producing either a 0, 1, or -1 for each bit location. The bit string of 0s, 1s, and -1s can be used to find the location of the leading one, except that it could be off by one in some cases. Instead of using a correction scheme that operates in parallel with the LOP hardware (requiring significant more energy), we use the speculative shift amount and then optionally shift the final outcome by one in case there was an error in the estimated shift amount. This also requires an adjustment to the exponent. To make this adjustment efficient, both values of the exponent are computed concurrently by using a dual-carry chain topology for the exponent adder.

D. Evaluation of Baseline Asynchronous FPA

We use a 65nm bulk CMOS process at the typical-typical (TT) corner. The steady state throughput and energy per operation results for our baseline asynchronous FPA with highest-precision HSIM/HSPICE simulation configuration are shown in Fig. 2. The different data points correspond to different supply voltages (0.6V and 1.1V). We added additional wire load in the SPICE file for every gate in the circuit.

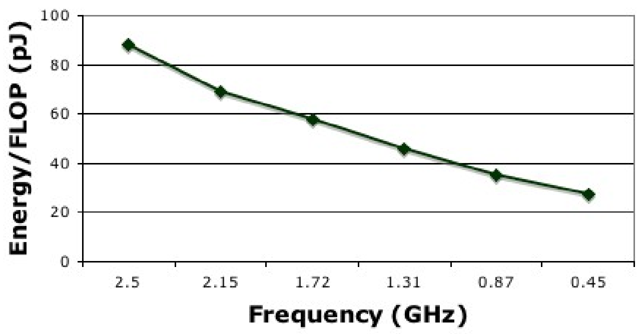

Fig. 2. Baseline FPA Energy vs Throughput

At a V_{DD} of 1V, the FPA operates at a throughput of 2.15 GHz with an average power dissipation of $149mW$, an energy/operation of 69.3 pJ/op. The power values include the gate and sub-threshold leakage power. Compared to the standard-cell library FPA in a 65nm SOI process by Quinnell et al. [15] operating at a throughput of 666 MHz with an average power-consumption of $118mW$, our baseline FPA design operating at 3.2 times higher throughput consumes 2.6 times less energy per operation even though we are using a bulk process.

E. Power Breakdown and Analysis

The last decade witnessed a significant change in the focus of arithmetic circuit designers from purely performance oriented high-speed circuits to energy-efficient circuit implementations. To improve the efficiency of any VLSI system, it is critical to first understand where energy and power are dissipated. We have not found a detailed energy/power breakdown of a state-of-the-art FPA datapath in the open literature.

Fig. 3 shows a detailed energy/power breakdown of our FPA datapath. Starting with 11% of Front-End and proceeding in the clock-wise direction, the energy/power contributions are in the same order as listed in the legend in the figure. Since in asynchronous PCEHB and WCHB pipelines the actual computation is folded and coupled into the pipelines, the percentage power usage of any particular functional block includes all pipeline overhead i.e. input validity, output validity and handshake acknowledge computation. Although, the Hybrid Kogge-Stone Carry-Select Adder is the largest power-consuming functional block in the pipeline, it is interesting to note that there is no single dominant high-power component in the FPA datapath. Hence, any effective power-saving optimizations would require us to tackle more than one function block.

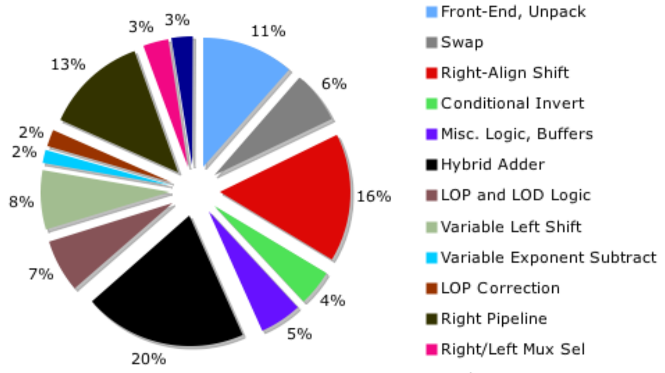

Fig. 3. FPA Pipeline Power Breakdown

The *Right-Align Shift* block which comes second in terms of power-consumption includes logic to compute the guard, round, and sticky bits to be used in the rounding mode. In the worst case, the sticky bit logic has to look at all 53 shifted out bits. To do this fast and in parallel with the right-align shifter, considerable extra circuitry is needed which consumes more power. The post addition *Right Pipeline* block is the third most power-consuming component of the FPA datapath. It includes the single position left or right shifter as well as complete rounding logic which includes significand increment logic and exponent increment/decrement logic blocks.

IV. Coarse Grain Power Reduction

Most synchronous FPAs (limited by worst-case computation delay) include complex circuitry to attain constant latency and throughput for the best, average, and worst case input patterns, although the best and average additions could have been done much faster and more efficiently. The important question to ask is how often the worst-case happens. If it happens very frequently then it justifies burning extra power with complex circuits to boost overall performance.

To answer this question, we used Intel's PIN [25] toolkit to profile input operands in a few floating-point intensive applications from SPEC2006 [26] and PARSEC [27] benchmark suites using reference input sets. The set of ten applications we chose for profiling came from very diverse fields such as quantum chemistry, speech recognition, financial services, molecular biology, 3D graphics, linear programming optimizations etc. The input operands in actual benchmark runs were saved to disk, and then used for statistical analysis. The application profiling statistics in the following sections were tabulated using ten billion input operands for each application.

A. *Interleaved Asynchronous Adder*

The delay of an N-bit adder primarily depends on how fast the carry reaches each bit position. In the worst-case, the carry may need to be propagated through all bits, hence synchronous implementations resort to tree adder topologies. However, as shown in Fig. 4, for most application benchmarks, almost 90% of the time the maximum carry-chain length is limited to 7 radix-4 positions.

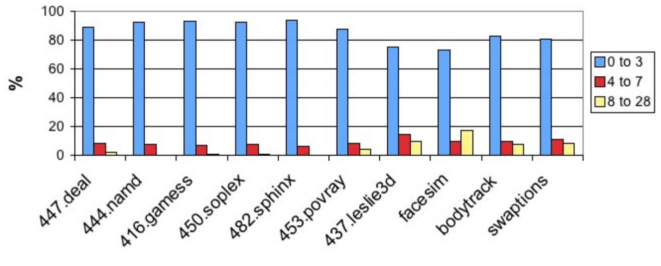

Fig. 4. Radix-4 Ripple-Adder Carry-Length

An N-bit ripple carry asynchronous adder has an average case delay of $O(\log N)$, the same order as a more complex synchronous parallel-prefix tree adder such as Kogge-Stone. However, the use of ripple-carry asynchronous adders is not feasible for high-performance FPA circuits because the pipeline stage waiting for the carry input stalls the previous pipeline stage until it computes the sum and the carry-out. Even a delay of one carry-propagation (which is two gate delays) stalls the preceding pipeline by a significant amount.

To circumvent the average throughput problem, we use an *interleaved* asynchronous adder as shown in Fig. 5. It uses two radix-4 ripple-carry adders: the *left* and *right* adders. Odd operand pairs are summed by the *right* adder, and even operand pairs are summed by the *left* adder. The notion of interleaving blocks has been used for a number of different structures in the past, including FIFOs [31] and high-speed communication circuits [32].

In a standard PCEHB reshuffling, the interleave stage has to wait for the acknowledge signal from ripple-stage before it can enter neutral stage and accept new tokens. However, this would cause the pipeline to stall in case of a longer carry chain. Hence, we do not use PCEHB reshuffling in our adder topology. Instead of waiting for the output acknowledge signals from the *right* ripple-carry adder, the interleave stage checks to see if the *left* ripple-carry adder is available. If it is, the interleave stage asks for new tokens from the previous pipeline stage and forwards the arriving tokens to the *left* adder. The two ripple-carry adders could be in operation at the same time on different input operands. Since our pipeline

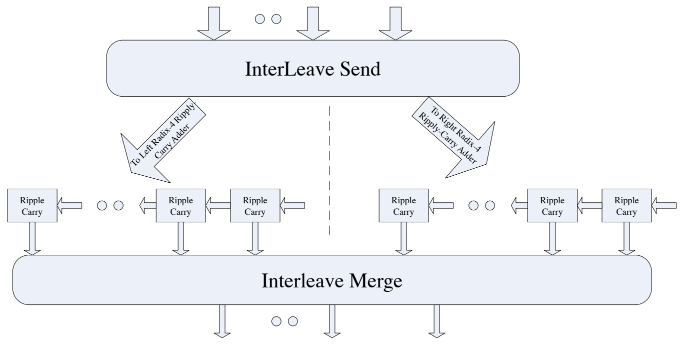

Fig. 5. Interleaved Asynchronous Adder

cycle time is approximately 18 logic transitions (gate delays), the next data tokens for the *right* adder are scheduled to arrive after 36 transitions of the first one. This gives ample time for even very long carry-chains to ripple through without causing any throughput stalls.

Table I shows the throughput results of our *interleaved* asynchronous adder using SPICE simulations with different input sets. Compared to the 56-bit Hybrid Kogge-Stone Carry-Select Adder which gave a throughput of 2.17 GHz and energy/operation of $13.6pJ$ when simulated by itself, the *interleaved* adder operates at an average throughput of 2.2 GHz for input cases with carry-length of fourteen or less while consuming only $2.9pJ$ per operation. Not only it reduces the energy/operation by more than 4X, it also reduces the number of transistors in the 56-bit adder by 35%.

TABLE I
THROUGHPUT ACROSS DIFFERENT CARRY LENGTHS

Input	0-3	4-7	8-14	15-20	27	Frequency
Deal	88%	9%	2.7%	0.3%	0%	2.2 GHz
I	0%	100%	0%	0%	0%	2.2 GHz
II	0%	0%	100%	0%	0%	2.2 GHz
III	0%	0%	0%	100%	0%	1.38 GHz
IV	0%	0%	0%	0%	100%	0.78 GHz

Deal corresponds to operand data from *447.deal* SPECFP 2006 application benchmark. Other applications from the SPECFP suite had similar statistics, so we simply picked one representative benchmark for comparison. The synthetic input sets (I to IV) are designed to have specific carry chain lengths, as can be seen from the statistics in Table I. The synthetic input sets III and IV generate input operands for the adder that yield fixed maximum carry-chain lengths of 20 and 27 (maximum for radix-4 56-bit addition) respectively. We did observe a dip in throughput for these two input sets, but since our statistical analysis reported earlier in the section show the probability of such high carry-chain lengths to be quite rare, it is feasible to take a throughput penalty for such rare occurrences (0.5% or less) in order to save more than four times the energy per operation for the 99.5% of input patterns with maximum carry-chain length of

14 or less.

B. Left or Right Pipeline

In our baseline asynchronous FPA, the post-addition datapath is divided into two separate pipelines: *Right* pipeline and *Left* normalize pipeline as shown in Fig. 1. The two pipelines handle disjoint cases that could occur during floating-point addition. The *Left* normalize pipeline handles cases when destructive cancellation can occur during floating-point addition, requiring a large left shift for normalization. The destructive cancellation scenario happens only when the exponent difference is less than two, and the FPA is subtracting the two operands. The *Right* pipeline handles all other cases.

Instead of activating both pipelines and selecting the result, we compute the selection condition early (prior to activating the LOP/LOD stage) and then only conditionally activate the appropriate path through the floating-point adder. The LOP/LOD function blocks determine the shift value for the left normalization shifter. The shift amount determined by LOP/LOD is only needed in cases which could potentially result in destructive cancellation. Hence, in the case of *Right* pipeline utilization, we also save energy associated with the LOP/LOD stage, because the results of the LOP/LOD are only used by the *Left* normalize pipeline. Compared to the baseline FPA, we get power savings of 13% for operands using the *Left* pipeline and power savings of up to 18% (11% *Left* pipe & 7% LOP/LOD) for operands using the *Right* pipeline which is the more frequent case as shown in Fig. 6.

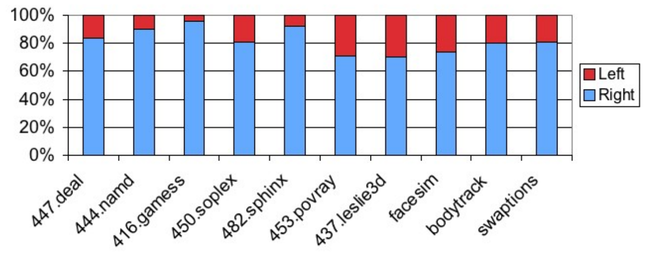

Fig. 6. Left/Right Pipeline Frequency

V. Operand-Based Optimizations

This section further improves the energy-efficiency of the FPA by examining other properties of the input operand distribution. We optimize four additional aspects of the FPA pipeline: (i) initial right align shifter; (ii) leading one prediction; (iii) post-addition increment; (iv) zero input operands.

A. Two-Way Right-Align Shift

The *Right-Align Shift* block is the second-most power consuming structure in the baseline FPA. It includes the right shifter logic as well as the logic to compute the guard, round, and sticky bits used for rounding. The sticky bit is set to one if any of the shifted out bits from the alignment shift stage is one; otherwise it is set to zero. In the worst case, the sticky bit logic has to examine all 53 shifted bits. To do this fast and in parallel with the right-align shifter, considerable extra circuitry is needed which consumes more power. For high throughput, the other (non-shifted) significand is slack-matched to the right-align shift logic using a number of WCHB pipeline stages. The *Right-Align Shift* block also compares the two significands to determine which of the two significands should be inverted in case of subtraction. The exponent difference and sign bit is used to generate enable control for the LOP. Each control bit is shared for two (one for each operand) radix-4 significand entries. Overall, this comparison of significands and generation of large number of control bits is not cheap in terms of power consumption.

The shifter comprises of three pipeline stages. The first stage shifts the significand between 0 to 3 bit positions based on the shift-control input. The second pipeline shifts by 0, 4, 8, or 12 bit positions and the third stage shifts by 0, 16, 32, or 48 bit positions using the shift-control input signals for the respective stages. Each radix-4 significand entry shift pipeline resembles a PCEHB template with a 4-to-1 multiplexor as the pull-down logic. Each stage produces multiple output copies to feed into 4 different PCEHB multiplexor blocks of the following pipeline stage. All this circuitry makes the shifter a costly structure in our FPA datapath.

The key advantage of the shifter topology is its fixed latency for any shift value ranging between 0 and 55 (the maximum align shift in a double-precion addition/subtraction). This advantage is also one of its drawbacks as it consumes the same power to do a shift by zero and a shift by a large value. Fig. 7 shows the right align shift patterns across 10 different benchmarks using operands gathered through PIN application profiling. Although, these benchmark applications are from totally unrelated disciplines, they exhibit a common property: a significant proportion of right align shift values range between 0 to 3 inclusive. For one benchmark, the proportion of right align shifts of 0 to 3 is almost 81%.

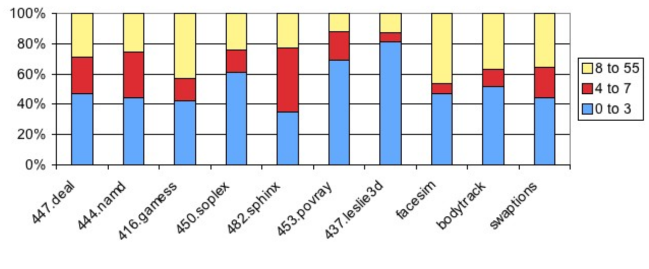

Fig. 7. Right Align Shifter Statistics

In our baseline right-align shift topology, shifts by 0 to 3 are done in the first pipeline stage. However, in spite of that the significand still needlessly goes through the other two shift stages and in doing so wastes considerable power. It would have been an acceptable trade-off if most operations required align shifts by a large value, but the shift patterns shown in Fig. 7 make it evident that our baseline align shifter topology is highly non-optimum from an energy perspective.

To improve the energy-efficiency of the align shifter, we split it into two paths. The first stage dealing with a right shift of 0 to 3 is shared between two paths. In case of a shift greater than 3 bit positions, the significand is forwarded to the second shift pipeline stage as in the original topology. However, for shifts of 0 to 3 bit positions, the significand output is bypassed to the post align-shifter pipeline stage as shown in Fig. 8. The post align-shift stage consists of a merge pipeline which receives inputs from both the regular shift path and the short bypass shift path. It selects the correct input using the buffered control signal which was earlier used to direct the significand to one of the two paths.The short shift path has multiple features which lead to significant power savings:

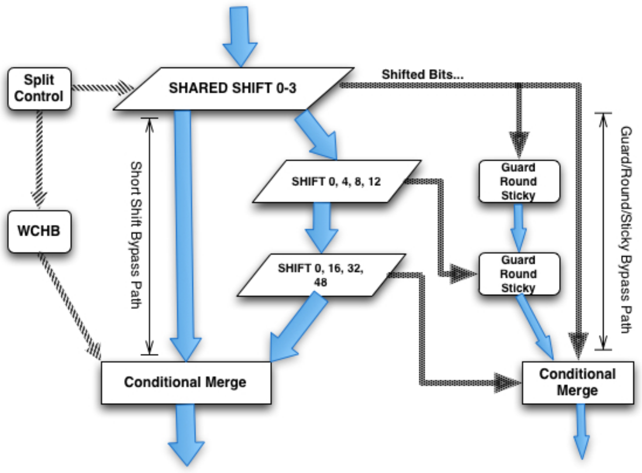

Fig. 8. Two-Path Right-Align Shift

- The shifted significand skips the remaining two shift pipelines.
- In contrast to the baseline topology which produces multiple significand outputs to be consumed in the following shift stages, the bypass shift path needs only one output for each significand.
- The guard, round, and sticky computation becomes quite simple and requires minimal energy as only a maximum of 3 bits are shifted out.
- The other (non-shifted) significand also bypasses the WCHB slack-matching buffers.
- No shift select signals need to be generated and copied for the second and third shift pipeline stages.

The new shifter topology poses a design choice of slack-matching the control to either the long-shift path with two pipeline stages or the short-bypass path with no pipeline buffering at all. If control is slack-matched to the short path, the shifts requiring long path may suffer from stalls and degrade the FPA throughput. Slack-matching the control to the long path increases the short path latency. The worst-case scenario is when the pipeline alternates between the two paths. However, our application profiling analysis in Fig. 9 reveal that across all application benchmarks, the proportion of times a short path shift follows another shift along the same path is considerably high. We saw similar results for the long path shifts. A detailed throughput and latency analysis, based on the profiled shift patterns, favored a control path which is slack-matched to none of the two shift paths. In our implementation, the merge control input has only one WCHB pipeline and has a throughput within 1.3% of the baseline FPA in the worst-case scenario.

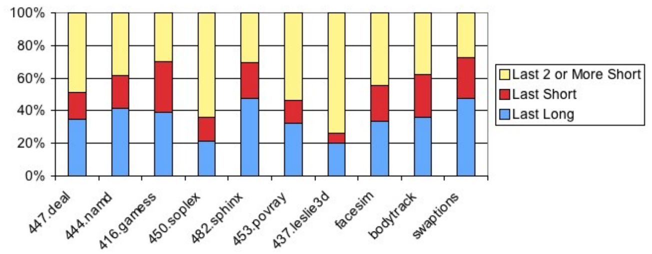

Fig. 9. Right Align Shift Short Path Pattern

B. Minimizing LOP Logic

For subtraction, the bits of the shifted significand are inverted except when the exponent difference is zero which then requires input from the significand comparison block to determine which one of the two significands is smaller. Since the case of exponent difference of zero corresponds to the bypass shift path, the significand comparison logic requiring multiple pipeline stages cannot be done in parallel with the bypass path without incurring a throughput penalty. Hence, the significand comparison is moved to earlier pipeline stages in the optimized FPA datapath.

With the result of significand comparison available early, the LOP logic stack can be simplified. As Bruguera et al. point out in [22], the logic to predict leading one when the sign-digit difference of two operands is positive is different from the case when the sign-digit difference of two operands is negative. In our optimized FPA, using the significand comparison result early in the FPA enables the LOP computation to assume that its first operand always corresponds to the larger significand. This information enables us to significantly reduce the circuitry required for LOP computation.

In the baseline FPA, there is a separate pipeline stage to conditionally invert bits in case of subtraction. The baseline FPA generates control signals for each radix-4 position specifying which of the two significands if any need to be inverted. Since the LOP control bits in our optimized FPA already contain information about the larger significand, we merged the conditional invert stage with pre-LOP selection pipeline which determines the larger of the two significands as LOP's first operand. This eliminates the need of separate control signals for inverting bits and including savings from cutting a full pipeline stage leads to energy reduction of over 3%.

C. *Post-Add Right Pipeline*

The *Right Pipeline* block is the third most power-consuming structure in the baseline FPA. It includes a single-position right or left shifter, a 53-bit significand incrementer, rounding logic, and final exponent computation block for operands utilizing the *Right Pipeline*. As shown earlier in Fig. 6, on average more than 80% of the FPA operations use this block. Hence, power-optimization techniques for the circuits in this block have a notable impact on average FPA power savings.

The baseline carry-select incrementer comprises of four-bit blocks with each computing the output for the carry input of one into that block. In parallel, there is a fast carry-logic which computes the correct carry-input for each four-bit block. Lastly, there is a mux pipeline stage which selects either the incremented output or the buffered non-incremented bits for each four-bit block using the carry select input. In case of a carry-out of one, the significand is right shifted by one bit position.

The key advantage of our baseline incrementer topology is its fixed latency for the best (no carry propagation) and worst-case (carry propagates through all the bits) alike. However, as seen in Fig. 10, for over 90% of the operations using the increment logic, the carry propagation length is less than four radix-4 bit positions. Also, the case of a final carry-out occurs no more than 0.5% of the time.

The carry-select incrementer targeted for worst-case scenarios is a non-optimum choice for the average-case incrementer carry-length patterns. To improve energy-efficiency, we instead use an interleaved incrementer similar to earlier described interleaved adder. Instead of using two ripple-carry adders, it uses much simpler two radix-4 ripple-carry incrementers. The odd data token is forwarded to the *right* incrementer. For the next arriving data token, the interleave stage checks to see if the *left* incrementer is available. If it is, the interleave stage forwards the arriving tokens to it. The interleave merge stage receives the inputs from both incrementers and forwards those to the next pipeline stage in the same interleaved order in which they were scheduled. This allows the two incrementers to be in operation at the same time on different input operands.

The incrementer is used to adjust the result due to rounding. Our new incrementer topology computes either the correct incremented or non-incremented output (not both) using the round-up bit as the carry-in, hence alleviating the need to have a separate mux stage to choose between two possible outputs. Our simulation results for the new topology show no throughput penalty for average-case inputs. Also, there is no need for a separate post-increment right shift pipeline stage. The case where the final result must be right shifted by one only occurs when all significand bits are one, and the result must be rounded up. In that scenario, the incrementer output is all zero and hence both shifted and unshifted versions of the incrementer result are identical. Hence, for correct output, only the most significant bit needs to be set to one.

In the baseline FPA, until the incrementer *carry-out* is computed the correct exponent value cannot be computed.

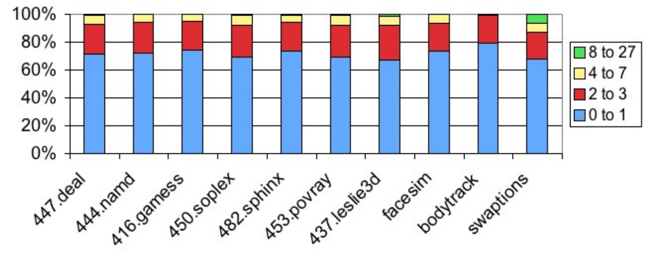

Fig. 10. Radix-4 Incrementer Carry Length

Since the carry-out is not available until the fourth pipeline stage in the *Right Pipeline* block, to prevent latency penalty the exponent values of *exponent*+C are always computed for $C = 0, \pm 1, +2$, with a mux stage choosing the correct output. To circumvent the problem of latency penalty, we replace the exponent computation block with an interleaved incrementer/decrementer which mitigates any latency degradation with its average-case behavior. It uses a two bit carry-in (first bit is set to 1 for increment, second bit is 1 for decrement, and both bits are 0 for a simple pass through) to compute *exponent*. Using dual-carry chain, *exponent + 1* is also computed simultaneously to be selected in case of a carry out. Overall, this computation of two exponent values is far more energy-efficient than the baseline.

D. *Zero-input Operands*

Fig. 11 shows that a few application benchmarks have a significant proportion of zero input operands. For the applications involving sparse-matrix manipulations such as *Deal* and *Soplex*, in spite of the use of specialized sparse-matrix computation libraries, the percentage of zero inputs can be as high as 36%. For other benchmarks, the zero-input percentage varies widely. In our baseline FPA and almost all synchronous FPA designs, operations involving zero-input operands use the full FPA datapath. Although, if one or both of the FPA operands are zero, the final FPA output could be computed without needing power-consuming computational blocks such as right-align shifter, significand adder, LOP/LOD, post-add normalization, and rounding.

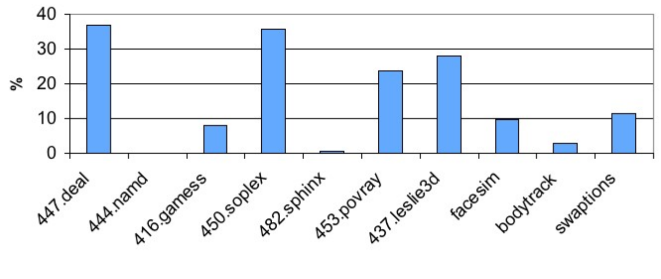

Fig. 11. Zero-input Operands

Since the *Unpack* pipeline stage already checks to see if any operand is zero, our optimized FPA utilizes the zero flag to inhibit the flow of tokens into the regular datapath. The zero flag is used as a control in the conditional split pipeline just prior to *Swap* stage to bypass the final sign, exponent, and significand bits to the last pipeline stage in case of a zero input. The last stage is replaced with a conditional merge pipeline which uses the buffered control signal to choose the input from either the zero bypass path or the regular FPA datapath. The huge slack disparity between two split pipelines makes the choice of control slack a critical one.

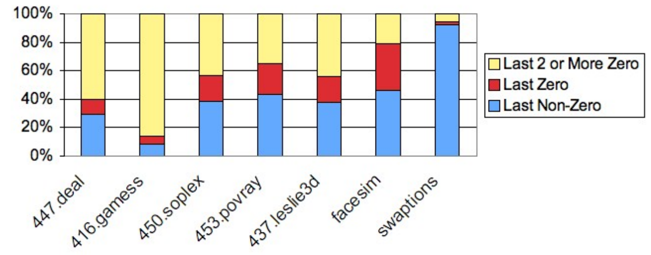

Fig. 12. Zero-input Pattern

Fig. 12 shows that for benchmark applications with significant proportion of zero inputs, the percentage of a zero-input followed by another zero-input operation is quite high except for the *Swaptions* benchmark. To choose the optimum level of control buffering, we simulated the optimized FPA with a number of synthetic input-patterns over a wide-range of control slack possibilities as seen in Fig. 13. *Mix-flip* refers to input sequence with alternating zero-input and nonzero-input operands. *Mix-pattern* sequence closely resembles the zero-input pattern seen in most benchmark applications. Based on these results, we chose to buffer the control with eight WCHB pipeline stages.

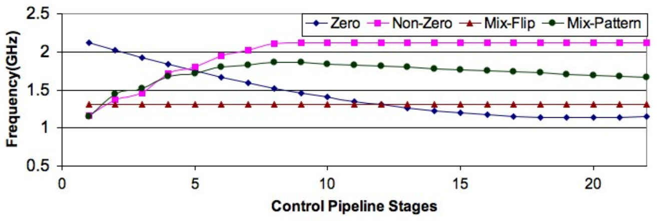

Fig. 13. Zero-Path Control Slack Analysis

Some zero-input patterns take a significant throughput hit even with eight WCHB pipeline stages for the control. To circumvent this problem, we explored the effect of adding some slack on the bypass path. Fig. 14 shows that the addition of two WCHB stages on the bypass path for sign, exponent, and significand bits greatly alleviates the throughput penalty albeit at a small cost in energy. Overall, the best throughput results are again attained with a slack of eight WCHB stages on the control. For *Mix-pattern* sequence, the throughput increases by 7.5% to 2 GHz. For the worst-case input set, *Mix-flip*, throughput increases by 49.8% to 1.95 GHz. The improvement in throughput comes at a cost of extra WCHB logic and hence more power. Our simulations using only one WCHB stage didn't show such profound throughput improvement and for cases beyond two WCHB pipeline stages, the small increases in throughput are overshadowed by power consumed in additional buffer stages.

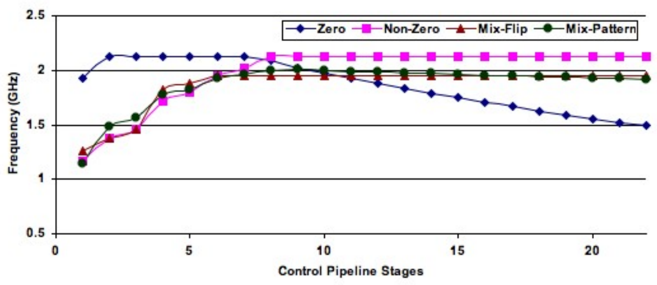

Fig. 14. 2-WCHB Zero-Path Control Slack

VI. Evaluation of Operand-Optimized FPA

The functional correctness of our asynchronous operand-optimized FPA was verified using `prsim`, our in-house asynchronous gate-level simulation tool. Ten billion randomly generated inputs were sourced into the FPA and the outputs were verified against the expected values from a standard processor. The random input set included verification tests for all four IEEE rounding modes as well as denormal, NaN, and infinity data inputs. The FPA was further tested with one billion stored inputs from actual application benchmarks. In the past, many of the designs have opted to handle denormal numbers using software traps [28] which can lead to long execution times [29]. Our design includes hardware support for denormal numbers.

Our improved asynchronous FPA combines all optimization techniques discussed in sections IV and V. On top of the energy savings associated with the these techniques, we were able to compact more logic together and in doing so eliminated a full pipeline stage. The transistors in our baseline FPA were sized using standard transistor sizing techniques [30]. To meet high performance targets, the pull-down stack was restricted to a maximum of six transistors in series (including the enable). The slow and power-consuming state-holding completion-elements were restricted to a maximum of three inputs at a time. Keepers and weak feedback inverters were added for each state-holding gate to ensure that charge would not drift even if the pipeline were stalled in an arbitrary state.

Since HSIM/HSPICE simulations do not account for wire capacitances, we included additional wire load in the SPICE file for every gate in the circuit. Based on prior experience with fabricated chips and post-layout simulation, we have found that our wire load estimates are conservative, and predicted energy and delay numbers are typically 10% higher than those from post-layout simulations. Our simulations use a 65nm bulk CMOS process at the typical-typical (TT) corner. Test vectors are injected into the SPICE simulation using a combined VCS/HSIM simulation, with Verilog models that implement the asynchronous handshake in the test environment. All simulations were carried out at the highest-precision setting.

As seen in Fig. 15, the energy per operation of the optimized FPA is approximately 2.3X (56.7%) less than that of baseline FPA across a wide range of throughput values for the same non-zero operands. In terms of overall throughput, our optimized FPA is within $\pm1.5\%$ of the baseline FPA across a range of voltages (0.6V to 1.1V). As noted earlier, it is possible to create pathological input operands that could degrade the throughput, for example long carry-chain lengths in the interleaved adder/incrementer or the case of alternating zero and non-zero operands; however, in practice, such inputs are rare. Even if they do occur, our FPA still operates correctly and produces IEEE-compliant output albeit at lower throughput.

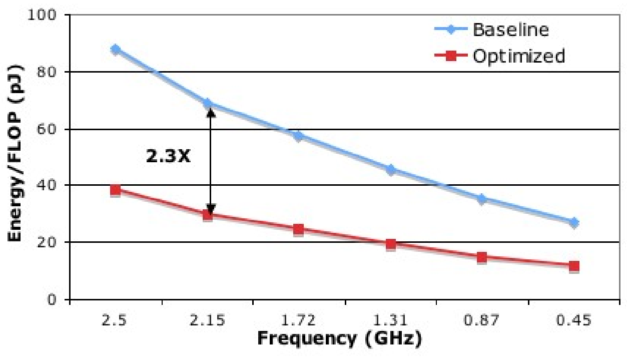

Fig. 15. Optimized vs. Baseline

The baseline FPA gives an energy-per-operation of $69.3pJ$ at an average throughput of 2.15 GHz for all input operands alike. The optimized FPA's energy-per-operation and throughput vary considerably based on the input operands as seen in Table II. These results, for SPICE simulations at a V$_{DD}$ of 1V with no slack on the zero operand bypass path, show our improved FPA design to be far superior in energy-efficiency than our baseline FPA.

TABLE II
OPTIMIZED FPA ENERGY & THROUGHPUT

Input Set	Energy/FLOP	Throughput
Nonzero (Align Shift 0-3)	30.2 pJ	2.15 GHz
Nonzero (Align Shift 4-55)	35.1 pJ	2.10 GHz
Nonzero (Align Shift Mix)	32.4 pJ	2.12 GHz
Zero Only	13.1 pJ	1.51 GHz
Zero-Nonzero Alternate	25.1 pJ	1.31 GHz
Zero 30%	27.4 pJ	1.85 GHz
Zero 8%	31.0 pJ	1.96 GHz

The energy-efficiency and throughput results for the FPA implementation with two WCHB pipeline stages on the zero bypass path are shown in Table III. The results for non-zero operands remain the same as before and hence are not repeated. The improvement in throughput for all zero-input patterns comes with additional power consumption. This offers a design choice to be made based on throughput and energy targets.

TABLE III
OPTIMIZED FPA 2-WCHB ZERO BYPASS

Input Set	Energy/FLOP	Throughput
Zero Only	14.2 pJ	2.1 GHz
Zero-Nonzero Alternate	26.1 pJ	1.95 GHz
Zero 30%	28.4 pJ	2.0 GHz
Zero 8%	32.1 pJ	2.1 GHz

In terms of actual application benchmarks, *Zero 8%* input mix corresponds to *416.gamess*, whereas *Zero 30%* corresponds to an average mix of operands from three applications: *447.deal*, *450.soplex*, and *437.leslie3d*.

The latency of our optimized FPA is also highly operand dependent. Table IV shows that compared to the baseline FPA's average latency of approximately 1098ps, the optimized FPA has an average latency of 737ps for zero operand cases (same for both two WCHB slack matching and no slack matching zero bypass implementations) and 1060ps for nonzero operands with align shifts of 0 to 3; a latency reduction of 32.8% and 3.5% respectively. The increase in latency, seen for rare some cases, could be attributed to the use of a variable-latency interleaved adder instead of fixed latency parallel-prefix tree adder.

TABLE IV
OPTIMIZED FPA LATENCY

Input Set	Latency
Nonzero (Align Shift 0-3)	1050-1070 ps
Nonzero (Align Shift 4-55)	1080-1120 ps
Zero	737 ps

Since leakage power has become an important design constraint, our simulations model sub-threshold and gate leakage effects in detail. Table V compares the total leakage power of our baseline and optimized FPAs at a V$_{DD}$ of 1V. Although, our optimized FPA includes extra control circuitry for multiple split-merge pipelines, there is a 19% reduction in leakage power.

TABLE V
LEAKAGE POWER

	Leakage Power
Baseline FPA	0.72 mW
Optimized FPA	0.58 mW

The decrease in leakage power could be attributed to the use of the interleaved adder and incrementer which use far fewer transistors compared to the Hybrid Kogge-Stone Carry-Select Adder and Carry-Select Incrementer. Also, compacting of logic stages eliminated a full pipeline stage and helped to reduce the total leakage power further. In terms of the total number of transistors, our optimized FPA uses 12% less transistors than the baseline.

Table VI compares the performance, power, and energy of our optimized FPA against both our own baseline and some of the latest FPAs and FMAs from industry and academia. The computer arithmetic literature has a large body of work on FPA and FMA designs, but few contain a detailed implementation that provides a reasonable point of comparison in a modern process. This guided our choice of other FPA/FMAs in Table VI. Our baseline and optimized FPA results are for simulations with an input-set comprising non-zero operands with right align shifts of 0 to 3.

We caution that the FMA numbers are not meant to be a direct comparison with our proposed FPA since an FMA contains additional circuitry. The FMA performance and power numbers were only included to show what is the best out there in industry and academia and that in spite of using a bulk process, our proposed FPAs are competitive both in terms of performance and energy-efficiency. Quinnell [15] has a lower overall latency for nonzero operands than our optimized FPA as well as our baseline FPA. However, this lower latency comes at the cost of 3.2X lower throughput and 5.9X higher energy per operation, as well as a higher V_{DD}.

All of our transistor-level simulation results quoted so far were for HSIM/HSPICE simulations done at a default temperature of 25°C. A set of simulations at 85°C showed a similar trend between the baseline and optimized FPAs but with an expected small performance degradation (10%) at higher temperature.

The high GFLOPS/Watt ratio of our optimized asynchronous FPA (26 GFLOPS/Watt at 2.5 GHz 1.1V) make a case for adopting asynchronous circuit solutions, similar to ours, in future high performance computing systems. Since asynchronous chips have been shown to work at fairly low voltages and are quite robust [34–36], getting 85.4 GFLOPS/Watt at a decent throughput of 450 MHz (at 0.6V) also shows the potential of our solution for embedded systems that require floating-point computation.

VII. Summary

We presented the detailed design of an asynchronous high-performance energy-efficient IEEE 754 compliant double-precision floating-point adder. Using QDI asynchronous pipelines, we created a high-performance design based on state-of-the-art FPA architectures. We analyzed the power consumption of the FPA datapath, identifying opportunities for energy reduction. By using asynchronous techniques that exploit average-case behavior, we reduced the energy of the FPA operation with nonzero operands by 56.7% compared to our baseline implementation while preserving the average throughput. In future, we plan to extend this work to develop asynchronous FMA architectures guided by similar principles to those outlined in this paper.

Acknowledgment

The authors would like to thank Carlos Tadeo Ortega Otero and Filipp Akopyan for setting up the HSIM and VCS cosimulation infrastructure which made all of our simulations possible.

The research was supported in part by NSF under grants CNS-0834582 and CCF-0428427. Equipment support was provided by NSF infrastructure grant CNS-0708788, and the processors were donated by Intel.

References

[1] The Institute of Electrical and Electronic Engineers, Inc. IEEE Standard for Binary Floating-point Arithmetic. ANSI/IEEE Std 754-1985.

[2] K. Krewell. Cell Moves into the Limelight. *Microprocessor Report*, February 14, 2005.

[3] D. Fang and R. Manohar. Non-Uniform Access Asynchronous Register Files. *Proc. IEEE International Symposium on Asynchronous Circuits and Systems*, 2004.

[4] A. Lines. Pipelined Asynchronous Circuits. Master's thesis, California Institute of Technology, 1995.

[5] A. J. Martin, A. Lines, R. Manohar, M. Nyström, P. Penzes, R. Southworth, U. V. Cummings, and T.-K. Lee. The Design of an Asynchronous MIPS R3000. In *Proc. Conference on Advanced Research in VLSI*, 1997.

[6] M. Ercegovac and T. Lang. *Digital Arithmetic.* Morgan-Kaufmann, 2004.

[7] A.W. Burks, H.H. Goldstein, and John von Neumann. Preliminary discussion of the logical design of an electronic computing instrument. Institute for Advanced Study, Princeton NJ, June 1946.

[8] R. Manohar and J.A. Tierno. Asynchronous Parallel Prefix Computation. *IEEE Transactions on Computers*, **47**(11):1244–1252, November 1998.

[9] R. Manohar. The Impact of Asynchrony on Computer Architecture. Ph.D. thesis, CS-TR-98-12, Department of Computer Science, California Institute of Technology, June 1998.

[10] D. Kinniment. An Evaluation of Asynchronous Addition. *IEEE Transactions on Very Large Scale Integrated Systems*, **4**(1):137–140, March 1996.

[11] J. Garside. A CMOS VLSI implementation of an asynchronous ALU. *Proc. IFIP Workshop on Asynchronous Design Methodologies*, 1993.

[12] S. Nowick, K.Y. Yun, P.A. Beerel, A.E. Dooply. Speculative Completion for the Design of High-Performance Asynchronous Dynamic Adders. *Proc. IEEE International Symposium on Asynchronous Circuits and Systems*, 1997.

[13] Joel R. Noche and Jose C. Araneta. An Asynchronous IEEE Floating-Point Arithmetic Unit. In *Proc. of Science Diliman*, Vol.19, No.2, 2007.

[14] Son Dao Trong, Martin Schmookler, Eric M Schwarz, and Michael Kroener. P6 Binary Floating-Point Unit. In *Proc. International Symposium on Computer Arithmetic*, 2007.

[15] Eric Quinnell, Earl E. Swartzlander,Jr, and Carl Lemonds. Floating-Point Fused Multiply-Add Architectures. *The Fortieth Asilomar Conference on Signals, Systems, and Computers*, 2007.

[16] T. Lang and J. D. Bruguera. Floating-Point Fused Multiply-Add: Reduced Latency for Floating-Point Additions. In *Proc. International Symposium on Computer Arithmetic*, 2005.

[17] Peter-Michael Seidel and Guy-Even. On the Design of Fast IEEE Floating-Point Adders. In *Proc. International Symposium on Computer Arithmetic*, 2001.

[18] Peter-Michael Seidel. Multiple path IEEE floating-point fused multiply-add. In *Proc. International Midwest Symposium on Circuits and Systems*, 2003.

[19] A. Beaumont-Smith, N. Burgess, S. Lefrere, and C. Lim. Reduced latency IEEE floating-point standard adder architectures. In *Proc. International Symposium on Computer Arithmetic*, 1999.

[20] S. Oberman, H. Al-Twaijry, and M. Flynn. The SNAP project: Design of floating point arithmetic units. In *Proc. International Symposium on Computer Arithmetic*, 1997.

[21] S. Oberman. Floating-point arithmetic unit including an efficient close data path. AMD, U.S. patent 6094668, 2000.

[22] J. D. Bruguera and T. Lang. Leading-One Prediction with Concurrent Position Correction. *IEEE Transactions on Computers*, Volume 48, Issue 10, October 1999.

TABLE VI
COMPARISON TO OTHER FPAS AND FMAS

Name	Type	Process	VDD	Frequency	Latency	Power	Energy/Op	GFLOPS/W
Async Optimized	FPA	65nm	1	2.15 GHz	57.2FO4s 1060ps	64.9mW	30.2pJ	33.1
Async Baseline	FPA	65nm	1	2.15 GHz	59.3FO4s $\approx$ 1098ps	149mW	69.3 pJ	14.5
IBM Power6 [14]	FMA	65nm SOI	1.1	4 GHz	78FO4s	310mW	77.5 pJ	12.9
Merrimac [33]	FMA	90nm	1	1 GHz	NA	110mW	110 pJ	9.09
Quinnell [15]	FPA	65nm SOI	1.3	666 MHz	946ps	118mW	177.17 pJ	5.64

[23] R. V. K. Pillai, D. Al-Khalili, and A. J. Al-Khalili. A Low Power Approach to Floating Point Adder Design. In *Proc. of the International Conference on Computer Design*, 1997.

[24] P. M. Kogge and H. S. Stone. A Parallel Algorithm for the Efficient Solution of a General Class of Recurrence Equations. *IEEE Transactions on Computers*, C-22, August 1973.

[25] Chi-Keung Luk, Robert Cohn, Robert Muth, Harish Patil, Artur Klauser, Geoff Lowney, Steven Wallace, Vijay Janapa Reddi, Kim Hazelwood. Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation. *Programming Language Design and Implementationn (PLDI)*, 2005.

[26] SPEC Benchmark Suite. Information available at http://www.spec.org

[27] Christian Bienia, Sanjeev Kumar, Jaswinder Pal Singh and Kai Li. The PARSEC Benchmark Suite: Characterization and Architectural Implications. In *Proc. of the 17th International Conference on Parallel Architectures and Compilation Techniques*, 2008.

[28] M. Schmookler, M. Putrino, A. Mather, J. Tyler, H. Nguyen, C. Roth, M. Pham, J. Lent, and M. Sharma. A Low-Power, High-Speed Implementation of a PowerPC Microprocessor Vector Extension. In *Proc. of the International Symposium on Computer Arithmetic*, 1999.

[29] Eric M Schwarz, M. Schmookler, and S. D. Trong. FPU implementations with denormalized numbers. *IEEE Transactions on Computers*, Volume 54, Issue 7, July 2005.

[30] N. Weste and D. Harris. CMOS VLSI Design: A Circuits and Systems Perspective. Addison Wesley, third edition, 2004.

[31] U. V. Cummings, A. M. Lines, and A. J. Martin. An Asynchronous Pipeline Lattice-structure Filter. In *Proc. of the International Symposium on Advanced Research in Asynchronous Circuits and Systems*, 1994.

[32] John Teifel and Rajit Manohar. A High Speed Clockless Serial Link Tranceiver. In *Proc. of the International Symposium on Asynchronous Circuits and Systems*, 2003.

[33] William J. Dally, Patrick Hanrahan, Mattan Erez, Timothy J. Knight, Franois Labont, Jung-Ho Ahn, Nuwan Jayasena, Ujval J. Kapasi, Abhishek Das, Jayanth Gummaraju, Ian Buck. Merrimac: Supercomputing with Streams. *IEEE Conference on Supercomputing*, 2003.

[34] V. Ekanayake, C. Kelly IV, and R. Manohar. An Ultra-low-power Processor for Sensor Networks. *Proceedings of the 11th International Conference on Architectural Support for Programming Languages and Operating Systems*, October 2004

[35] A.J. Martin, M. Nystrom, K. Papadantonakis et al. The Lutonium: A Sub-Nanojoule Aynchronous 8051 Microcontroller. *Proc. 9th IEEE International Symposium on Asynchronous Circuits and Systems*, May 2003.

[36] D. Fang, J. Teifel, and R. Manohar. A High-Performance Asynchronous FPGA: Test Results. *2005 IEEE Symposium on Field-Programmable Custom Computing Machines*, April 2005.