A Three-Path Floating-Point Fused Multiply-Add Design

Eric Quinnell
AMD
University of Texas at Austin
Eric.Quinnell@amd.com

Earl E. Swartzlander, Jr.
Dept. of Electrical and
Computer Engineering
University of Texas at Austin

Carl Lemonds
AMD
9500 Arboretum Blvd.
Austin, TX 78752

ABSTRACT

A new floating-point fused multiply-add architecture for the single instruction execution of (A x B) + C is presented. The implementation uses parallel hardware paths based on the arithmetic fundamentals similar to those employed in the design of modern dual-path floating-point adders.

Both the three-path fused multiply-add unit and the classical fused multiply-add unit have been designed using AMD 65nm silicon on insulator technology with the 'Barcelona' native quad-core standard cell library to provide a realistic and fair comparison of both architectures.

Although its area is about 40% larger, the three-path architecture shows an approximate reduction of 12% in latency as well as a reduction of about 15% in power consumption relative to the classical fused multiplier-adder.

1 INTRODUCTION

The fused multiply-add operation was introduced in 1990 on the IBM RS/6000 for the single instruction execution of the equation (A x B) + C with single and double precision floating-point operands [1], [2]. This hardware unit was designed to reduce the latency of dot product calculations. Additionally, the fused multiply-add unit provides greater precision since only a single rounding is performed on the combined full precision product and sum.

Since 1990, a plethora of algorithms that utilize the (A x B) + C single-instruction operation have been introduced, for applications in DSP and graphics processing [3], [4], FFTs [5], FIR filters [3], division [6], argument reduction [7], etc. To accommodate the increased use of the fused multiply add instruction, several commercial processors have implemented embedded fused multiply-add units. These chips include designs by IBM [1], [8]-[10], HP [11], [12], MIPS [13], ARM [3], and Intel [14], [15]. Some chips entirely replace the floating-point adder and floating-point multiplier with a fused multiply-add unit by using constants to perform single floating-point operations, e.g., (A x B) + 0.0 for multiplies and (A x 1.0) + C for adds. All these realizations use a serial implementation based on a modified IBM RS/6000 architecture. This combination of commercial implementation and increased algorithmic activity has pushed the IEEE 754r committee to consider including the fused multiply-add instruction into the proposed IEEE standard for floating-point arithmetic [16].

Several proposals for the improvement of fused multiply-add execution units have been made. A reduced latency fused multiplier-adder has been proposed that combines the addition and rounding stage to reduce the number of serial stages required by a traditional fused multiply-add unit [17]. A 5-case fused multiply-add that processes data in two parallel hardware paths is proposed in [18]. Fused multiply-add designs have been proposed that use a common floating-point adder dual-path scheme within the fused multiply-add to allow floating-point additions to bypass the multiplier [19]. Though such proposals have been made, they have yet to be implemented.

This paper introduces a new architecture to reduce the latency and power consumption of fused multiplier-adders. The three-path fused multiply-add design uses three parallel hardware paths to reduce latency by a return to the floating-point arithmetic fundamentals presented by Farmwald's dual-path floating-point adder [20], rather than an attempt to force a fused multiply-add unit into a floating-point adder dual-path system. Additionally, a three-case hardware system that selectively turns on paths for arithmetic data ranges provides a unique opportunity for power savings when using a method similar to that of [21]. In this design, only one of the three paths is ever turned on for any possible instruction.

Both the three-path fused multiplier-adder as well as a classic fused multiply-add circuit have been designed using AMD's 65nm silicon on insulator technology models and circuit design flow. The circuits use the 'Barcelona' native quad-core standard cell library in a fully floorplanned model and have both been simulated with full cell optimizations and routing parasitics. These complete implementations provide a realistic and fair comparison of the timing, power consumption, and area of both of the architectures.

2.1 THE CLASSIC FUSED MULTIPLY-ADD ARCHITECTURE

The fused multiply-add architectures implemented in industry are all based on the original serial design of the IBM RS/6000 [1], [2]. There are localized improvements and varying bit-width changes from one design to another, but the basic architecture remains the same. The classic fused multiply-add architecture is shown in Figure 1.

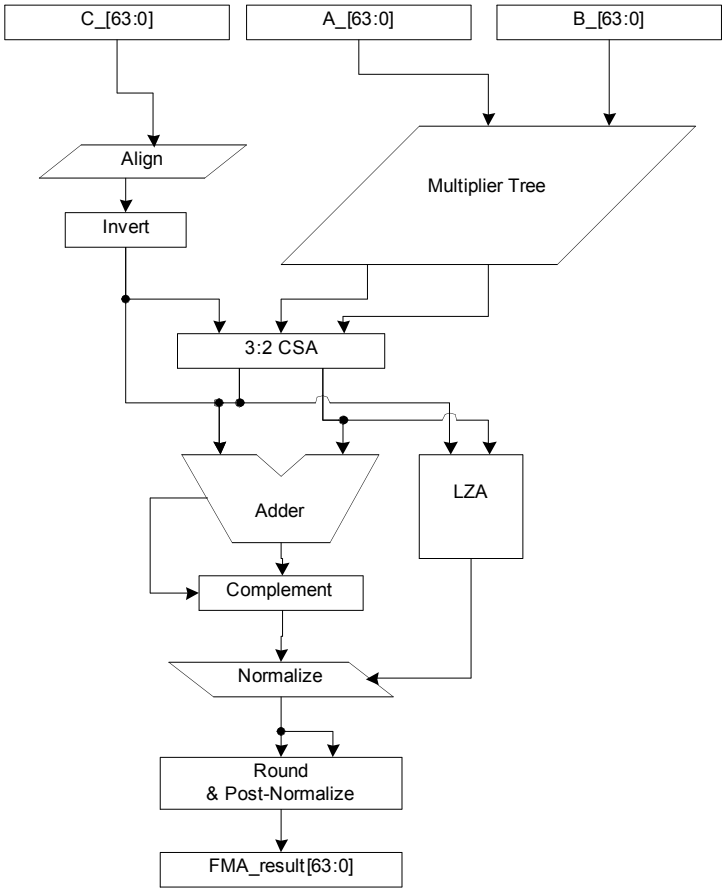

Figure 1. Classic Fused Multiply-Add Architecture.

There are several serial steps to complete the execution of a fused multiply-add instruction in double-precision format on the traditional architecture:

1.) Multiply the 53-bit input significands A x B to produce a 106 bit carry-save format result. Align the addend C to the appropriate point in the range of the 106-bit product, including up to 55 bits above the product (53-bit significand length and a double carry buffer of 2 bits) based on exponent difference. This requires a 161-bit aligner.
2.) Combine the lower 106 bits of the addend and the product with a 3:2 CSA to produce a 161-bit string of data in carry-save format.
3.) Input the 161-bit string into a 161-bit adder, or more efficiently, a 109-bit adder followed by a 52-bit incrementer.
4.) Input the lower 109 bits into a 109-bit LZA that processes in parallel with the adder for the case of massive cancellation as well as addends smaller than the product.
5.) Complement the sum if required. This step may be performed a number of ways, including incrementers, end-around-carry (EAC) adders, or massive parallelism.
6.) Normalize the sum based on the output of the LZA, the exponent difference, and in some designs, a zero detect on the output of the top 52-bit incrementer.
7.) Round the result and post-normalize if necessary. This requires a 52-bit adder or incrementer plus control logic, depending on data-type support.

2.2 CLASSIC FUSED MULTIPLY-ADD CRITICAL PATHS

In 65nm implementations, the trouble paths of the classic fused multiply-add implementation come directly from data paths with large interconnect distances, especially those that do not naturally fan out into inverter drivers. These paths are impacted significantly by the scaling problems found with interconnects at sub-micron levels. Although transistors improve in performance as the feature size is reduced, wire interconnects face increased resistance and wire capacitance. As a result interconnections are quickly becoming a significant performance limitation for modern VLSI circuits [22] - [24].

With the dominant presence of interconnect scaling problems, the timing tools quickly identify the critical steps (in terms of latency) in the classic fused multiply-add unit as those found in

steps 1, 3, and 5 from above: the 161-bit shifters and 161-bit add/complement stages. In the first half of the classic fused multiply-add architecture, the long interconnect distances and fan-out latencies of the exponent difference logic followed by the 161-bit aligner described in step 1 actually exceed the delay of the multiplier. Additionally, the power consumption of this unit is high, as the presence of long data paths with heavy interconnect capacitance requires a large amount of dynamic power consumption to provide acceptable edge-rates.

In the middle of the fused multiply-add circuit, the 109-bit adder from step 3 provides a very difficult timing arc for a similar interconnect reasons. The popular use of prefix-style adders [25] generally requires inverting PG cells at every adder level in the form of an AOI/OAI circuit. In adders as large as 109-bits, the interconnect distances increase exponentially from each logical stage to the next. Due to the poor driving ability of AOI/OAI circuits and the increasing physical net distance, the adder PG cells require buffering mid-way through the circuit, increasing the logic stage count.

The final high latency block of the classic fused multiply-add architecture comes from the requirement for a complement in the case of an incorrect operand inversion. This contributes to high latency by either adding fan-out load to the prefix adder in the form of an end-around-carry (EAC), or by introducing an independent complement/incrementer block.

These classic fused multiply-add timing and power difficulties are expected to become more severe as the feature sizes continue to decrease. Stages 1, 3, and 5 of the classic fused multiply-add are the critical pieces to consider in any fused multiply-add implementation, and should be addressed by any new fused multiply-add architectures.

3.1 THREE-PATH FUSED MULTIPLY-ADD

Following the study of the classic fused multiply-add architecture, implementation, and critical paths described in Section 2 as well as the consideration of the push in the literature for fused multiply-add parallelism, this paper proposes a new fused multiply-add architecture that incorporates the basic philosophies listed for an improved architectural fused multiply-add design:

1. Parallelize fused multiply-add hardware so the data is not subjected to prediction stages that, for most cases, provide little computational benefit.
2. When designing parallel hardware paths, follow the basic concept of the Farmwald dual-path floating-point adder [20].
3. Fix the classic fused multiply-add wire dominance problem by reducing the size of the critical path components or by completely removing them.
4. Use the front-end multiplier CSA delay as an opportunity to pre-compute the necessary parallel fused multiply-add path.
5. Design the parallel hardware paths so that they are logically exclusive, allowing for path pre-computation logic to shut them down in a power-reduction effort if incoming data will not use them.

The three-path fused multiply-add architecture shown in Figure 2 is designed to follow these guidelines. The global design splits the data-path following the multiplier tree into three case specific blocks, each designed with different data "anchors" as explained later. This partitioning of anchor cases removes the need for a massive aligner as well as a complementing stage. Instead, the design partitions alignments and inversions at local levels.

Following the path selection, the appropriate block processes and prepares the numerical data for a combined add/round stage. As in many modern arithmetic unit designs, a combined add/round stage removes the requirement for a massive adder followed by another addition/increment unit for the purpose of IEEE-754 compliant rounding.

The specifics of each path, as well as an explanation of the selected add/round scheme, are described in detail in the following sections.

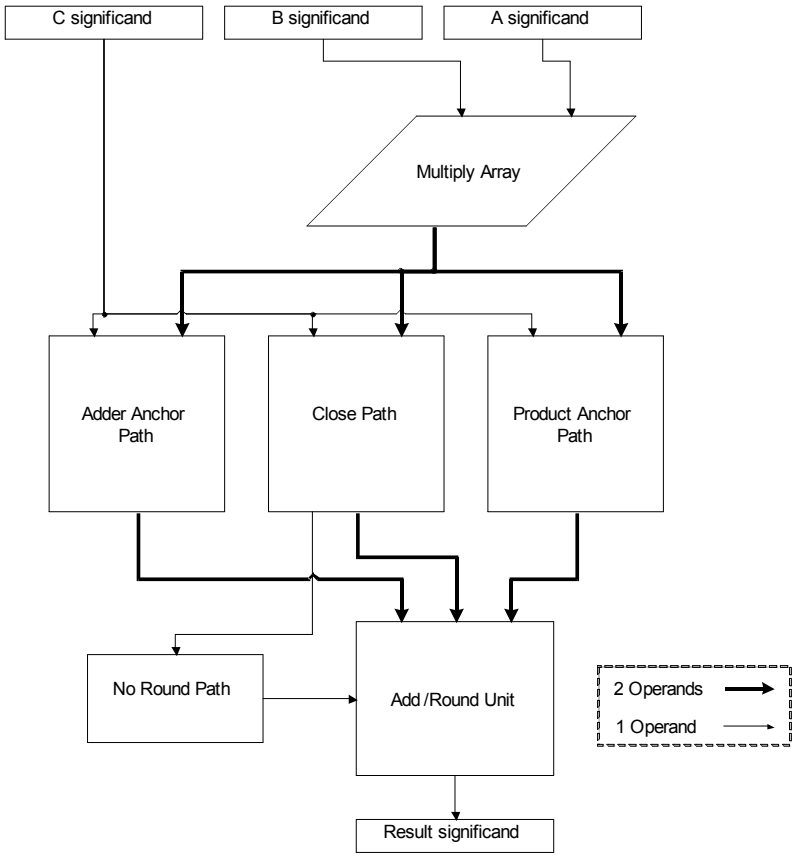

Figure 2. Three-Path Fused Multiply-Add Architecture.

3.2 THREE-PATH FUSED MULTIPLIER-ADDER ANCHOR PATHS

The three-path fused multiply-add unit uses two "anchor" paths for data dependent processing. As seen in Figure 2, these two blocks are the adder anchor path and the product anchor path. The use of the term "anchor" is a reference to the design philosophy found in the Farmwald floating-point adder designs [20].

In a dual-path floating-point adder, the "far" path always begins by finding the larger number and locking its position, i.e., using it as an "anchor." Once the larger operand is known, the second operand is aligned and inverted in the case of subtraction without ever needing a corrective complement.

In the case of a fused multiplier-adder, a similar use of a floating-point adder far path is not feasible, as the range of positions in double-precision format spans 161-bits as compared to 52-

bits in an floating-point adder. Additionally, if such a system were applied to a fused multiplier-adder, both 161-bit ranges of addends and products would need to have the option of inversion and swapping. Applying this system to something as massive as the fused multiply-add data range is not realistic.

A better solution for dealing with the fused multiply-add data range is by splitting the anchor-based algorithm into two cases. To start, the benefit of a fused multiply-add unit is that the exponent difference is known well ahead of the multiplier product, so a logical data-range may be selected early in the circuit. In cases of large exponent differences, either the addend or the product will be larger and always without ambiguity. The three-path fused multiply-add design takes the exponent difference (which is known early) and anchors whichever operand is larger, forcing the other operand to invert (if necessary) and align. This anchoring method requires partitioning of the 161-bit data range into two smaller sets.

Figure 3 shows the adder anchor path in detail. This path is selected when the exponent difference detects that the addend is larger. For this case, the addend is anchored. The later arriving product is then aligned over a 57-bit range and inverted for subtracts. Following the inversion stages, all three operands are combined in 3:2 carry save adders (CSAs) or half adders (HAs) to produce two 163-bit numbers. The most significant bits of both results are used for corner-case correction, and the lower 55-58 bits are sent to a carry/sticky tree, as the least significant bits will never be selected in the final result.

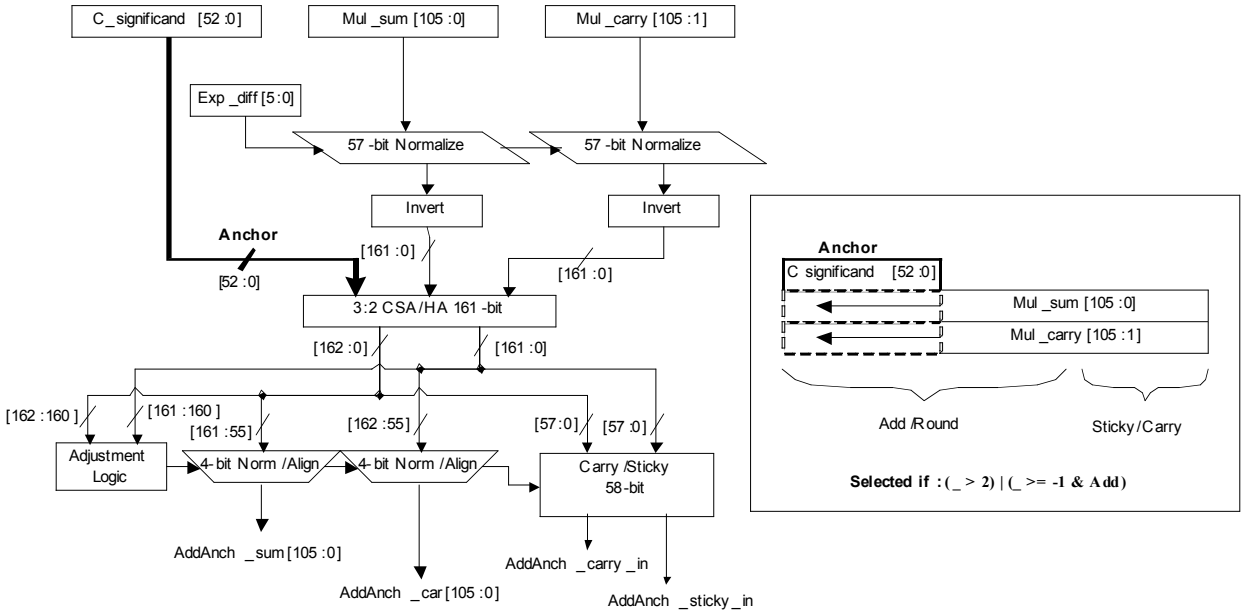

Figure 3. The Adder Anchor Path.

The adder anchor path finalizes with two 106-bit operands ready for addition and rounding as well as an input carry and sticky bit generated by the discarded lower bits. The adder anchor unit is not on the critical path, so there is sufficient time to normalize the product over a 57-bit range.

Figure 4 shows the product anchor path. This path is the complement of the adder anchor path and is enabled when the exponent difference determines that the product is larger than the addend. Much like the classic fused multiplier-adder, the addend is aligned and inverted (if necessary) against the position of the product anchors. However, in this design the data need only cover a 106-bit range as opposed to the original 161-bit range. When the product arrives from the multiplier, all the data are combined in a 3:2 CSA, adjusted and sent in 106-bit sum/carry form to the add/round stage.

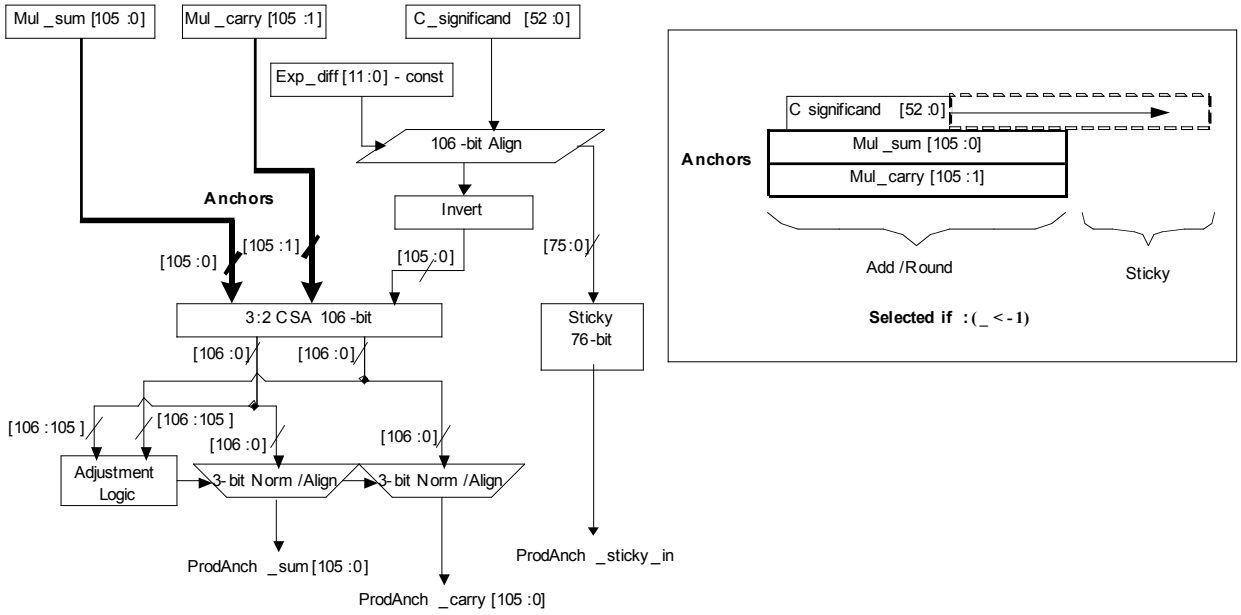

Figure 4. The Product Anchor Path.

3.3 THREE-PATH FUSED MULTIPLIER-ADDER CLOSE PATH

For cases when the exponent difference between the addend and the product is too close to easily determine a larger operand, all data is passed to the close path. This path only handles fused multiplier-adder subtraction operations and is geared specifically to deal with massive cancellation.

To follow suit with the two anchor paths, the close path is designed to remove the requirement of a complementation stage. Shown in Figure 5, the close path accomplishes this via significand swapping. First, the close path uses 3:2 CSAs and HAs to combine a complemented aligned addend with the product. Likewise, the logically opposite term is also created with a complemented product and an un-complemented addend.

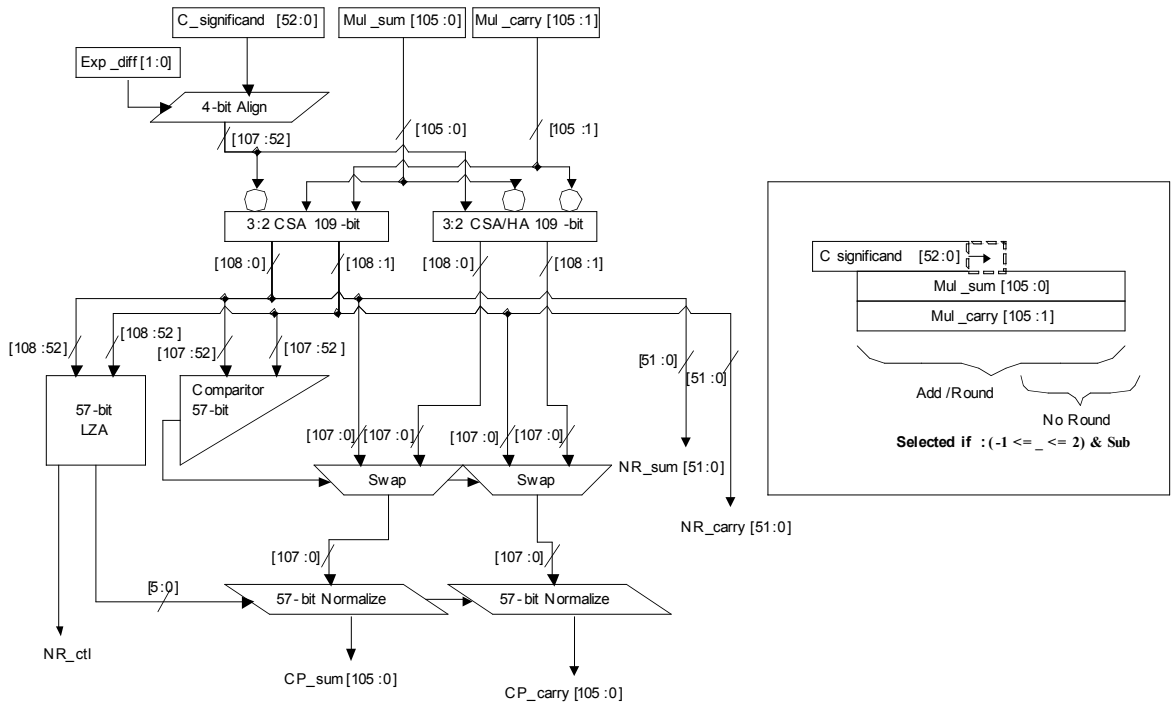

Figure 5. The Close Path.

The first 3:2 combination is passed to a 57-bit comparator (57-bits is selected since all bits after position 57 in the aligned addend are always '0's) to determine which of the operands is larger. The comparator signals the swap multiplexers to choose the correct inversion combination and the results are normalized in preparation for addition and rounding. The LZA that controls this normalization is passed only one combination of inversion inputs, as its functionality is not affected by which operand is larger. Depending on the addition/rounding scheme selected, the one-bit LZA correction shift may be handled in the add/round block.

Timing simulations early in the design of the three-path fused multiply-add unit quickly identified the close path as the critical timing arc. As a result the design was changed to reduce the critical path by shrinking the bit-sizes of the high-latency components. Specifically, the original 109-bit LZA and 109-bit normalizers were reduced to a 57-bit range. Logically, this reduction is a legal move, as cases of massive cancellation exceeding 57-bits in length will produce a result that needs no rounding. This "no round" case is triggered by the 57-bit LZA '1's detection' term. If selected, data enter the no round path and an addition and normalization are performedin parallel with the add/round stage.

3.4 THREE-PATH FUSED MULTIPLIER-ADDER ADD/ROUND STAGE

All three middle stage paths in the three-path fused multiply-add design prepare the data for the 106-bit add/round stage. The combined addition and rounding stage algorithm combines various suggestions for the add/round stages of a floating-point multiplier [26]-[28] with modifications to the control logic, signals, and multiplexer sizes to account for the fused multiplier-adder functionality. Finally, a "no round" path block has also been added in parallel to the scheme to handle the extra output case from the close path.

The combined fused multiplier-adder add/round stage is shown in Figure 6. The stage begins with a control block that selects the correct three path output and directs the data to the add/round scheme. The upper 54-bits of the selected data enter two half adder stages that remove least significant bits for rounding control. The lower 53-bits are passed to a carry and sticky block that produces the round and carry bits for the final round logic. One of the stage input bits goes to both.

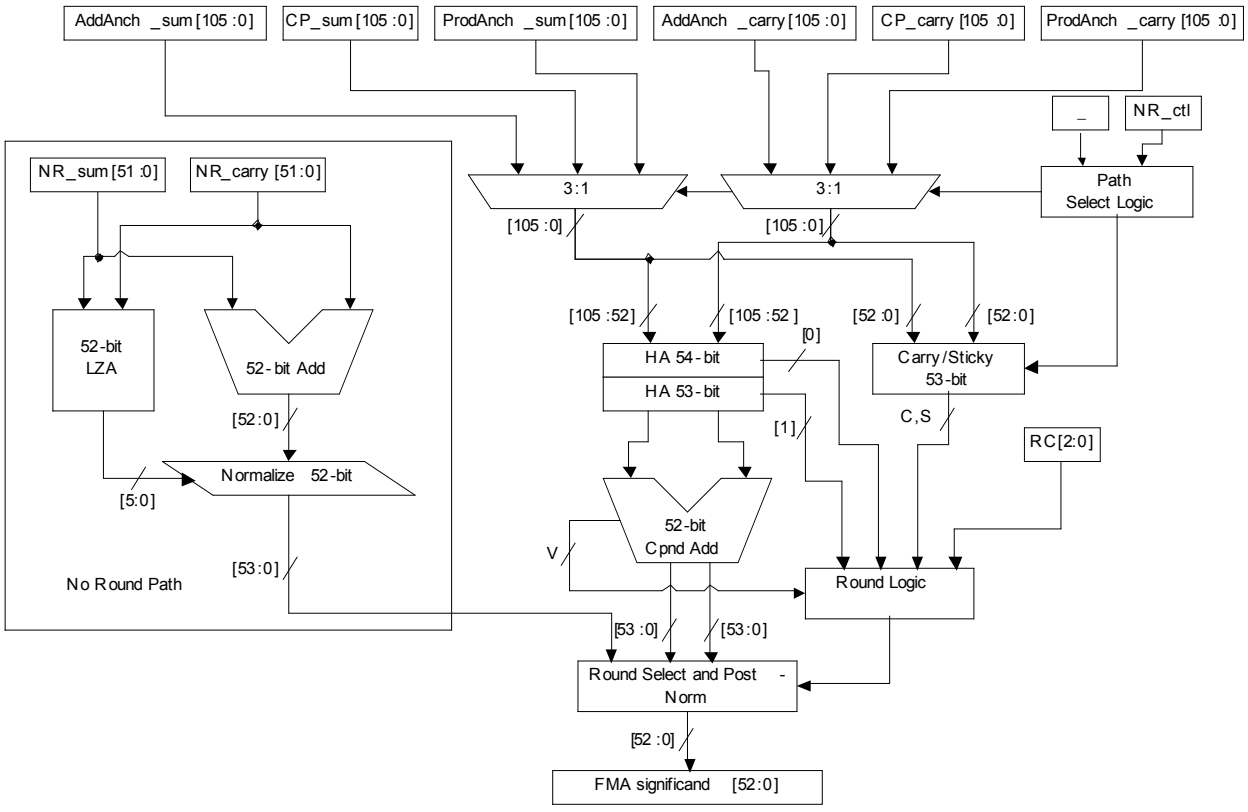

Figure 6. The No Round Path (left) and Add/Round Stage (right).

The data from the half adder stages enter a 52-bit compound adder, which produces a sum and an augmented sum (i.e., sum+1). Meanwhile, the rounding logic adjusts the lower 2 bits of the half adders and sends a carry out select signal to choose the correct adder output. The selected result is post-normalized and latched.

In the case of a close path selection with the no round signal assertion, the no round data inputs are added and normalized in a path separate from and parallel to the add/round stage. The result from this "no round" path is forwarded to the add/round result multiplexer, post-normalized, and latched.

When either the no round path result or add/round result is latched, the fused multiply-add instruction is complete and the data exit the unit.

4 IMPLEMENTATION

Both the three-path fused multiply-add architecture and the classic fused multiply-add architecture have been designed in the AMD 65nm circuit design tool flow using the 'Barcelona' native quad-core silicon on insulator standard cell libraries. Both implementations were designed with equivalent components and identical design rules to ensure that the resulting circuits are suitable for a realistic and fair comparison.

Both designs were first developed and simulated at a behavioral level using Verilog2k RTL code. These high level designs were compiled using Synopsys Chronologic VCS compilers and designed/debugged using Novas Debussy waveform viewers. The behavioral code was verified through a combination of both custom test benches and component behavioral equivalence checks embedded throughout the code.

Following behavioral Verilog RTL model completion, both architectures were reduced to gate level Verilog using standard cell library modules. The gate level equivalents were formally verified against the RTL code via Cadence/Verplex LEC software.

The complete gate level descriptions of both circuits were then ported to an AMD internal floor planning tool, which includes power grids and clock-gates. Then they were placed and resized based on cell fan-out and estimated routing parasitic capacitances. The final floor planned models were timed using Synopsys Primetime software with AMD SOI 65nm transistor models and characterizations. The model routing information was generated using Steiner routing estimates for every net and data extracted using AMD process parasitic capacitance metal routing tables.

Final area results were calculated by measuring the floorplan dimensions of each circuit. Any unused space within the circuit's area did not affect the final area calculations. Timing runs were performed first at a 1.3V 100 ˚C RVT corner, and then a 0.7V 100 ˚C LVT corner to confirm that the relative architecture timing is process corner independent. Both timing runs ran from fused multiplier-adder input to output with no internal pipelining.

Finally, power calculations were performed in HSim using extracted SPICE netlists generated from the gate level descriptions including the floor planning parasitics. The simulations were given randomly generated input sequences and a clock period of 1250 ps (to cover the latency of a single worst case classic fused multiply-add calculation) at 1.3V 100 ˚C for a series of runs. The final power result was measured from the maximum worst case single cycle power load and then normalized by the simulation frequency.

5 RESULTS

Table 1 compares the classic fused multiply-add and the three-path fused multiply-add designs in the categories of latency, area, and power consumption. The comparison results provide absolute as well as relative results. The difference row provides the increase/decrease of the three-path fused multiplier-adder relative to the classic fused multiplier-adder.

As shown in Table 1, the three-path fused multiply-add design shows about a 12% decrease in latency as compared to a classic fused multiply-add unit. Additionally, when clocked at the same frequency, the three-path fused multiply-add design provides about a 15% reduction in the power

consumption. Both the power and latency gains of the three-path fused multiply-add architecture come at the price of a nearly 40% increase in area.

Table 1. Fused Multiply-Add Results.

Design	Latency 1.3V 100 ˚C RVT	Latency 0.7V 100 ˚C LVT	Area 65nm AMD SOI	Power (max) 800 MHz 1.3V RVT
Classic FMA	1224ps	3363ps	186,930 um^2	499mW
Three-Path FMA	1081ps	2959ps	259,005 um^2	425mW
Difference	–11.7%	–12.0%	38.6%	–14.8%

6 CONCLUSION

A new architecture for the design and implementation of a reduced power and latency fused multiplier-adder is presented. The three-path fused multiply-add unit reduces the latency by using a parallel design focused on the fundamental notion of data dependent "anchors" that are logically split into three mutually exclusive paths. Additionally, this approach achieves significant power reduction from its ability to shut down two of the three hardware paths for every possible data range. The improvements in latency and power consumption are achieved at the cost of a modest increase in circuit area and complexity.

REFERENCES

[1] R.K. Montoye, E. Hokenek and S.L. Runyon, "Design of the IBM RISC System/6000 floating-point execution unit," *IBM Journal of Research & Development*, Vol. 34, pp. 59-70, 1990.

[2] E. Hokenek, R. Montoye and P.W. Cook, "Second-Generation RISC Floating Point with Multiply-Add Fused," *IEEE Journal of Solid-State Circuits*, Vol. 25, pp. 1207-1213, 1990.

[3] C. Hinds, "An Enhanced Floating Point Coprocessor for Embedded Signal Processing and Graphics Applications," *Conference Record of the Thirty-Third Asilomar Conference on Signals, Systems, and Computers*, pp. 147-151, 1999.

[4] Y. Voronenko and M. Puschel, "Automatic Generation of Implementations for DSP Transforms on Fused Multiply-Add Architectures*," International Conference on Acoustics, Speech and Signal Processing*, pp. V-101-V-104, 2004.

[5] E. N. Linzer, "Implementation of Efficient FFT Algorithms on Fused Multiply-Add Architectures," *IEEE Transactions on Signal Processing*, Vol. 41, pp. 93-107, 1993.

[6] A. D. Robison, "N-Bit Unsigned Division Via N-Bit Multiply-Add," *Proceedings of the 17th IEEE Symposium On Computer Arithmetic*, pp. 131-139, 2005.

[7] R.-C. Li, S. Boldo and M. Daumas, "Theroems on Efficient Argument Reductions," *Proceedings of the 16th IEEE Symposium on Computer Arithmetic*, pp. 129-136, 2003.

[8] F. P. O'Connell and S. W. White, "POWER3: The Next Generation of PowerPC Processors," *IBM Journal of Research and Development*, Vol. 44, pp. 873-884, 2000.

[9] R. Jessani and C. Olson, "The Floating-Point Unit of the PowerPC 603e," *IBM Journal of Research and Development*, Vol. 40, pp. 559-566, 1996.

[10] R.M. Jessani and M. Putrino, "Comparison of Single- and Dual-Pass Multiply-Add Fused Floating-Point Units," *IEEE Transactions on Computers*, Vol. 47, pp. 927-937. 1998.

[11] A. Kumar, "The HP PA-8000 RISC CPU," *IEEE Micro Magazine*, Vol. 17, Issue 2, pp. 27-32, April, 1997.

[12] D. Hunt, "Advanced Performance Features of the 64-bit PA-8000," *Proceedings of Compcon*, pp. 123-128, 1995.

[13] K. C. Yeager, "The MIPS R10000 superscalar microprocessor," *IEEE Micro Magazine*, Vol. 16, No. 2, pp. 28-40, March, 1996.

[14] B. Greer, J. Harrison, G. Henry, W. Li and P. Tang, "Scientific Computing on the Itanium Processor," *Proceedings of the ACM/IEEE SC2001 Conference*, pp. 1-8, 2001.

[15] H. Sharangpani and K. Arora, "Itanium Processor Microarchitecture," *IEEE Micro Magazine*, Vol. 20, No. 5, pp. 24-43, Sept-Oct, 2000.

[16] *DRAFT Standard for Floating-Point Arithmetic P754*, IEEE Standard (proposed), Aug 18, 2006.

[17] T. Lang and J. D. Bruguera, "Floating-Point Fused Multiply-Add with Reduced Latency," *IEEE Transactions on Computers*, Vol. 53, pp. 988-1003, 2004.

[18] P.-M. Seidel, "Multiple Path IEEE Floating-Point Fused Multiply-Add," *Proceedings of the 46th IEEE International Midwest Symposium on Circuits and Systems*, pp. 1359-1362, 2003.

[19] T. Lang and J. D. Bruguera, "Floating-Point Fused Multiply-Add: Reduced Latency for Floating-Point Addition," *Proceedings of the 17th IEEE Symposium on Computer Arithmetic,* pp. 42-51, 2005.

[20] M. P. Farmwald, *On the Design of High Performance Digital Arithmetic Units,* Ph.D. thesis, Stanford University, 1981.

[21] R.V.K. Pillai, S.Y.A. Shah, A.J. Al-Khalili, and D. Al-Khalili, "Low Power Floating Point MAFs – A Comparative Study," *Sixth International Symposium on Signal Processing and its Applications*, Vol. 1, pp. 284-287, August, 2001.

[22] J. A. Davis, R. Venkatesan, A. Kaloyeros, M. Beylansky, S. J. Souri, K. Banerjee, K. C. Saraswat, A. Rahman, R. Reif, and J. D. Meindl, "Interconnect Limits on Gigascale Integration (GSI) in the 21st Century," *Proceedings of the IEEE*, Vol. 89, pp. 305-324, 2001.

[23] S. Natarajan and A. Marshall, "Technological Innovations to Advance Scalability and Interconnects in Bulk and SOI," *Proceedings of the 15th International Conference on VLSI Design,* pp. 297-298, 2002.

[24] R.K. Krishnarnurthy, A. Alvandpour, V. De, and S. Borkar, "High-Performance and Low-Power Challenges for Sub-70 nm Microprocessor Circuits," *Proceedings of the IEEE 2002 Custom Integrated Circuits Conference,* pp. 125-128, 2002.

[25] G. Dimitrakopoulos and D. Nikolos, "High-Speed Parallel-Prefix VLSI Ling Adders," *IEEE Transactions on Computers*, Vol. 54, pp. 225-231, 2005.

[26] G. Even and P. M. Seidel, "A Comparison of Three Rounding Algorithms for IEEE Floating-Point Multiplication," *IEEE Transactions on Computers,* Vol. 49, pp. 638-650, 2000.

[27] R. K. Yu and G. B. Zyner, "167 MHz Radix-4 Floating-Point Multiplier," in *Proceedings of the 12th Symposium on Computer Arithmetic,* pp. 149-154, 1995.

[28] N. Quach, N. Takagi, and M. Flynn, *On Fast IEEE Rounding*, Technical Report CSL-TR-91-459, Stanford Univ., Jan. 1991.