Bridged 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 design for the single instruction execution of (A x B) + C is presented. The bridge fused multiply-add unit re-uses common floating-point unit components to allow the hardware capability of a fused multiply-add instruction without degrading the performance of standard floating-point add or floating-point multiply instructions.

To evaluate the performance, area, and power costs of adding fused multiply-add functionality to existing common floating-point arithmetic units, several circuits including a double-precision floating-point adder, a floating-point multiplier, a classic fused multiply-add unit, and a bridge fused multiply-add unit have been designed and implemented with AMD 65nm silicon on insulator technology to provide a realistic and fair comparison of architectural options for floating-point arithmetic units.

The new bridge fused multiply-add unit shows almost identical latency and power consumption for addition and multiplication instructions (with a higher-performance fused multiply-add operation) relative to stand-alone floating-point arithmetic units at the cost of increased area. It does exhibit lower-performance for the fused multiply-add operation relative to the classic fused multiplier-adder.

1. INTRODUCTION

The fused multiply-add operation was introduced on the IBM RS/6000 for the single instruction execution of the equation (A x B) + C [1], [2]. This hardware unit was designed to reduce the latency of dot-product calculations. Additionally, the fused multiply-add unit provides greater precision results since only a single rounding is performed after both the multiplication and addition are performed at full precision.

Since 1990, many algorithms that utilize the (A x B) + C single-instruction equation have been introduced, for applications in DSP and graphics processing [3], [4], FFTs [5], FIR filters [3], division [6], argument reduction [7], and so on. Several industrial level chips have been designed and implemented with 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 (FPA) and floating-point multiplier (FPM) with an fused multiply-add unit by using constants to perform single floating-point operations, e.g., (A x B) + 0.0 for single multiplies and (A x 1.0) + C for single adds. This combination of industrial implementation and increased algorithmic activity has pushed the IEEE 754r committee to consider including the fused multiply-add instruction into the IEEE standard for floating-point arithmetic [16].

However, the greatest advantage of the modern fused multiply-add unit is also the greatest argument against its use. Fused multiply-add units remove the need for any single multiplication or add instruction and gain in performance by combining the two. As described already, any calls to stand-alone additions or multiplications require the insertion of a constant into the unit to emulate the desired arithmetic result. Due to the increased complexity of a fused multiply-add unit, such stand-alone instructions are subject to greater latencies than if they were processed in a single function arithmetic unit (i.e., floating-point adder or floating-point multiplier). For developers that do not wish to re-compile their existing code or for algorithms that are not amenable to implementation with a fused multiply-add instruction, the replacement of a floating-point adder and a floating-point multiplier with a fused multiply-add unit may be an unattractive endeavor.

A few possible solutions have been identified. Fused multiply-add designs have been proposed [17], [18] that use a floating-point adder dual-path scheme within a reduced latency fused multiply-add unit to allow floating-point additions to bypass the multiplier tree, approaching the latency of a normal floating-point adder instruction. Though such proposals have been made, they have yet to be implemented. Additionally, no study has yet been presented that identifies the relative costs of creating a floating-point unit capable of performing all three basic floating-point arithmetic instructions in hardware.

This paper presents a new architecture that builds hardware fused multiply-add functionality between a floating-point adder and a floating-point multiplier, creating a "bridge" that connects the two. The architecture is designed to re-use as much hardware as possible from both the floating-point adder and floating-point multiplier to minimize the area and the power consumption. This design is intended to provide an identification of the implementation costs in an arithmetic unit capable of floating-point addition, multiplication, and fused multiply-add instructions completely processed by hardware.

This paper presents the results of several custom circuit implementations, including a double-precision floating-point add unit, a double-precision floating-point multiply unit, a double-precision classical fused multiply-add unit, and a new double-precision bridge fused multiply-add unit. All circuits have been designed and implemented on AMD 65nm silicon on insulator technology using the 'Barcelona' native quad-core standard cell library to provide a realistic and fair comparison of performance and cost between the bridge architecture and standard floating-point arithmetic units.

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 design to design, but the basic micro architecture has remained invariant. The traditional architecture (denoted classic fused multiply-add unit in this paper) is shown in Figure 1.

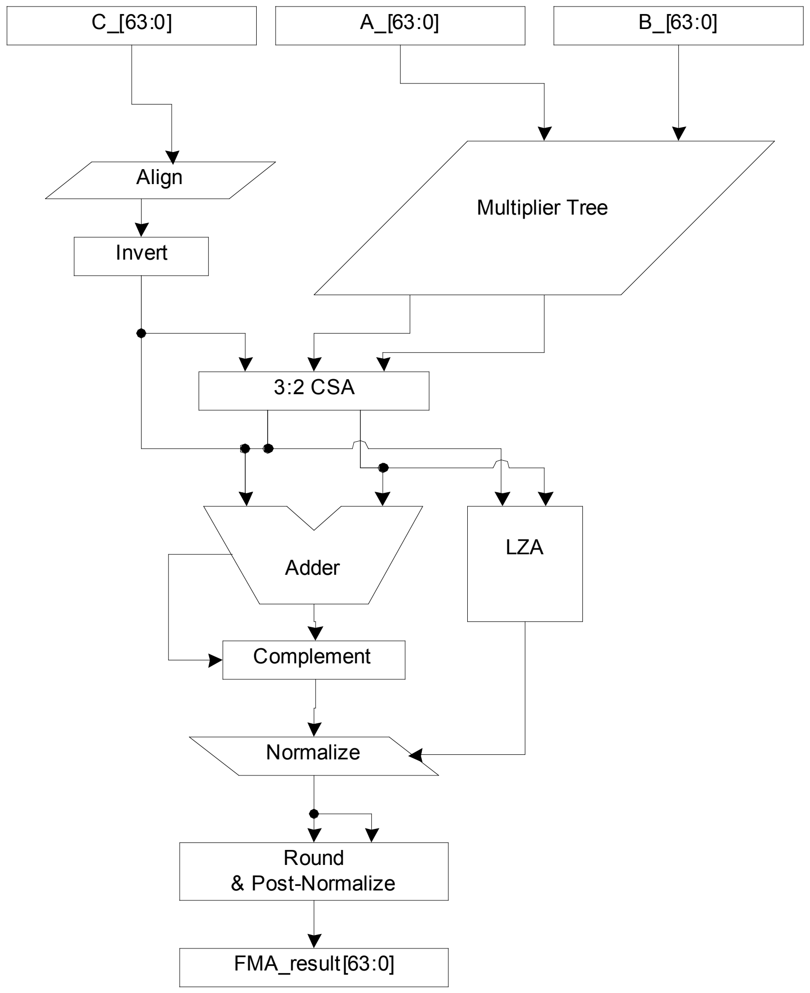

Figure 1. Classic Fused Multiply-Add Architecture.

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 in the multiply array to produce a 106 bit carry-save format result. Align the addend C to any point in the range of the 106-bit product, including 55 bits above the product (53-bit operand 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 to 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 161-bit result 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 ADDITION AND MULTIPLICATION IN A CLASSIC FUSED MULTIPLY-ADD UNIT

The classic fused multiply-add architecture shown in Figure 1 reduces the computation time of a multiply-add instruction executed on a floating-point multiplier followed by a floating-point adder in two ways: the bypass result bus and the rounding stage. A bypass result bus commonly connects the output of a floating-point multiplier to the input of a floating-point adder and consists of large interconnect routes that add RC latency to the fused multiply-add instruction path. Additionally, fused multiply-add instructions executed by floating-point multiplier units followed by floating-point adder units must use two rounding stages in total: one at the floating-point multiplier for the intermediate result and one at the floating-point adder for the final result.

A classic fused multiply-add arithmetic unit requires no bypass bus, as the results are all internal to the unit. Additionally, a fused multiply-add unit needs only one round stage for the final combined result. This removal of a bypass bus and the second round stage provides significant performance gains when processing fused multiply-add instructions.

However, the lack of a bypass bus and independent rounding unit show a performance loss when a fused multiply-add unit receives a stand-alone add or a stand-alone multiply instruction. Single floating-point multiply or add instructions have no early exit points anywhere in the entire unit's data-path. These single instructions are instead forced to use hardware that provides them no computational benefit. A floating-point multiply instruction must use steps 2, 4, 5, and 6 from above, even though they do no real work on the data. Likewise, a floating-point add instruction must flow through steps 1 and 2 from above, while also being subjected to larger alignment/normalization bit-lengths.

The use of a fused multiply-add unit in place of a floating-point adder and floating-point multiplier has yet another drawback. Due to their large area and power consumption, implemented fused multiply-add blocks typically replace the floating-point adder and floating-point multiplier entirely. This replacement removes the ability to have floating-point add and multiply instructions execute independently in different parallel units. For code that needs strings of floating-point adds and multiplies executed independently, the use of a fused multiply-add unit will reduce the throughput by 30% to 75%.

The degradation of performance for single floating-point add and multiply instructions in floating-point units using fused multiply-add units provides a very unattractive option for code developers who either do not wish to re-optimize existing software (commonly referred to as "legacy code") or for applications that require long series of single adds or multiplies. A fused multiply-add hardware unit that provides independent single floating-point add and multiply instruction executions would be an attractive option for such applications.

3.1 PROPOSED BRIDGE FUSED MULTIPLY-ADD ARCHITECTURE

The bridge fused multiply-add architecture is intended to find a low area, latency, and power consumption solution to the performance degradation of single floating-point adds and multiplies in current fused multiply-add units. Additionally, the bridge architecture is intended to provide a realistic study of the implementation costs involved when building an arithmetic unit capable of all three basic floating-point mathematical hardware instructions.

Figure 2 shows a high level block diagram of the bridge fused multiply-add architecture. The design begins with common floating-point multiplier and floating-point adder units capable of independent execution. Several blocks are added between the two arithmetic units, creating a "bridge" capable of carrying data from one unit to the other to perform a fused multiply-add instruction.

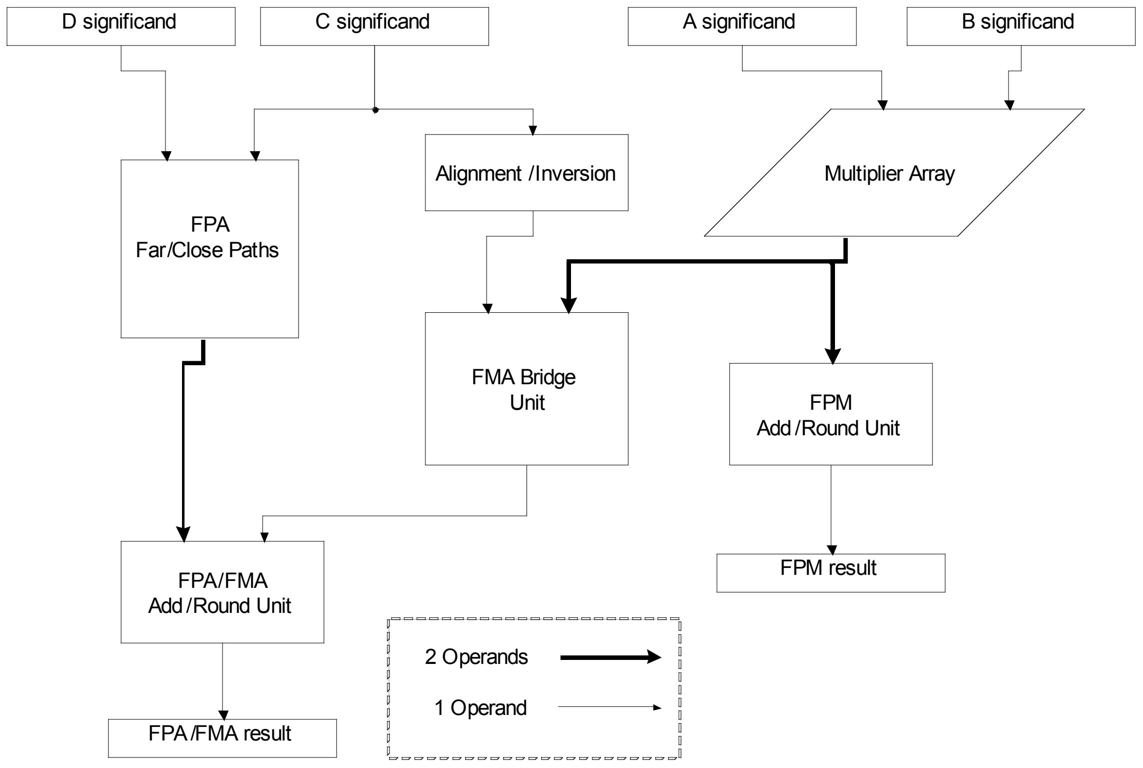

Figure 2. The Bridge Fused Multiply-add unit Block Diagram.

The bridge fused multiply-add architecture does not require an entire independent fused multiply-add hardware implementation. Pieces from both the floating-point multiply and floating-point add units are modified and reused for dual functionality. Specifically, the floating-point adder add/round stage is used for floating-point adds and fused multiply-adds, while the multiplier re-uses the single largest component block of any arithmetic unit, the multiplier tree. The remaining hardware requirements for a complete fused multiply-add instruction are implemented in the bridge unit, which is only powered on during a fused multiply-add instruction.

3.2 THE MULTIPLIER

The bridge fused multiply-add architecture uses a floating-point multiplier to process both stand-alone multiplications as well as the first stage of a fused multiply-add instruction. As shown in Figure 3, the double-precision multiplier unit takes two 64-bit operands as inputs. The significands are processed in a 53 x 53-bit multiplier, while the exponent and sign bits are processed in parallel. For a floating-point multiplication instruction, the multiplier array forwards the 106-bit sum and carry results to a floating-point multiplier rounding unit designed from a combination of several multiplication rounding schemes [19] - [21].

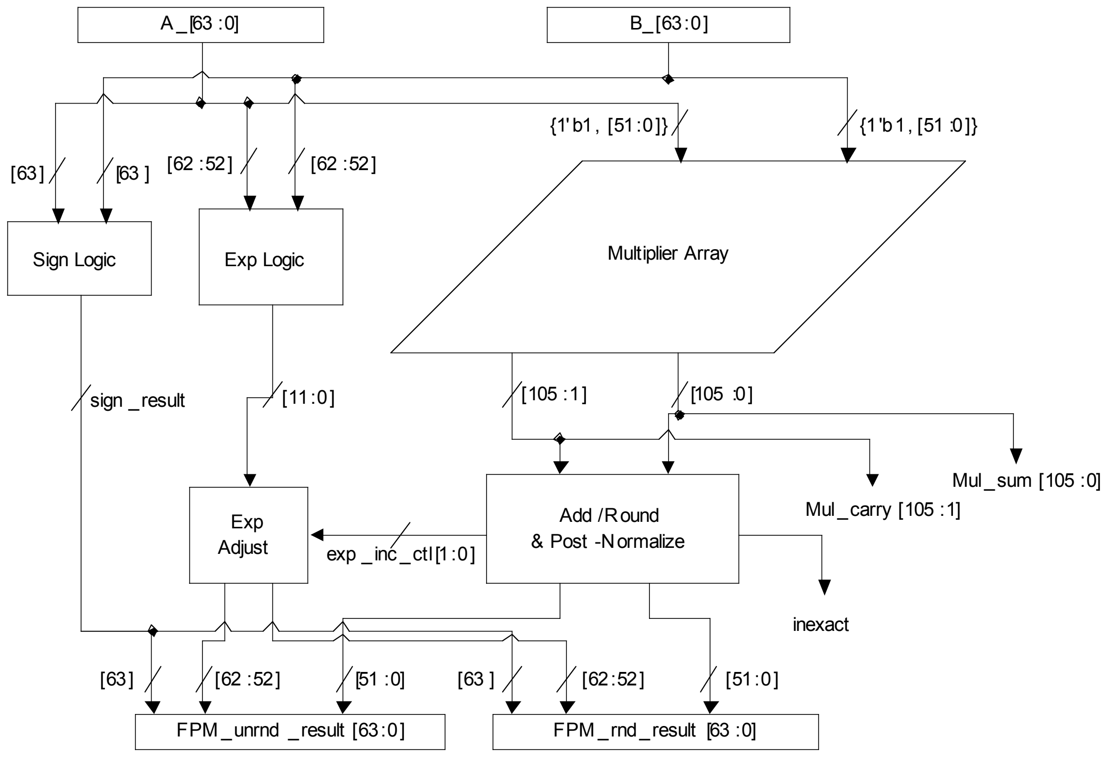

Figure 3. The Multiplier.

When the required operation is a fused multiply-add instruction, the unit begins execution in the same way as a floating-point multiply. However, when the multiplier tree produces a product in sum/carry format, it is passed to the bridge and the floating-point multiplier round element is shut down.

3.3 THE BRIDGE

The bridge unit is shown in Figure 4. The unit is essentially the classic fused multiply-add architecture described in Section 2 without the multiplier array, rounding, or post-normalization block. Instead, the bridge unit accepts the product from the floating-point multiplier and combines it with a pre-aligned 161-bit addend. The unit then proceeds with steps 2-6 of the fused multiply-add algorithm described in Section 2.

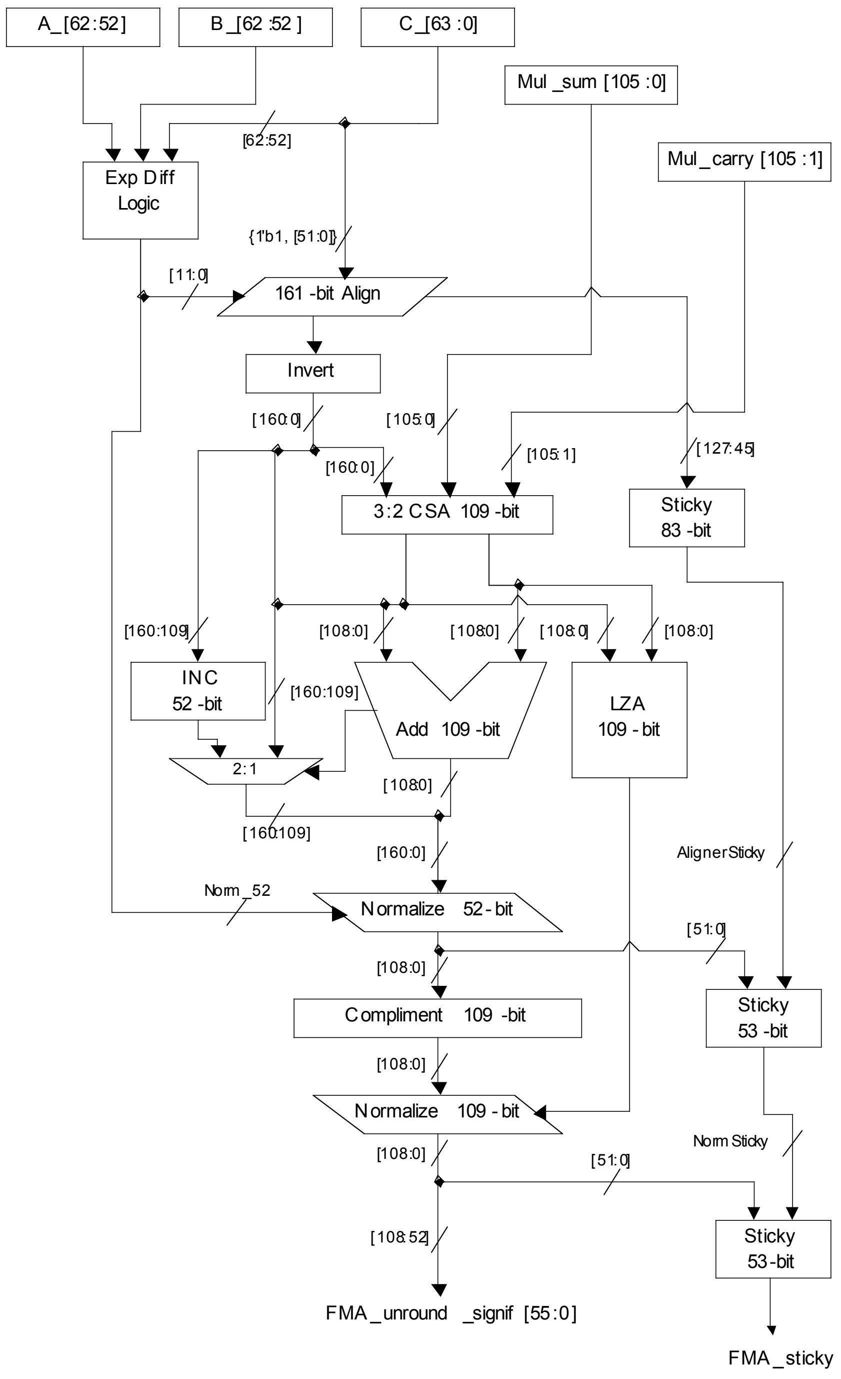

Figure 4. The Bridge.

3.4 THE ADDER

The bridge fused multiply-add architecture uses a common Farmwald [22] dual-path floating-point adder design to execute stand-alone addition instructions. As shown in Figure 5, the addition unit uses a far and close path to handle the two classical floating-point addition cases.

The far path, shown on the left side of Figure 5, is used to process input significands for either an addition or a subtraction if their exponents differ by more than 1. For this path, the significands of both inputs are passed to a swap multiplexer that awaits the results of a comparison of the exponents. When the larger significand is detected, it is anchored and the smaller significand is aligned until the exponents match.

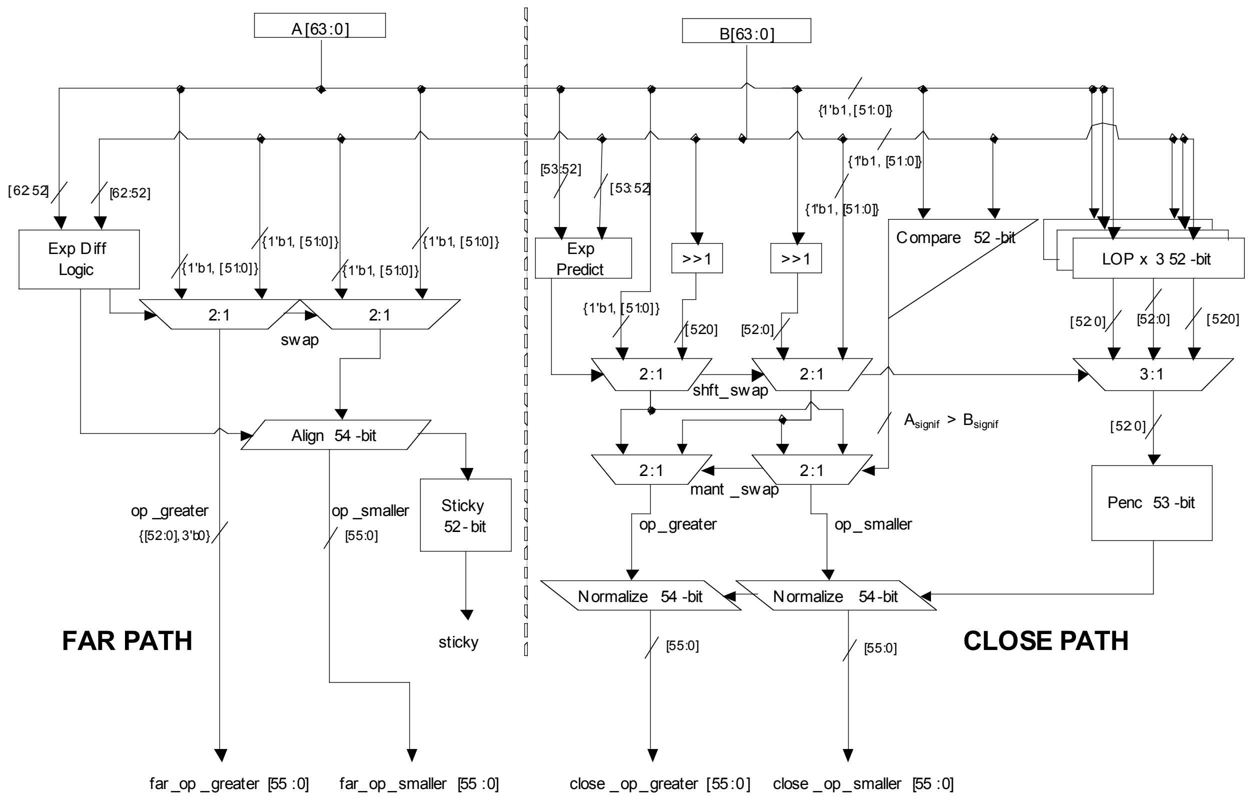

Figure 5. The Adder.

For cases of subtraction where the exponents are equal or differ by ±1, the input data are processed in the addition unit close path that is shown on the right side of Figure 5. The close path pre-shifts both input significands by one and inputs shifted and non-shifted operands to a swap multiplexer. Meanwhile, a comparator is used to determine the larger significand in the case of no exponent difference, all while three leading one predictors (LOP) operate in parallel on each possible exponent difference case.

The exponent prediction logic and significand comparator drive the select lines on several sets of swap multiplexers. The resulting LOP selection enters a 53-bit priority encoder and is reduced to a 5-bit normalization control. Both the larger and smaller significands in the close path are normalized by up to 54-bits, and the stage is complete.

The larger and smaller operands from both the far and close path exit the block and are forwarded to the FMA/FPA add/round stage for path merging, rounding, and instruction completion.

3.5 THE ADD/ROUND UNIT

The addition and rounding unit is designed to perform several roles. When a stand-alone floating-point addition instruction is required, the add/round unit acts as a common floating-point adder dual-path merge stage, selecting between the far and close path operands for inputs to the addition and rounding units. For fused multiply-add instructions, the same multiplexer used for a merge in the floating-point adder path is expanded to select the fused multiply-add unit un-rounded result. The second operand input to the addition and round units is passed a null string, as another operator is not needed for the fused multiply-add rounding completion.

The add/round unit is shown in Figure 6. It uses a combined add/round scheme suggested by several schemes seen in [23], [24]. The two selected input operands are passed to dual 59-bit adders producing a result and a result plus 2 (or plus 1 for subtraction). Providing these arithmetic results, as better explained by the literature [23], allows for an easy LSB fix-up, shift, post-alignment, and final result selection. The controls for the shifts come from the overflow bits of the adders, and the rounding selections are decided by combinational rounding logic.

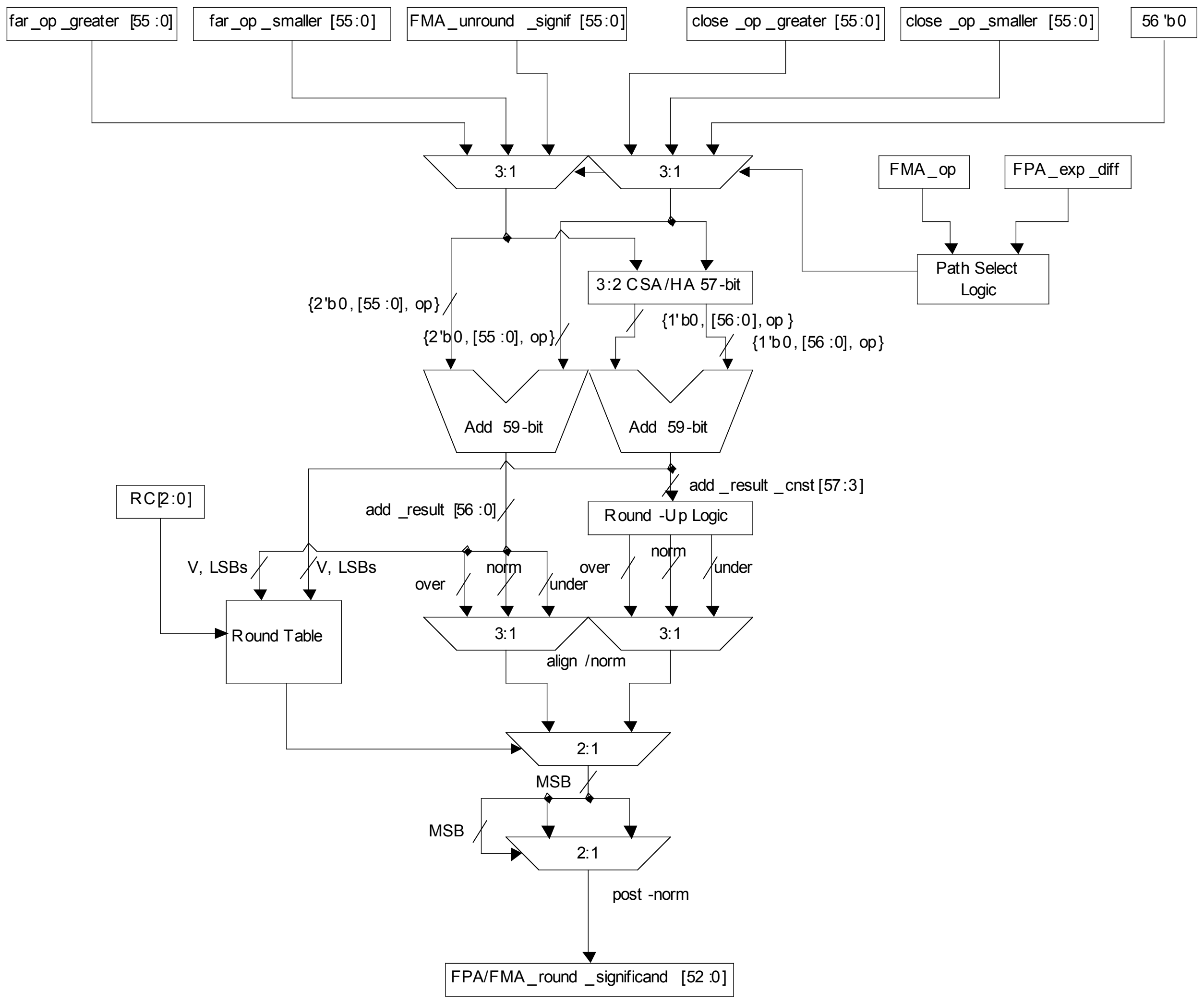

Figure 6. The Add/Round Unit.

4. IMPLEMENTATION

The various floating-point architectures have been designed in the AMD 65nm circuit design tool flow using 'Barcelona' native quad-core silicon on insulator standard cell libraries. To provide a fair comparison of floating-point add and floating-point multiply units against both the classic fused multiply-add unit and the proposed bridge fused multiply-add architecture, each block has been individually designed and implemented with equivalent components and identical design rules to ensure that the resulting circuits are suitable for a realistic and fair comparison.

All four floating-point units were first designed and simulated at a behavioral level using Verilog2k RTL code. These high level designs were compiled using Synopsys Chronologic VCS compilers and debugged using Novas Debussy waveform viewers. The behavioral code was verified through a combination of both custom test benches as well as 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 a floor planning tool, which includes detailed features such as power grids and clock-gater rows. There they were placed and resized based on cell fan-out and estimated routing parasitic capacitances. The final floor planned models were then timed using Synopsys Primetime software with 65nm transistor models and characterizations. The model routing information was generated using Steiner routing estimates for every net with data extracted using parasitic capacitance metal routing tables.

Final area results were calculated by measuring the floorplan dimensions of each circuit placement. Any unused space within the circuit's area did not adjust 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 architectural latency gains were process corner independent. Both timing runs ran from FPA/FPM/FMA input to FPA/FPM/FMA output and did not include internal pipelining.

Power calculations were performed in HSim using extracted SPICE netlists generated from the gate level descriptions and route parasitics from the floorplan. The simulations were given randomly generated input sequences and a clock period of 1500ps (to cover the latency of a single worst case bridge fused multiply-add unit 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

The bridge fused multiply-add unit implementation is compared in Tables 1-4 to a floating-point adder, floating-point multiplier, and classic fused multiply-add unit implementation over the categories of latency, area, and power consumption. The comparison results provide the absolute simulation calculations as well as the relative performance of all architectures in stand-alone floating-point add, multiply, and fused multiply-add instructions. The '**Δ**' rows provide the increase/decrease of an implementation's results relative to the first row of the table.

Table 1. Raw Results.

Design	Latency 1.3V 100 ˚C RVT	Latency 0.7V 100 ˚C LVT	Area 65nm AMD SOI	Peak Power 666 MHz 1.3V RVT
FPA_DP	946ps	2556ps	72,075 um^2	118mW
FPM_DP	701ps	1950ps	131,130 um^2	187mW
Classic FMA	1224ps	3363ps	186,930 um^2	416mW
Bridge FMA	**FPA:** 978ps **FPM:** 708ps **FMA:** 1454ps	**FPA**: 2625ps **FPM**: 1979ps **FMA**: 3973ps	283,650 um^2	**FPA:** 118mW **FPM:** 187mW **FMA:** 501mW

As shown in Tables 2-3, the bridge fused multiply-add architecture provides delay and power consumption comparable to stand-alone floating-point adders and floating-point multipliers for individual instructions. The bridge fused multiply-add architecture is about 40% larger than the combination of a stand-alone floating-point adder and a stand-alone floating-point multiplier. The bridge architecture is 30% to 70% faster and 50% to 70% lower in power consumption than a classic fused multiply-add unit when executing single-unit instructions. The bridge fused multiply-add architecture is about 50% larger than a classic fused multiply-add unit.

Table 2. Results Normalized to an FPA Stand-Alone Addition.

Design	Latency Difference 1.3V 100 ˚C RVT	Latency Difference 0.7V 100 ˚C LVT	Area Difference 65nm AMD SOI	Power Difference 666 MHz 1.3V RVT
FPA	0%	0%	0%	0%
Classic FMA **Δ**	29.4%	31.6%	159.4%	252.5%
Bridge FMA **Δ**	3.4%	2.7%	293.6%	0%

Table 3. Results Normalized to an FPM Stand-Alone Multiplication.

Design	Latency Difference 1.3V 100 ˚C RVT	Latency Difference 0.7V 100 ˚C LVT	Area Difference 65nm AMD SOI	Power Difference 666 MHz 1.3V RVT
FPM	0%	0%	0%	0%
Classic FMA **Δ**	74.6%	72.5%	42.6%	122.5%
Bridge FMA **Δ**	1.0%	1.5%	116.3%	0%

As shown by Table 4, when compared to the combination of a floating-point adder and floating-point multiplier, the bridge fused multiply-add unit still shows about a 12% performance gain for fused multiply-add instructions. However, the bridge fused multiply-add unit requires about 40% more area than a floating-point adder and floating-point multiplier combination and 64% more power for fused multiply-add instructions.

Table 4. Results Normalized to an FPA and FPM Fused Multiply-Add.

Design	Latency Difference 1.3V 100 ˚C RVT	Latency Difference 0.7V 100 ˚C LVT	Area Difference 65nm AMD SOI	Power Difference 666 MHz 1.3V RVT
FPA + FPM	0%	0%	0%	0%
Classic FMA **Δ**	-25.7%	-25.4%	-8%	36.4%
Bridge FMA **Δ**	-11.7%	-11.8%	39.6%	64.3%

6. CONCLUSION

A new architecture for the design and implementation of a fused multiply-add unit with high performance stand-alone floating-point addition and multiplication instructions has been presented. The bridge fused multiply-add architecture implements a fused multiply-add instruction by adding a "bridge" in between a standard floating-point adder and floating-point multiplier, all while re-using components from both to minimize the implementation costs. The bridge fused multiply-add unit shows almost identical latency and power consumption for addition and multiplication instructions as compared to typical stand-alone floating-point arithmetic units at the cost of increased area and a higher-performance fused multiply-add operation as compared to the stand-alone units. It does exhibit lower-performance for the fused multiply-add operation as compared to the classic fused multiply-add unit.

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, No. 1, pp. 59-70, 1990.

[2] E. Hokenek, R. Montoye and P.W. Cook, "Second-Generation RISC Floating Point with Multiply-Add Fused," *IEEE J. Solid-State Circuits*, Vol. 25, No. 5, pp. 1207-1213, 1990.

[3] C. Hinds, "An Enhanced Floating Point Coprocessor for Embedded Signal Processing and Graphics Applications," *IEEE 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- 101-4, 2004.

[5] E. N. Linzer, "Implementation of Efficient FFT Algorithms on Fused Multiply-Add Architectures," *IEEE Transactions on Signal Processing*, Vol. 41, No. 1, pp. 93-107, Jan, 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, No. 6, 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, No. 5, 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*, 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- 1, Nov. 2001.

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

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

[17] H. Sun and M. Gao, "A Novel Architecture for Floating-Point Multiply-Add-Fused Operation", *Proceedings of the 2003 4th International Conference on Information, Communications and Signal Processing and the 4th Pacific Rim Conference on Multimedia*, Vol. 3, pp. 1675-1679, Dec, 2003.

[18] 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, June 2005.

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

[20] R. K. Yu and G. B. Zyner, "167 MHz Radix-4 Floating-Point Multiplier," in *Proc. 12th Symp. Computer Arithmetic,* pp. 149-154, 1995.

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

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

[23] N. Quach, M.J. Flynn, "An Improved Algorithm for High-Speed Floating Point Addition," *Technical Report CSL-TR-90-442*. Computer Systems Laboratory, Stanford Univ., Aug. 1990.

[24] A. Naini, A. Dhablania, W. James, and D. Das Sarma, "1 GHz HAL Sparc64 Dual Floating Point Unit with RAS Features," *Proceedings of the 15th Symposium on Computer Arithmetic,* pp.173-183.