A High-Performance Double Precision Accumulator

Krishna K. Nagar and Jason D. Bakos

Dept. of Computer Science and Engineering, Univ. of South Carolina
Columbia, SC 29208 USA
{nagar,jbakos}@engr.sc.edu

Abstract—**The accumulation operation $A_{new} = A_{old} + X$ is required for many numerical methods. However, when using a floating-point adder with pipeline latency α, the data hazard that exists between A_{new} and A_{old} creates design challenges for situations where inputs must be delivered to the accumulator at a rate exceeding $1/\alpha$. Each of the techniques proposed to address this problem requires either static data scheduling or overly complex micro-architectures having multiple adders, a large amount of memory, or control overheads that force the accumulator to operate at a diminished speed relative to the adder on which it is based. In this paper we present a design for a double precision accumulator that achieves high performance without the need for data scheduling or an overly complex implementation. We achieve this by integrating a coalescing reduction circuit within the low-level design of a base-converting floating-point adder. When implemented on our Virtex-2 Pro 100 FPGA, our design achieves a speed of 170 MHz.**

I. Introduction

The accumulation operation must be performed in any special-purpose architecture that performs a computation that requires a summation. When used as a component of a micro-architecture that supplies a new value to the accumulator every clock cycle, the designer cannot use a simple feedback-based accumulator circuit if the adder has a latency greater than one cycle, since the latency prevents the adder from providing the current sum before the next value to be accumulated arrives. To make matters worse, in many applications the incoming values belong to different accumulation sets and there is no separation between values belonging to different sets. This further complicates the accumulator design.

FPGA kernel designers have dealt with this problem using a number of different methods. Older designs used a static approach, where inputs were delivered to the accumulator in an ordering where the values belonging to different accumulation sets were interleaved according to the latency of the adder [1]. Newer designs use a dynamic approach. Prasanna's group at the University of Southern California has written several seminal papers in this area [2,3,4]. Their most recent design requires two memories of size α^2 and a significantly complex controller that makes the reduction circuit operate more slowly than the adder upon which it is based. An improved single-adder reduction architecture was later developed at the University of Twente [5]. This architecture reduced the memory requirement to $3\alpha + \alpha\lceil \lg \alpha \rceil + 2$ but still required complex control.

Both of these designs are based on instancing pre-made floating-point adders into a top-level reduction architecture. In an alternative approach, which is limited to single-precision, the adder itself is changed such that its de-normalization and significand addition step are designed to have a single cycle latency and a feedback loop is formed over only this stage. Since the only portions of a floating-point adder that need be involved in the accumulator's feedback loop are the de-normalize and significand addition, this turns the adder into an accumulator. The other aspects of the adder, specifically those that deal with IEEE 754 formatting, need not be included in the adder data path. In order to make this approach practical, the designer must minimize the latency across both the de-normalize (composed of a comparison and subtraction of the exponents) and the significand addition (an integer addition). Intel and a group from Princeton accomplished this by increasing the integer adder width while decreasing the width of the exponent comparator by converting the significand from base-2 to base-32 [6,7].

Our goal is to design a double precision accumulator that requires minimal memory and control logic such that its speed is limited only by the speed of the significand adder. Because double precision requires a wider exponent compare and significand addition, we pipeline this portion of the architecture instead performing this in a single stage. To solve the resultant internal data hazard, we apply a simplified reduction technique within the adder design.

II. Accumulator Architecture

Our top-level accumulator architecture is shown in Figure 1. As shown in the figure, the first two stages are used to condition the incoming value. The base conversion step (box 1) converts the incoming value from base 2 to an arbitrary base, which is set as a "generic" parameter in our VHDL. For base *b*, this step performs the following:

1. adds a 1-bit to the left-hand side of the 52-bit significand value (the implied leading binary digit to the left of the decimal point),
2. shifts the significand value to the left by the value stored in the low order lg *b* bits in the exponent field (note that this effectively adds *b* - 1 bits to the width of the significand),
3. strips the lower lg *b* bits from the exponent, and
4. adds a sign bit and carry-out bit "00" to the left side of the resultant (53 + *b* - 1)-bit base-*b* significand value, resulting in (54 + *b*) total bits.

The next stage performs an arithmetic negation of this value if the original sign bit was set to one (box 2).

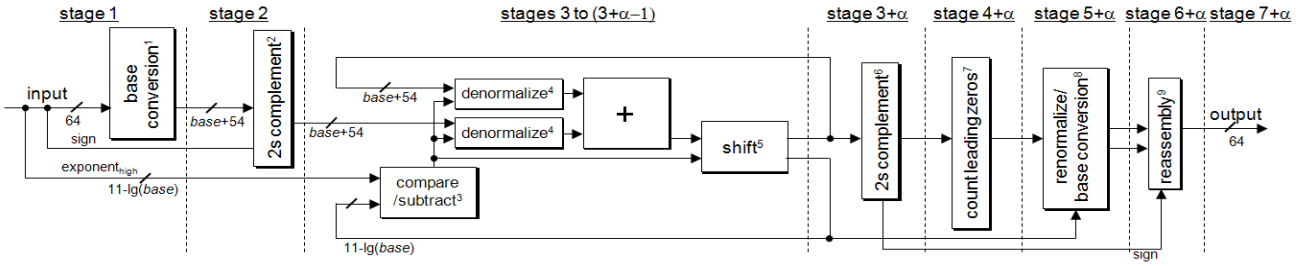

Fig. 1. Top-level design of accumulator architecture. Alpha represents the pipeline latency of de-normalize/addition datapath.

The third stage is where the de-normalize and significand addition begins. This is comprised of the following steps:

1. compare the high-order 11-(lg *b*) bits of both exponents, *exp1* and *exp2* (corresponding to base-*b* significands *sig1* and *sig2*),
2. if *exp1* > *exp2*, shift *sig2* to the right by *b**(*exp1-exp2*) bits, else shift *sig1* to the right by *b**(*exp2-exp1*) bits,
3. add the resultant *sig1* and *sig2*, and
4. if the addition caused the carry out bit of the sum to be set, add one to max(*exp1*,*exp2*) and shift the sum *b* bits to the right.

This step involves sequential operations on both the high order bits of the original exponent and the base-converted significands. Larger values of *b* will result in lower latency exponent operations but a wider and thus higher latency integer addition, while lower values of *b* will result in wider and thus higher latency exponent operations and lower latency integer addition.

Table 1 shows this trade-off as the base *b* is increased. The single-precision accumulator designs from the literature perform these operations in one cycle and they chose 32 as the base without providing any justification or analysis of why this value was chosen. Since our accumulator is double precision, we performed a synthesis-based analysis to determine how the base value affects the resultant speed of the adder. We describe this analysis in the next section.

The remaining stages are used to re-condition the base-converted sum into IEEE 754 format. Box 6 computes the absolute value of the sum, box 7 counts the number of leading zeros, and box 8 uses this information to shift the significand and adjust the exponent in order to convert the significant back to base 2 and to re-normalize. The last stage repackages the value into IEEE 754 format.

In order to determine the base value that provides the highest performance, we synthesized versions of the design shown in Figure 1 over a range of base values. For each base value, we also added delays to the outputs of the de-normalize/add stage and enabled re-timing and pipelining in the synthesizer to give it the ability to distribute this step across multiple pipeline stages. In other words, we added cycles of latency to the de-normalize/add step to improve the overall pipeline speed of the accumulator. Note that in addition to the exponent comparison, exponent subtraction, and significand addition, there are also two shifters involved in this step (de-normalize and renormalize for a carry-out).

In our analysis the versions of the design having a de-normalize/add latency greater than one are not functionally correct without the addition of the reduction features described later in this paper. However, since this analysis is for timing purposes only and the reduction features will require only minimal timing overhead, we do not include them in this analysis. Our analysis included base values ranging from 4 to 512.

Figure 2 shows the results of the analysis. We used Synplify Pro 8.8.0.4 as the synthesizer and targeted both our in-house Virtex-2 Pro 100 FPGA as well as the Xilinx Virtex-5 LX 330T for comparison. The Virtex-2 Pro 100 design achieves its highest pipeline speed with a pipeline depth of 3 and base of 128, while the Virtex-5 LX 330T design achieves its highest pipeline speed with a pipeline depth of 3 and a base of 64.

TABLE I
EXPONENT COMPARISON WIDTH VS. ADDER WIDTH

Base	Exponent Compare	Adder Width
32	6	86
64	5	118
128	4	182
256	3	310
512	2	566

III. REDUCTION CIRCUIT

The reduction circuit must reconcile the data hazard created by the three cycle latency of de-normalize/add step, since each input to this step depends on the most recent output. Note that any of the previously designed reduction circuits from the literature would fulfil this requirement. However, these previous reduction circuits were designed for much longer pipelines (i.e. an entire floating-point adder pipeline as opposed to only the de-normalize/add pipeline). In this case, we only need a reduction circuit to operate over a three-stage pipeline, which gives us the opportunity to design a reduction circuit that is significantly less complex than previous designs.

The goal of our reduction circuit design is to make its implementation as simple as possible in order to not impose any additional timing overhead on the overall adder pipeline. In other words, the addition of the logic required for the reduction circuit should not shift the critical path from the de-normalize/add stages. In previously reported work, the

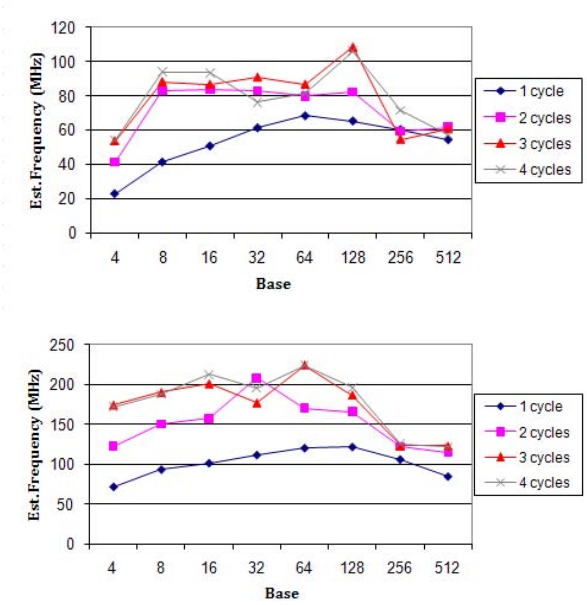

Fig. 2. Synthesis results for Virtex 2 Pro 100 (top) and Virtex-5 LX 330T (bottom), as operating frequency versus base value.

control and memory overheads required by the reduction circuit scale with the depth of the pipeline. The goal of our design is to keep the memory requirements constant, needing only to scale the control logic for the pipeline depth.

After a sufficient number of clock cycles have passed reducing a single input set, the reduction circuit operates in steady-state mode, where it routes the current input value and the output of the pipeline back into the input of the pipeline. In this operating state, the pipeline contains α partial sums, where α is the pipeline depth. When there is a change in input set, the pipeline must take a series of actions to coalesce these partial sums while still accepting values from the next input set.

As shown in Figure 3, our reduction circuit design requires a single input buffer and a single output buffer. The inputs to the pipeline can be routed according to the following four different configurations:

- **Configuration A:** buffer the incoming value, route the buffered output value and the output currently being produced by the pipeline back into the pipeline. For a pipeline depth of α, this must occur once for every internal node of a binary tree having α leaves, equalling α- 1 occurrences. To ensure that the buffer depth may be limited to one, the value in the input buffer must be consumed (using configuration C) once between each instance of configuration A.
- **Configuration B:** add the incoming value with the value currently being produced by the pipeline. This is the "steady-state" configuration, and is used when accumulating the current input set into α partial sums.
- **Configuration C:** add the buffered input value with the incoming input value. This occurs during cycles when the output of the pipeline need not re-enter the pipeline. This includes the cycles where the pipeline output is buffered (which must occur once before the architecture enters configuration A) and the cycles where an input set is reduced to a final sum (which occurs once per input set).
- **Configuration D:** add the incoming value with zero. This only occurs one time per input set, prior to the first time an input is buffered.

As shown in Table 2 for a pipeline depth of $\alpha = 3$, starting with the first cycle where the incoming value belongs to a new input set, the controller will instruct the reduction circuit to cycle through a deterministic series of configuration changes for the following eight cycles that will reduce the previous input set to a single sum while continuing to accept values from the new input set. In the table, $\alpha 1$, $\alpha 2$, and $\alpha 3$ represent the partial sums from the previous input set.

The controller is implemented as a single 9-state FSM, where all state transitions are unconditional except for the input state where the next input set (incoming from stage 2 of the accumulator pipeline) differs from the current input set. This is detected by comparing the input set from stage 3 and stage 2 in the top-level accumulator pipeline.

Routing is performed with a 2-input mux before the first input and a 3-input mux before the second input to the de-normalize/add pipeline. Note that each input value consists of a (54 + b)-bit significand and a (11-lg b)-bit upper exponent value. The controller also raises the data_valid flag to indicate the output sum is valid for each input set.

For this reduction technique, there is a minimum set size that must be enforced in order to allow for the coalesce process for the previous input set to finish before the current set ends. For a pipeline depth of α, the minimum set size is $\alpha\lceil \lg\alpha + 1\rceil - 1$ cycles, since after each α-cycle pass, there are half the number of partial sums in the pipeline.

IV. Accumulator Characteristics

The total latency of the accumulator is 7 cycles for the base conversion and IEEE 754 overheads plus 8 cycles for the reduction, totalling 15 cycles. The synthesis-based timing analysis described in Section III did not include routing delays. To determine the actual performance of the accumulator, we placed and routed the fully-designed base-32, base-64, base-128, and base-256 accumulators, including the reduction circuit, and placed each inside a system-specific wrapper that

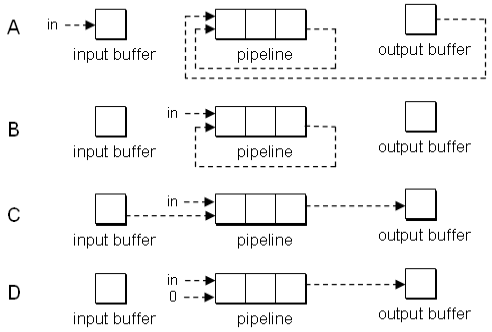

Fig. 3. Configuration states for the reduction circuit.

TABLE II
EXAMPLE OF THE REDUCTION CIRCUIT OPERATING OVER A PIPELINE OF DEPTH 3.

Clock cycle	Accum. input	Input buffer	Adder pipeline			Output buffer	Notes
0							Configuration B (Steady-state)
1	B1		α3	α2	α1		Configuration D Set A complete, adder pipeline full
2	B2		B1+0	α3	α2	α1	Configuration A
3	B3	B2	α1+ α2	B1+0	α3		Configuration C
4	B4		B2+B3	α1+ α2	B1+0	α3	Configuration B
5	B5		B1+B4	B2+B3	α1+ α2	α3	Configuration A
6	B6	B5	α1+ α2+ α3	B1+B4	B2+B3		Configuration B
7	B7	B5	B2+B3+B6	α1+ α2+ α3	B1+B4		Configuration B
8	B8	B5	B1+B4+B7	B2+B3+B6	α1+ α2+ α3		Configuration C Set A accumulation complete, use this cycle to clear input buffer
9	B9/C1		B5+B8	B1+B4+B7	B2+B3+B6		Configuration B/D Earliest valid cycle for input set C to begin

allows each accumulator to be instanced on the FPGA and the host to send it inputs and sample its outputs, through FIFOs that are accessible by programmed I/O calls from the host. Using this technique, we incrementally increased the accumulator pipeline speed, and after each increase we compare the results from the accumulator with results computed on the host. Using this technique we can determine the maximum speed for each base value on our Virtex-2 Pro 100 FPGA on our Annapolis Micro Systems WildStar-II Pro card.

The performance and resource usage results are shown in Table 3. The resource usage was measured by placing and routing the accumulator only, while the performance results were measured with additional host-FPGA interface logic. The performance of each circuit is higher than the synthesis results indicated. Contrary to the results shown during our synthesis analysis, the base-64 version of the accumulator achieves the highest speed. We assume this is due to inaccuracies in the synthesizer estimate. The resource usage results show that these circuits, except for the base-256 version, use significantly less resources than the accumulators from the literature.

TABLE III
ACCUMULATOR RESOURCE AND PERFORMANCE RESULTS

Base	Slices	Maximum Actual Operating Freq.
32	1884	160 MHz
64	2045	170 MHz
128	2986	130 MHz
256	5336	125 MHz

V. CONCLUSION

In this paper we describe a new design technique for high performance, low resource double precision accumulators. Our approach combines elements from two previous techniques. The first is the use of base-converted adders to expose a reduced-latency addition operation, as opposed to basing the accumulator feedback around a full floating-point adder architecture. The second technique is to use a simplified version of the reduction architecture described in several recent publications.

We tested our accumulator on a Virtex-2 Pro 100 FPGA on our Annapolis computing card. Through these tests, we have achieved an observed maximum speed of 170 MHz and a minimum resource usage of 1884 slices.

VI. ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under Grant Nos. CCF-0844951 and CCF-0915608.

REFERENCES

[1] M. deLorimier, A. DeHon, “Floating-point sparse matrix-vector multiply for FPGAs,” Proc. 13th ACM/SIGDA Symposium on Field-Programmable Gate Arrays (FPGA 2005).

[2] L. Zhou, V. K. Prasanna, “Sparse Matrix-Vector Multiplication on FPGAs,” Proc. 133h ACM/SIGDA Symposium on Field-Programmable Gate Arrays (FPGA 2005).

[3] L.Zhuo, V. K. Prasanna, “High-Performance Reduction Circuits Using Deeply Pipelined Operators on FPGAs,” IEEE Trans. Parallel and Dist. Sys., Vol. 18, No. 10, October 2007.

[4] Jason D. Bakos, Krishna K. Nagar, "Exploiting Matrix Symmetry to Improve FPGA-Accelerated Conjugate Gradient," 17th Annual IEEE International Symposium on Field Programmable Custom Computing Machines, April 5-8, 2009.

[5] M. Gerards, “Streaming Reduction Circuit for Sparse Matrix Vector Multiplication in FPGAs”. Master Thesis, University of Twente, The Netherlands, August 15, 2008.

[6] S. R. Vangal, Y. V. Hoskote, N. Y. Borkar, A. Alvandpour, “A 6.2-GFlops Floating-Point Multiply-Accumulator With Conditional Normalization,” IEEE Journal of Solid-State Circuits, Vol. 41, No. 10, Oct. 2006.

[7] Z. Luo, M. Martonosi, "Accelerating Pipelined Integer and Floating Point Accumulations in Configurable Hardware with Delayed Addition Techniques," IEEE Transactions on Computers, Vol. 49 No. 3 March 2000.