Presented at: **34th Annual Conference on Signals, Systems, and Computers, Asilomar, Pacific Grove, October, 2000.**

The IEEE Rounding for Multiplier with Redundant Operands

M. I. Ferguson
fergie@cs.ucla.edu

Miloš D. Ercegovac
milos@cs.ucla.edu

Computer Science Department
University of California, Los Angeles

Abstract

We present a design for a multiplier with redundant operands which conforms to the IEEE-754 floating-point standard. The design consists of a multiplier core and a rounding unit which conforms to the rounding modes specified by the IEEE standard and introduces a novel technique for calculating the sticky bit directly from the operands. We simulate a single-extended precision design and evaluate the delay versus a conventional design to be roughly equivalent, with extra savings possible with minor modification. The area is also estimated to be roughly 15% larger than a conventional design of the same precision.

1 Introduction

The use of redundant representations in the design of arithmetic units such as multipliers, dividers, and accumulators is a well-established technique for reducing the critical path in implementation. However, its use has been internal and the inputs/outputs are assumed to be in conventional form. Conversion of the redundant result to a conventional one is a time-consuming step since it involves propagation of carries over the working precision. To avoid the conversion step arithmetic schemes which accept redundant operands and produce redundant results have been proposed [1], [2] [3]. The authors have developed a radix-4 fully redundant multiplier core [4] and extend its design in this paper. To perform a relevant comparison, we comply with the IEEE Floating-Point Standard [5].

The main thrust of this research is to understand the effects of using redundant form operands in floating-point multiplication. In order to evaluate this, a baseline design must be developed in parallel, so that contrast can be achieved. The baseline design is a floating-point multiplier using two 32-bit conventional form operands of IEEE single-extended precision. The scope of this paper is restricted to the multiplier core and rounding unit, as the exponent adjustment is not representation specific. The design of the multiplier and specification of rounding modes is first developed using operand-independent blocks in section 2, then representation dependent parameters are discussed in section 3 and those contrasts are evaluated in section 4.

2 Baseline Design

The multiplication operation is denoted $z = x \times y$ where x and y are the operands whose significands are represented by X and Y, respectively and are in the range $[1, 2)$. The m-bit operands are of the form $M = M_0.M_1M_2...M_{m-1}$. The final result is obtained by performing normalization and rounding on an intermediate result P, $Z = \mathrm{RN}(P) = \mathrm{RN}(X \times Y)$ where RN() indicates a fused normalize and round operation. The intermediate result P, is in Carry/Save form

$(P^c_{0..2m-3}/P^s_{-1..2m-2})$[1]: Since $1 \leq X, Y < 2$ the result is in the range$[1, 4)$. Three special digit positions are recognized, $L = P_{m-1}$, $G = P_m$, and $T = \texttt{OR}(P_{m+1} \cdots P_{2m-2})$ where the indices indicate bit location **after** normalization. L specifies the digit of least precision, G is a guard digit and T is the sticky bit which indicates the tie condition where the result is exactly between two m-digit representable values. An overview of the carry/save form of the multiplier can be found in figure 1, the conventional form differing in that there is no "radix convert" block as the multiplicand X is used directly. Another minor difference is that the result index 0 is implied with a normalized conventional number, but not in carry/save form because there could be a 2 in position 1.. The outputs in the figure indicated by the wide arrows are busses too complicated to show schematically. The **A** and **B** busses from the `radix convert` block are the carry/sum bits as described in section 3. The bus coming from the recode block are the digits of the recoded multiplier in unary code (+1, -1, +2, -2) which encode the range of recoded multiplier digits $\{-2, \ldots, 2\}$, zero indicated as the null set of those four bit lines.

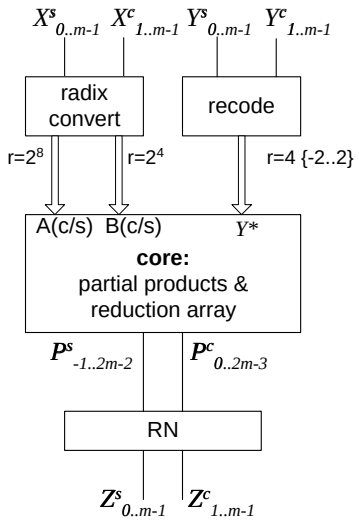

Figure 1. Overview of the carry/save form of the multiplier.

[1] The index can range from index 0 to $2m-2$ if no recoding is done on the multiplier.

Core: The operands for the core are multiplicand X and multiplier Y. The multiplier is recoded to the set $\{-2, \ldots, 2\}$. Partial products are reduced using the TDM method developed in [6]. The multiplicand is partially recombined as in [4] to reduce the complexity of the reduction array, this is discussed further below.

Rounding Unit: There are three rounding modes specified by the IEEE 754 include **Round to zero**, **Round to ±infinity** and **Round to nearest (tie even)**. We concentrate here on Round to nearest as it is the default rounding mode. In this mode we add rnd to L where $rnd = G$ `AND` $(T$ `OR` $L)$. Equivalently a 1 is added to G and then L is forced to zero if $T = 0$. However, since the index of these digits isn't known until after normalization, a 2 might have to added at G instead. The choice between these is determined by whether or not P overflowed to the range $[2, 4)$. The two choices can be pre-computed, the selection between them being performed later based on the carry bit into position m, denoted C_m. The method used to combine the normalization, and adjustment of L is based on original work by Santoro [7] and expanded upon in [8]. The method for determining the sticky bit T is to subtract 1 from the carry/save output in the lower half of P by adding a string of 1s. If the result remains -1, then $T = 0$. This leads to a simple tree of `OR` gates to determine T with the inclusion of P_m in the case of P_{ovf}. The structure for determining C_m is a simple carry-lookahead design with all the logic for computing intermediate sums and carries removed. An overview of the carry/save output version of the rounding unit can be seen in figure 2, the conventional form varies only in that the outputs of the `Pre-Compute` block, the `Mux` and the `R1 Shift` are not in carry/save form.

3 Representation Specific Issues

The use of redundant form operands has several effects on both the multiplier core as well as the RN block.

Partial Recombination: Naively, since the multiplicand has two rows of bits (X_s,X_c), there must be twice as many partial products generated, which then feed into the reduction array. However,

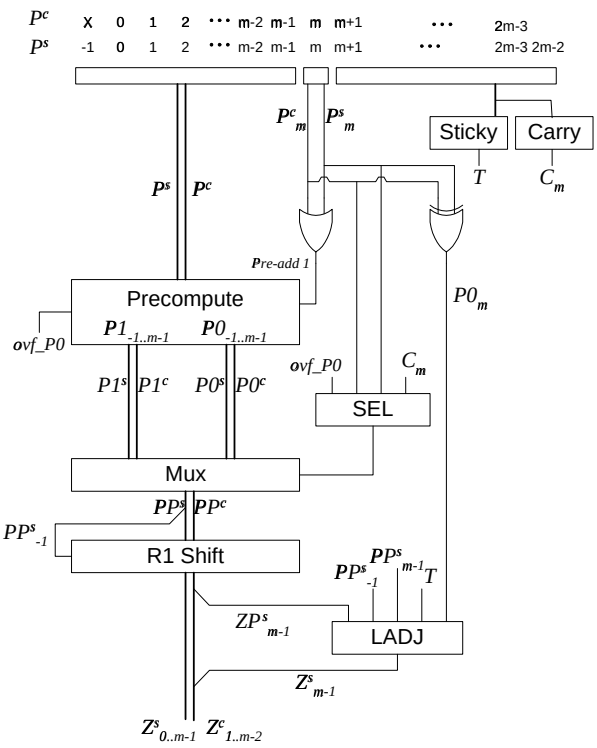

Figure 2. The block diagram of the Rounding module.

we use a technique which introduces strings of zeros in the Carry vector by partially recombining the radix-2 C/S to a radix-2^k C/S form. In this form, X_s^* is the same size and X_c^* consists of m/k blocks of $k-1$ zeros separated by carry bits . This reduces the number of bits in the carry vector of the partial products. Our design uses a special version in which $k = 4$ for the first two partial products and $k = 8$ for the remaining 14 rows, which has been shown to produce good results [4]. The X_c^* generate approximately 20% more inputs to the array.

Recoding of Y The multiplier Y is recoded in both versions of the multiplier to radix-4 $\{-2, \ldots, 2\}$. There is a minor difference in gate delay between recoding C/S and 2's complement.

Intermediate Product as Output:

In order to accommodate the special rounding technique suggested in [7], [8] to remove C_m from the critical path an extra bit "pre-add 1" is passed to bit L. We massage P differently depending on the design.

Conventional case A row of m half-adders (HA) is used before a full precision compound adder, see figure 3a).

Carry/Save case . A row of $(m-2)$ HA is augmented by an optimized circuit which absorbs the "pre-add 1" bit into the two least significant digit positions see part b) of the figure. In this case there is no need to perform a full propagate addition. However, there is the case that P overflows to range $[2, 4)$ and detection is done with a module identical in structure to that which is used to determining C_m.

4 Comparison Results

The comparison between the delay of the conventional form multiplier and the C/S design is done by comparing the delays of blocks along the critical path. The delay of each block is determined by performing layout in a circuit design tool[2] using the SCMOS library of gates. A Spice netlist of that circuit is then extracted and T-Spice is then used to model the circuit using a model file obtained from Mosis. The technology file used is AMOS14TB with a feature size of $0.6\mu m$. This analysis takes into interconnect complexity at a rudimentary level, but it should be noted that a more detailed study would optimize the blocks to take into account the varying arrival times across inputs to blocks.

The comparison between blocks is shown in table 1. It should be emphasized that the main design difference between the two schemes is that there is no full-precision addition performed in the `Pre-compute` block for the carry/save form, consequently `Mux` and `R1 Shift` are done on carry/save form, but these can operate bit-parallel and so only add fractionally more delay due to increased routing.

Another comparison can be made with respect to area, those results are shown in table 2

[2]Tanner L-Edit v8.1.

Module	Conventional	Carry/Save
Recode	1	7
Radix convert	-	6
Generator	3	10
Reduction Array	30	38
Sticky	6	6
Carry	5	5
SEL	1	1
pre-compute	24	5
MUX	1	1
R1Shift	1	1
LADJ	1	1
Critical Path	60	63

Table 1. Results of T-Spice calculations of blocks. Units are ns.

Module	Conventional	Carry/Save
Recode	1.4	4.8
Radix convert	-	4
Generator + Array	62	73
Sticky	1.3	1.3
Carry	0.7	0.7
SEL	0.1	0.1
pre-compute	6	2.6
MUX	0.9	1.4
R1Shift	0.9	1.4
LADJ	0.1	0.1
Total	73	89

Table 2. Area of blocks given in millions of square Locator Units (LU^2.) The generator and reduction array are combined because if the reduction array is generated as a separate block there is tremendous overhead due to the large number of inputs

X_0	X_1	X^-	X^+
0	1	1.0	1.0
0	2	1.0	1.5
1	0	1.0	1.5
1	1	1.5	2.0^-

Table 3. X^- and X^+ indicate the min and max values attainable by X. 2.0^- indicates a value close to but less than 2.0.

X^-	X^+	Y^-	Y^+	P^-	P^+	N
1.0	1.0	1.0	1.0	1.0	1.0	0
1.0	1.0	1.0	1.5	1.0	1.5	0
1.0	1.0	1.5	2.0^-	1.5	2.0^+	?
1.0	1.5	1.0	1.0	1.0	1.5	0
1.0	1.5	1.0	1.5	1.0	2.25	?
1.0	1.5	1.5	2.0^-	1.5	3.0^-	?
1.5	2.0^-	1.0	1.0	1.5	2.0^+	?
1.5	2.0^-	1.0	1.5	1.5	3.0^-	?
1.5	2.0^-	1.5	2.0^-	2.25	4.0^-	1

Table 4. The normalization decision based on 2 input digits of each operand.

4.1 Refinements

A refinement to the C/S design can be made using the fact that the decision of whether or not to normalize can be made for some fraction of the cases using only the first two digits of the operands $X_0^s, X_1^{c,s}$ and $Y_0^s, Y_1^{c,s}$, thus mitigating the need for ovf_P0. This only makes sense for the C/S version because for the conventional design the full carry propagation must be made on the upper half of P anyway. This allows the C/S version to work at two different speeds, the finish state being signified by a single bit. In order to determine the normalization state we can determine the *min* and *max* of X and Y, as in table 3, and then determine the range of P based on those ranges as shown in table 4. This leads to knowing the overflow condition for 4 out of 9 cases. For these operands the delay of the multiplier is 58ns, plus some small logic delay for analyzing the two leading digits.

4.2 Conclusion

We have shown that using standard layout tools and Spice analysis one can design a floating-point multiplier which accepts inputs in carry/save form. The delay of this multiplier is comparable with the delay of a multiplier with conventional operands of the same precision, paying only about an extra 15% in area. These savings are in addition to the delay which was not generated in the unit which produced the operands. Also, since the primary difference between the redundant form and the conventional is that of the full length CPA which must be performed, the savings should grow when moving to double and quad precision operands.

References

[1] A. Glew, "Processor with architecture for improved pipelining of arithmetic instructions by forwarding redundant intermediate data forms," *U. S. Patent no. 5619664,* April 1997.

[2] T. Nishiyama et al., "High Speed Multiplier Utilizing Signed-Digit and Carry-Save Operands," *U.S. Patent Number 4,868,777,* Sept. 1989.

[3] M. Matula and A. Nielson *Pipelined packet-forwarding floating point. I. Foundations and a rounder, Proceedings.* 13^{th} *IEEE Symposium on Computer Arithmetic*, Asilomar, CA, 6-9 July 1997. p.140-7.

[4] M.I. Ferguson, M. D. Ercegovac, "A Multiplier with Redundant Operands," *Proceedings of the* 33^{rd} *Annual Conference on Signals, Systems, and Computers*, Asilomar, Pacific Grove, October 24-27,1999 p. 1322-1325.

[5] *IEEE Standard for Binary Floating-Point Arithmetic,* Inst. Electr. & Electron. Eng., New York, NY, USA, 12 Aug. 1985. 18 pp.

[6] V. G. Oklobdzija, et al., "A Method for Speed Optimized Partial Product Reduction and Generation of Fast Parallel Multipliers Using an Algorithmic Approach," *IEEE Transactions on Computers,* Vol. 45, No. 3, March 1996.

[7] M. Santoro, G. Bewick, M. Horowitz, "Rounding Algorithms for IEEE Multipliers". *Proceedings of the* 9^{th} *Symposium on Computer Arithmetic*, Santa Monica, 6-8 Sept. 1989 p.176-83.

[8] M. D. Ercegovac and T. Lang, "Digital Arithmetic", Sept. 2000, manuscript in progress.

[9] Internet address www.mosis.com /Technical/Processes/proc-hp-amos14tb.html

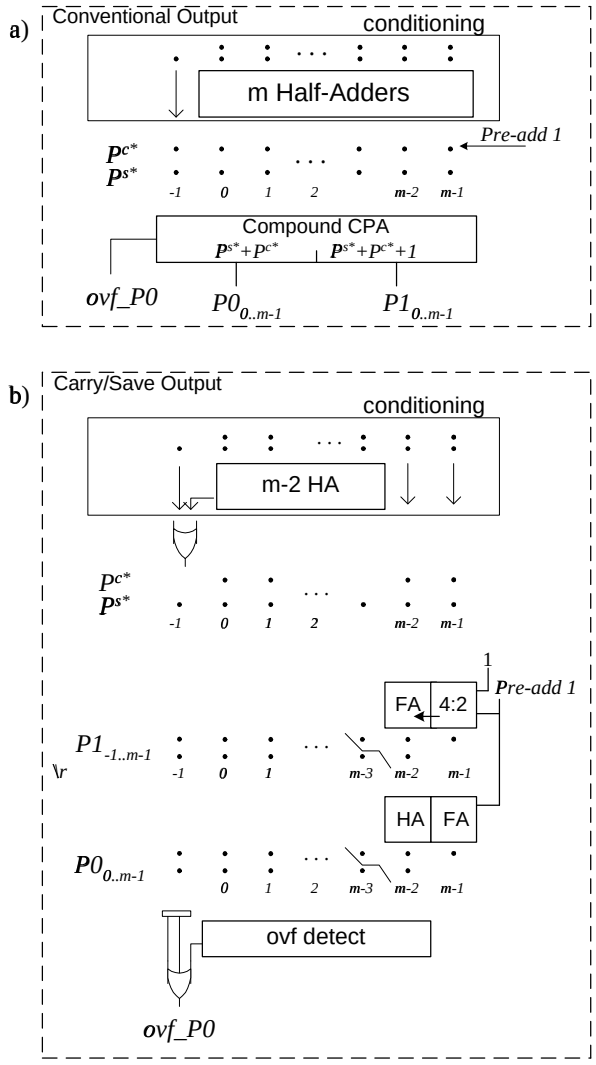

Figure 3. The difference between rounding to a conventional and a carry/save operand. The outputs to the left of the jagged line come directly from Pc* and Ps*