A Dual Mode IEEE Multiplier

Guy Even*, Silvia M. Mueller†, and Peter-Michael Seidel‡
Dept. 14: Computer Science
University of Saarland
66123 Saarbruecken, Germany
E-mail: guy,smueller,pmseidel@cs.uni-sb.de

Abstract

We present an IEEE floating-point multiplier capable of performing either a double-precision multiplication or a single-precision multiplication. In single-precision the latency is two clock cycles and in double-precision the latency is three clock cycles, where each pipeline stage contains roughly fifteen logic levels. A single-precision multiplication can be followed immediately by another multiplication of either single or double-precision. A double-precision multiplication requires one stall cycle, namely, two cycles after issuing a double-precision multiplication, a new multiplication of either precision can be issued. Therefore, the throughput in single-precision is one multiplication per clock cycle, and the throughput in double-precision is one multiplication per two clock cycles. Hardware cost is reduced by using only a half-sized multiplication array and by sharing the rounding circuitry for both precisions.

1. Introduction

Fast low precision floating-point operations have been acknowledged recently as very useful for real-time 3D graphic applications [2, 5, 6, 9]. The new 3D graphic applications reverse the previous trend of focusing on higher and higher precisions and set a new performance goal. The new design goal is to design cheap IEEE floating-point units (FPUs) capable of both double-precision and single-precision operations with a higher throughput for single-precision operations.

IEEE FPUs support both single and double-precision operations. However, the throughput of low-precision floating-point operations is often smaller than high-precision operations. This is caused by the necessity to translate single-precision operands to double-precision format, and then translate the double-precision result back to single-precision format. In the x86 architecture (e.g. Pentium) and in the Motorola 68000 family, computations are performed in extended double-precision and translation to low precision requires storing the result to memory.

*Supported by the North Atlantic Treaty Organization under a grant awarded in 1996.
†Supported by the DFG.
‡Supported by Graduiertenkolleg "Effizienz und Komplexität von Algorithmen und Rechenanlagen", Universität des Saarlandes.

0-7803-4275-5/97 $8.00

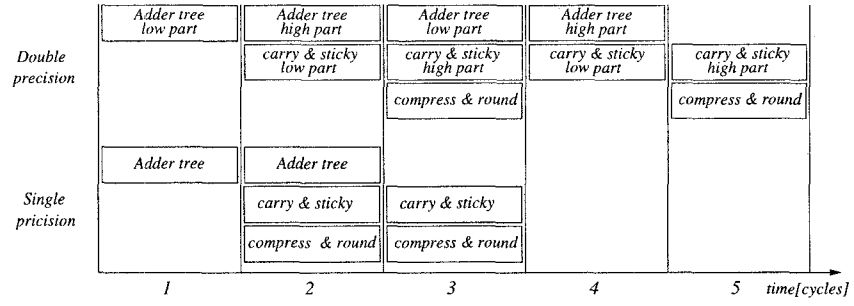

Figure 1: Timing diagram of dual mode multiplier. The shaded boxes correspond to the next multiplication.

Our goal is to present designs of cheap and versatile IEEE FPUs that deliver good performance for both precisions.

Our multiplier supports all of the IEEE rounding modes. In single-precision the latency is two clock cycles and in double-precision the latency is three clock cycles, where each pipeline stage contains roughly fifteen logic levels. The multiplier uses a half-sized (i.e. 27×53) multiplication array, and thus, the cost of the multiplier is reduced (see for example [1, 3]). In double-precision multiplication, the multiplication array is used during the first two cycles. Therefore, new multiplications can be issued only after two clock cycles. In single-precision multiplication, a new multiplication can be issued in the next clock cycle. The hardware overhead and delay caused by supporting both precisions compared to a similar design that supports only double-precision multiplications is surprisingly small.

We present a new rounding algorithm which simplifies supporting different precisions and enables using the same rounding circuitry for both precisions. In addition, our rounding algorithm advances some of the rounding computation to the addition tree without adding delay to the addition tree. Our rounding algorithm differs from previous suggestions [4, 7, 10, 11] by injecting a value that depends only on the rounding mode.

2. Description

In this section we describe the multiplier. A block diagram of the significand's path in the multiplier is depicted in Figure 2. We describe the operation of the multiplier in each mode separately.

2.1. Double-Precision

We assume that the two significands, A and B, are normalized (namely, in the range $[1, 2)$), and held in registers.

Figure 1 depicts the timing diagram describing the operation of the multiplier. In the first clock cycle, A is multiplied by the 26 least-significant bits of B . The empty (i.e., first) row in the multiplication array is used for the injection. The addition of the injection reduces all

represented by a sum and carry vector, each consisting of 80 bits (corresponding to positions $[-26:-105]$). The sum and carry, denoted by S and C, are latched by a register.

In the second clock cycle, A is multiplied by the 27 most-significant bits of B. The 53 most-significant bits of S and C (bit positions $[-26:-78]$) are fed back to the addition tree for accumulation. The addition tree outputs the upper half of the product (bit positions $[1:-78]$) which is latched by Registers S and C. During the second clock cycle, the 27 least-significant bits of Registers S and C (bit positions $[-79:-105]$) are input to the carry and sticky computation, resulting with a carry-bit c_78 and a sticky-bit $sticky\_low$ that are latched by a register. (Note, that the register that latches c_78 and $sticky\_low$ is reset except for the end of the second clock cycle of a double-precision multiplication).

In the third clock cycle, the 26 least-significant bits of S and C (bit positions $[-53:-78]$) are input to the carry and sticky computation box. This box combines the carry-bit c_78 and the sticky-bit $sticky\_low$ computed during the previous cycle (for positions $[-79:-105]$) with the carry for positions $[-53:-78]$ and the sticky-bit for positions $[-54:-78]$. The results of this computation are the carry signal c_S that is to be input to position $[-53]$, the carry signal c_R that is input to position $[-52]$, and the sticky-bit corresponding to positions $[-54:-105]$.

During the third cycle, the 54 most-significant bits of S and C (bit positions $[1:-52]$) are added by a 3-way adder that computes the sum, the incremented sum, and the sum plus 2. (To be precise, the values computed are: the sum, the sum plus 2^{-52}, and the sum plus 2^{-51}). After the 3-way addition, three selections take place to determine the rounded product: (a) the c_R signal controls the selection of the pair $sum, sum+1$ or the pair $sum+1, sum+2$ (we denote the selected pair by $x, x+1$). (b) The MSB (bit position [1]) of the sum together with the rounding-mode, carry c_S, and the round-bits $S[-53], C[-53]$ determine which value is selected: x or $x+1$. (c) The MSB determines whether the product (ignoring the carry from position [53] and the possible rounding increment) is in the range $[1,2)$ or in the range $[2,4)$. According to the MSB, the product is shifted to the right so that it will be in the range $[1,2]$. The order of the selections is chosen to reduce the delay. Since the c_R signal is valid before the output of the 3-way adder, we put the *carry-mux* first. One could change the order of the right-shifting and the *inc-mux* to reduce the delay even further, but that would require having two right-shifters rather than one.

Finally, two corrections take place: (a) The LSB of the rounded product is fixed (in the case of round-to-nearest-even, as explained later); and (b) The MSB is fixed since significand overflow might occur. Therefore, the product has to be brought to the range $[1,2)$. This is done by ORing the two most-significant bits of the product (that are both to the left of the radix point) to produce the final most-significant bit of the product.

Note, that the first stage of the pipeline is engaged in double-precision multiplication during the first and second clock cycles. Therefore, a new multiplication can be issued only after two clock cycles, yielding a throughput of one double-precision multiplication per two clock cycles as depicted in Figure 1.

We assume that Registers A and B hold 24-bit normalized significands. Note, that one could consider packing two single-precision significands in each register, and perform the corresponding multiplications one after the other. We omit this packing possibility to simplify the description.

In single-precision multiplication, multiplexer $Amux$ shifts A by 29 positions to the right. This "right-justification" causes the least-significant bit of the single-precision product (i.e. bit $[-23]$) to be placed in the same position as the least-significant bit of the double-precision product (i.e. bit $[-52]$). The advantage of this shifting is that the bit positions required for rounding in both precisions become identical and hence the same circuitry can be used. Multiplexer $Bmux$ selects B but keeps it "left-justified" (i.e. the radix point is aligned with the radix point in double-precision mode).

In the first clock cycle the multiplication array computes $A \cdot B + injection$ (the injection is explained later). The computed products are latched in the S and C registers, and no feedback is used. Since the multipliers B has 24 bits, one of the empty rows in the multiplication array can be used for the injection.

During the second clock cycle, the product, stored in S and C, is compressed and rounded. The lower part is input to the carry and sticky computation box, and the upper part is added using the 3-way adder. The only modification compared to the double-precision rounding is the position of the MSB. Since the single-precision product is aligned with the double-precision product with respect to the L-bit position, the MSB of the single-precision number appears in position $[-28]$. The selection of the correct MSB is done by the *msb-mux*. At the end of the computation, the product needs to be shifted 29 positions to the left so that that it is "left-justified".

Note, that the first stage of the pipeline is engaged in single-precision multiplication only during the first clock cycle. Therefore, a multiplication can be issued immediately after a single-precision multiplication, yielding a throughput of one single-precision multiplication per clock cycle as depicted in Figure 1.

3. Details

In this section we focus on two methods that our used in our multiplier: the rounding algorithm and the sticky-bit computation.

3.1. Rounding

Our rounding algorithm is designed to support multiple precisions and enable using the same rounding circuitry while still obtaining a fast rounding algorithm.

We briefly describe previous rounding algorithms that we are aware of:

1. Santoro *et al.* [10] and Quach *et al.* [10, 7] reduced the four IEEE rounding-modes to three modes called *round-to-zero* (RZ), *round-to-infinity* (RI), and *round-to-nearest-even* (RNE). These modes correspond to a "floor" rounding, a "ceiling" rounding, and the

Furthermore, they described how to reduce RNE to a rounding mode called round-to-nearest-up (RNU). The difference between RNE and RNU is that in a case of a tie, the number is rounded up in RNU, whereas in RNE it is rounded to the even significand. Given a RNU rounding unit, one can easily obtain the correct RNE result by pulling down the LSB in case of a tie that should have been rounded-down rather than rounded-up.
The rounding algorithms of Santoro *et al.* [10] and Quach *et al.* [7] are based on the above reduction and on truncating the sum and carry vectors in the L position and adding an *ulp* (called the prediction) based on the rounding-mode and the values of the sum and carry vectors in the R position. These rounding algorithms also require a 3-way adder.

2. Yu and Zyner [11, 12] presented a fast rounding algorithm in which the product given as a carry-save number is divided into 3 parts: the lower part produces a carry-bit and a sticky-bit, the upper part is compressed to obtain the sum and the incremented sum, and the middle part is used to compute the least-significant bit, the guard-bit, and the round-bit. Two parallel paths are used for the rounding decision: one for the case that the product is in the range $[1, 2)$ and one for the case that the product is in the range $[2, 4)$. The final selection is based on the MSB of the product and the carry generated by the rounding.
3. Gamez *et al.* [4] presented a rounding algorithm that avoids having to shift the operands and the product while partially sharing the same rounding circuitry for all precisions. Their algorithm is input a non redundant representation of the number to be rounded. The algorithm is based on having two paths: In one path, the number is padded with zeros starting from the rounding position. In the second path, the number is padded by ones starting at the round position, and then it is incremented (increment occurs at a fixed position). The rounding decision based on the true LSB, round-bit, and sticky-bit selects one of these numbers as the rounded result. Note that this rounding algorithm requires normalizing the product before rounding so that it is in the range $[1, 2)$.
4. Saishi *et al.* [8] presented a rounding algorithm for fixed-point two's complement multiplication that supports two rounding modes (round-to-zero and RNU). Their rounding algorithm is based on injecting a value determined by the sign of the product and the rounding mode so that rounding is reduced to truncation. Our rounding algorithm is an extension of this idea to accommodate for different precisions in the context of floating-point multiplication. In addition, our injection is added directly to the addition tree depending only on the rounding mode, whereas the injection presented by Saishi *et al.* depends also on the sign of the product.

Our rounding algorithm adds an injection to the addition tree based on the rounding mode in order to reduce the three rounding modes to truncation. Formally,

$$round(a \cdot b) = truncate(a \cdot b + injection).$$

In RZ mode the injection value is zero, in RNU mode the injection value is 2^{-p}, and in RI the injection value is $2^{-(p-1)} - 2^{-2(p-1)}$, where $p = 24$ in single-precision, and $p = 53$ in double-precision.

The injection reduces all three rounding modes (RZ, RNU, and RI) to truncation provided that the product is in the range $[1, 2)$. When the product is in the range $[2, 4)$, the injection should have been $2 \cdot 2^{-p}$ in RNU mode, and $2^{-(p-2)} - 2^{-2(p-1)}$ in RI mode. This requires adding an

the L position. However, adding 2^{-p} translates to an increment only if the bit in the R position is one. This is why the correction box needs $S[R]$, $C[R]$, and c_S (the XOR of these bits equals the bit in the R position of the product).

A careful reader will notice that we determine the case that the product is in the range $[1, 2)$ or $[2, 4)$ based on the MSB of the product after the initial injection is added in. This means that it is possible for the exact product to be less than 2 while the initial injection increases it to 2. Does this imply that correcting the injection yields a wrong rounded result? The answer is no, and the reason is that the LSB equals zero in this case, and the increment does not generate a carry. Shifting the product to the right discards the LSB and the final rounded result is correct.

3.2. Sticky-bit and Carry-bit Computation

The sticky-bit and carry-bit computation is performed on a carry-save number having 27 digits. One could compress this number and OR the bits to obtain the sticky-bit. Alternatively, one can compute the sticky-bit from the carry-save number as suggested by Yu *et al.* [11]. Their circuit requires only a constant number of gates before the OR tree. The carry-bit can be computed using a carry lookahead computation and can share some of the circuitry of the sticky-bit computation.

4. Conclusions

We have presented an IEEE floating-point multiplier capable of performing either single precision or double-precision multiplications. In single precision, the latency is two clock cycles and the throughput is one multiplication per clock cycle. In double-precision, the latency is three clock cycles and the throughput is one multiplication per two clock cycles. The multiplier saves hardware by using a half size multiplication array and by using the same rounding circuitry for both precisions.

5. Acknowledgments

We would like to thank Holger Leister, Stuart Oberman, and Wolfgang Paul for helpful discussions. Access to patent descriptions provided by the IBM Patent Server helped us find related patents.

6. References

[1] W.S. Briggs and D.W. Matula. A 17 x 69 bit multiply and add unit with redundant binary feedback and single cycle latency. In *Proceedings 11th Symposium on Computer Arithmetic*, number 11, pages 163–170, 1993.

[2] Chromatic's Mpact 2 boosts 3D. *Microprocessor Report*, 10(15), Nov. 1996.

[3] J. Fandrianto and B.Y. Woo. VLSI floating-point processors. In *Proceedings 7th Symposium on Computer Arithmetic*, number 7, pages 93–100, 1985.

[4] C. Gamez and Pang R. Apparatus and method for rounding operands. U.S. patent 5258943, 1993.

[6] Hitachi SH-4 gets graphically superscalar. *Microprocessor Report*, 10(14), Oct. 1996.

[7] N. Quach, N. Takagi, and M. Flynn. On fast IEEE rounding. Technical Report CSL-TR-91-459, Stanford University, January 1991.

[8] M. Saishi and T. Minemaru. Multiplication circuit having rounding function. U.S. patent 5500812, 1996.

[9] Samsung launches media processor. *Microprocessor Report*, 10(11), Aug. 1996.

[10] M.R. Santoro, G. Bewick, and M.A. Horowitz. Rounding algorithms for IEEE multipliers. In *Proceedings 9th Symposium on Computer Arithmetic*, pages 176–183, 1989.

[11] R.K. Yu and G.B. Zyner. 167 MHz radix-4 floating point multiplier. In *Proceedings 12th Symposium on Computer Arithmetic*, pages 149–154, 1995.

[12] G. Zyner. Circuitry for rounding in a floating point multiplier. U.S. patent 5150319, 1992.

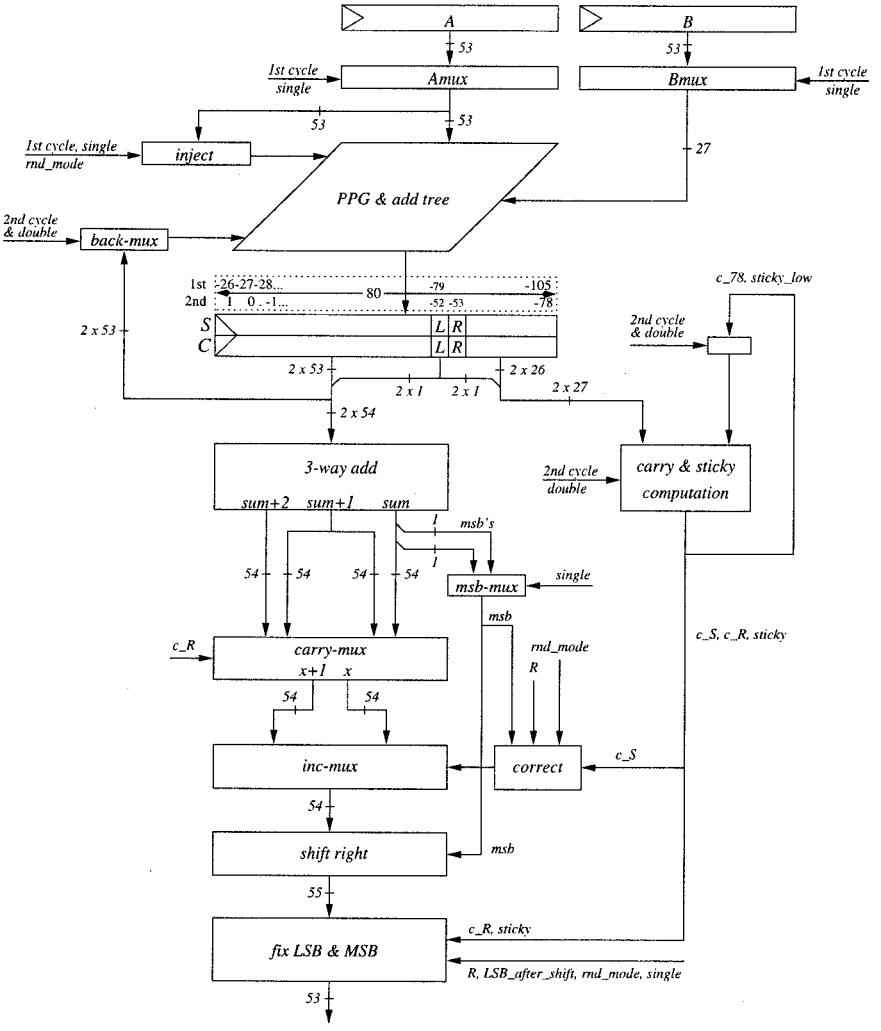

Figure 2: Block diagram of the dual-mode multiplier. Shaded boxes are the only components that change their behavior depending on the mode (i.e. single or double-precision).