Reduced Latency IEEE Floating-Point Standard Adder Architectures

A.Beaumont-Smith, N.Burgess, S. Lefrere and C.C. Lim
CHiPTec, Department of Electrical and Electronic Engineering, The University of Adelaide
Adelaide, 5005, Australia
abeaumon@eleceng.adelaide.edu.au

Abstract

The design and implementation of a double precision floating-point IEEE-754 standard adder is described which uses "flagged prefix addition" to merge rounding with the significand addition. The floating-point adder is implemented in $0.5\mu m$ CMOS, measures $1.8mm^2$, has a 3-cycle latency and implements all rounding modes. A modified version of this floating-point adder can perform accumulation in 2-cycles with a small amount of extra hardware for use in a parallel processor node. This is achieved by feeding back the previous un-normalised but correctly rounded result together with the normalisation distance. A 2-cycle latency floating-point adder architecture with potentially the same cycle time that also employs flagged prefix addition is described. It also incorporates a fast prediction scheme for the true subtraction of significands with an exponent difference of 1, with one less adder.

Key Words: floating-point, adder, arithmetic, VLSI.

1. Introduction

Floating-point (FP) addition is the most frequent FP operation and FP adders are therefore critically important components in modern microprocessors [4, 6, 7, 12, 5] and digital signal processors [23]. FP adders must be fast to match the increasing clock rates demanded by deep submicron technologies with a small number of pipelining stages to minimise latency and improve branch resolution time. FP adders must also be small, particularly for use in parallel processing systems [1, 11] with multiple FP units (FPUs).

The design of FP adders is considered more difficult than most other arithmetic units due to the relatively large number of sequentially dependent operations required for a single addition (we use addition to mean an add or subtract operation) and the extra circuits to deal with special cases such as infinity arithmetic, zeros and NaNs, as demanded by the IEEE-754 standard. As a result, there is scope for investigating smaller and faster FP adders by re-organising the algorithm and using new circuit techniques.

This paper discusses the design and implementation of an IEEE-754 compliant double precision 3-cycle floating-point adder which uses minimal hardware for use in a parallel processor with multiple FPUs [1]. This is achieved by merging the rounding stage with the significand addition by using a flagged prefix adder [2] which provides an "instant" plus "1" or plus "2" of the significand result for a small increase (one complex gate plus a half adder delay) in critical path length with much less hardware than two adders or one compound adder. We then presents a modified version of this FP adder that supports 2-cycle latency accumulation by overlapping the alignment and normalisation shifts into the first pipeline stage. A further algorithm is proposed to perform FP addition in 2 cycles which can match the latency of FP multipliers. Providing matched latencies of multipliers, adders and accumulators in multiple FPUs in a microprocessor simplifies the pipeline scheduling and improves branch resolution times. This new FP adder also incorporates a fast prediction scheme for the true subtraction of significands with an exponent difference of "1", with one less adder.

Traditional methods for performing FP addition can be found in Omondi [18] and Goldberg [4], who describe algorithms based on the sequence of significand operations: swap, shift, add, normalise and round. They also discuss how to construct faster FP adders. Implementations of FP adders are reported in [6, 7, 12, 5, 9, 13, 10]. Algorithms and circuits which have been used to improve their design are described in [17, 8, 3, 20, 16, 21, 15, 22, 19].

Some of these improvements are as follows:

- the serial computations such as additions can be reduced by placing extra adders (or parts thereof) in parallel to compute speculative results for exponent subtraction ($E_a - E_b$ and $E_b - E_a$) and rounding ($M_a + M_b$ and $M_a + M_b + 1$) then selecting the correct result (e.g. T9000 [9]).

- by using a binary weighted alignment shifter, the cal-

culation of the exponent differences ($E_a - E_b$ and $E_b - E_a$) can be done in parallel with the significand alignment shift.

- the position of where the round bit is to be added when performing a true addition depends on whether the significand result overflows, so speculative computation of the two cases may be done [10].
- calculation of the normalisation distance can be done by a leading zero anticipator to provide the normalisation shift to within one bit, in parallel with the significand addition [8].
- a fast integer adder is crucial to the design of FP adders for calculating the result significand and sets the minimum cycle time [14].

Further improvements in speed can be made by splitting the algorithm into two parallel data paths based on the exponent difference [5, 10, 3, 16, 15], namely near ($|E_a - E_b| \leq 1$) and far ($|E_a - E_b| > 1$) computations, by noting that the alignment and normalisation phases are disjoint operations for large shifts. However there is a significant increase in hardware cost since the significand addition hardware cannot be shared as with the traditional three stage pipeline.

Other FP adder designs have moved the rounding stage before the normalisation [21, 19, 12]. Quach and Flynn describe an FP adder [21] which uses a compound significand adder with two outputs plus a row of half adders and effectively has a duplicated carry chain. Kowaleski *et al.* [12] describe an adder as part of a 263,000 transistor FP unit which contains separate add and multiply pipelines with a latency of 4-cycles at $433MHz$ in a $0.35\mu m$ process. The significands are simultaneously added and rounded by employing a half adder which makes available a LSB for the case where a "1" is added for taking the two's complement of one input. Decode logic on two LSBs of both operands calculates if there will be a carry out of this section as a result of rounding, either by adding one or two. A circuit is used to determine if the MSB is "1" as a result of adding the significands in which case the round bit is added to the second LSB to compensate for the subsequent normalisation by 1-bit to the right. The significands are added by using a combination of carry look-ahead and carry select logic. Precomputed signals predict how far the rounding carry will propagate into the adder. The MSB computed before rounding is used to select the bitwise sums for the result. Nielsen *et al.* describe a redundant-add FP adder [15] with addition and rounding separated by normalisation. By using redundant arithmetic, the increment for the rounding is not costly. A variable latency architecture [16] has been proposed which results in a one, two, or three clock cycle data dependent addition. However, this implies that the host system can take advantage of up to three simultaneously emerging results, which represents a considerable scheduling difficulty.

FP adders that have an accumulation mode or can bypass incomplete results are of interest in vector and deeply pipelined machines. Hagihara *et al.* [6] report a $0.35\mu m$, $125MHz$ FPU with a 3-cycle FP adder which adds three operands simultaneously, so two new operands can be accumulated per cycle, saving 25% of the accumulation execution time in a vector pipelined processor for use in supercomputers. Heikes and Colon-Bonet [7] report a FMAC architecture incorporating a floating point adder path with 4-cycle latency at $250MHz$ in a $0.5\mu m$ CMOS process employing dual-rail domino logic. The rounding is delayed and can produce a bypassable unrounded result in 3-cycles.

Throughout the rest of this paper, all operands are assumed to be 64-bit IEEE-754 format floating-point numbers with a 52-bit fractional significand ($M = s_{51}s_{50}\ldots s_0$), an 11-bit biased exponent ($E = e_{10}e_9\ldots e_0$) and a 1-bit sign (S). For all examples in this paper, the FP numbers A and B have significands M_a and M_b, exponents E_a and E_b and signs S_a and S_b respectively. The prefix true is used to describe the underlying addition and subtraction process of the significand adder, i.e. (-1.0) add (1.5) is a true subtraction.

2. A 3-cycle Floating Point Adder using a Flagged Prefix Integer Adder

A 3-cycle IEEE-754 double precision FP adder is shown in Figure 1. The control signals have been omitted for clarity and the pipelining stages are indicated by horizontal bars. It is similar in design to the traditional three stage pipeline adder in that the first stage is an exponent subtraction, significand swap, unpacking and alignment shift, the second stage performs the significand addition and the third stage performs the post normalisation shift. Rounding of the significand, which is normally done after the normalisation shift, is merged with the significand addition in the second pipeline stage using a flagged prefix adder.

In stage 1, a pass transistor type shifter was used which consumes less area than a binary weighted multiplexer. Two adders subtract 54 from the twin exponent differences to provide a valid signal for the barrel shifter if the shift is within range. The critical path for the first stage is through the exponent subtractors, alignment decoder and pass transistor select circuits in the shifter.

2.1. Significand addition and rounding

In the case of true subtraction, the smaller number can be negated using two's complement form and added using a fast carry propagate adder to produce the magnitude of the result. Finding the smallest significand can be very costly

if the exponents are equal, so one technique to speed up the operation employs two adders to calculate $M_a - M_b$ and $M_b - M_a$ in parallel and select the positive result. Another similar method is to calculate $M_a + \overline{M_b} + 1$ ($M_a - M_b$) and $M_a + \overline{M_b}$ which can be negated to give $M_b - M_a$. The same circuit can be used to perform rounding for true addition. Both methods generally use two adders in parallel.

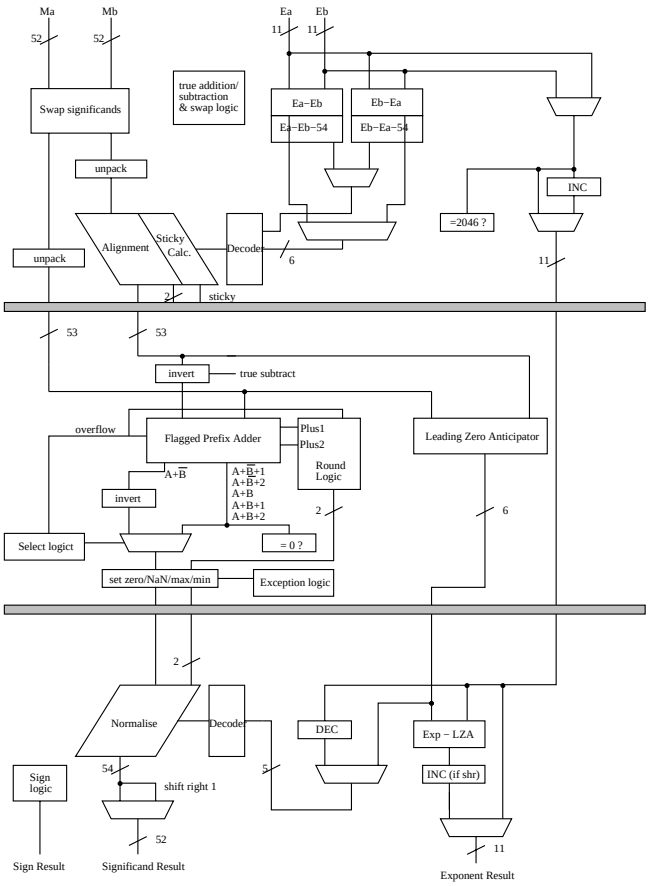

Figure 1. A 3-cycle floating point adder.

There are four implemented rounding modes, round towards zero, nearest and plus/minus infinity. Rounding of the result to 53-bits must be carried out as if the result was exact (to infinite precision) and then the rounding rules applied to determine if the significand of the result should be truncated or incremented. The 53-bit result, pre-shift guard, round and sticky bits are be passed to the rounder circuit; the final guard, round and sticky bits are determined from the postshift amount. The guard bit is the bit below the LSB of the 53-bit result and the round bit is to the right of the guard bit. The pre-shift sticky bit is obtained by OR'ing the pre-shifted significand bits below the pre-shift round bit.

The true subtraction operation requires that a "1" be added into the LSB of the smallest number for the two's complement operation of M_b since the positive result is needed. To produce an exact representation of the result, the "1" must be added into a bit infinitely below the number so that the "1" propagates through the infinite string of trailing zeroes up to the pre-shifted LSB. The only case where the "1" can propagate to the round bit is when all of the pre-shifted bits of M_b below the round bit are zero (which are then inverted). So "1" is added into the round bit if the pre-shift sticky bit is zero for true-subtraction. True addition can either:

- produce a significand result greater than or equal to one and less than two and the pre-shift guard, pre-shift round and pre-shift sticky bits become the guard, round and sticky bits for the rounding circuit.
- cause an overflow to occur if the result is greater than or equal to two and less than four, in which case the LSB becomes the guard bit, the pre-shift guard becomes the round bit, the pre-shift round and pre-shift sticky bits are OR'd to form the sticky bit for the rounding circuit.

True subtraction can produce (assuming $A \geq B$):

- $E_a = E_b$: no rounding is possible, result is exact
- $E_a - E_b = 1$: no rounding is done if the normalisation shift is ≥ 1 since the round and sticky bits must be zero. If the normalisation shift is zero then rounding is performed (only the guard bit, which has become the round bit, is used).
- $E_a - E_b \geq 2$: the normalisation shift will be at most one, and rounding is needed.

The rounding circuit drives the control signals for the flagged prefix adder, described below, to instantly change the significand result by plus "1" (for a true addition round increment or a true subtraction with no increment) or plus "2" (if the true addition result had overflowed or true subtraction and a round increment occurred).

It is possible for the rounding incrementer to produce an overflow of the significand and possibly the exponent. This is detected and corrected with a shift right in the normalisation phase and uses the same multiplexer to do this as the correction for the LZA.

2.2. A flagged prefix adder for merged significand addition/rounding

In this section only, we refer to A and B as significands. To merge the rounding process with the significand addition, the following results need to be computed: $A + B$, $A + B + 1$, $A + B + 2$, $A - B$, $A - B + 1$, $B - A$. To compute this (in two's complement form), the actual calculations are; $A + B$, $A + B + 1$, $A + B + 2$, $A + \overline{B} + 1$,

$A+\overline{B}+2$, $\overline{A+\overline{B}}$. Burgess proposed a flagged-prefix adder [2] which computes $A+B$, $A+B+1$, $-(A+B+2)$, $-(A+B+1)$, $A-B-1$, $A-B$, $B-A-1$ and $B-A$ for a minor increase in delay from a single addition (1 complex gate plus a half adder). A flagged prefix adder shown in Figure 2(a) is a modified parallel prefix-type carry lookahead adder which calculates the longest string of one's (up to and including the first "0") from the second LSB upwards and sets them as "flag bits" to indicate those bits which can be inverted for incrementing a two's complement number. The overhead is approximately 1 extra complex gate plus a half adder per bit over a prefix type adder. By way of comparison, a compound adder replaces each prefix cell by a pair of multiplexers and has an extra row of output multiplexers. Therefore flagged prefix adders use approximately 70% of the logic of a compound adder.

A subset of the available functions in a flagged prefix adder was used which simplified the flagged inversion cells shown in Figure 2(b). The 2 least significant flagged inversion cells were replaced with flag logic which provides a plus "2" function in addition to the standard plus "1" function.

The flag bits start from the second LSB and must detect the following strings of bit propagate (p), bit generate (g) and bit kill (k) signals to facilitate a "plus 2" operation:

$$....ppppppp, ...pppppk,$$

$$....ppppkg, ...ppppkgg, etc.$$

To simplify the detection of these bit strings a row of half adders is placed before the prefix adder to convert the "kgg..." conditions to strings of bit propagate signals. The flag bits are then defined as $f_i = f_{i-1}.p_{i-1} = P^0_{i-1}$, the $i-1^{th}$ group propagate signal, available from the prefix tree.

The control signals, *plus1* and *plus2* for the flagged prefix adder are generated from the rounding mode, sticky bit, result LSB, round bits and the true addition/subtraction signals. These control signals together with the bit propagate and generate signals, p0,g0,p1,g1 and the overflow signal are used to drive the *inc* signal for the flagged inversion cells.

In a true addition, $A+B$ needs to be calculated. If the result of the rounding circuit is an increment then *plus1* is set. If $A+B$ overflows, then adding a "1" becomes adding "2" to the unshifted significand, the LSB becomes the round bit and *plus2* is set. If the rounded result ($A+B+2$ or $A+B+1$) produces an overflow then this is shifted right one place and truncated.

For the case of true subtraction the *plus1* signal is automatically set (for the negation of the second operand). If the result also needs to be incremented due to rounding, then the *plus2* signal is set. We could have added the compulsory "1" as a carry input to the LSB of the adder but we may need $\overline{A+\overline{B}}$ (actually $B-A+1$) if $A+\overline{B}+1$ overflows and the requirement for $A+B+2$ for true addition means all operations can be performed with a single flagged prefix adder.

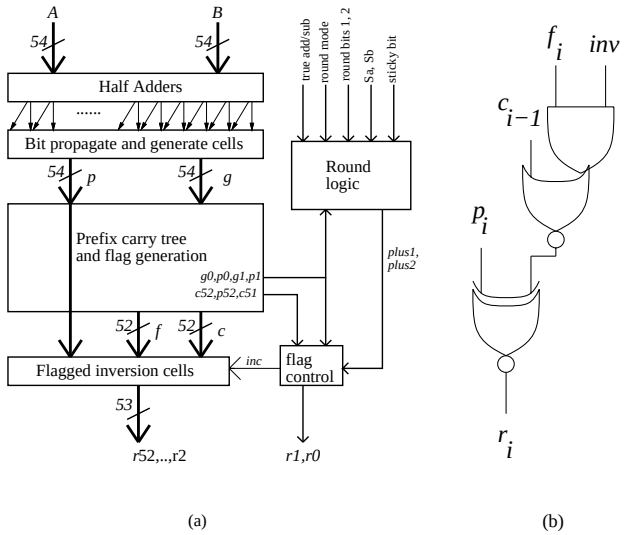

Figure 2. (a) A flagged prefix adder for merged significand addition and rounding and (b) a flagged inversion cell.

2.3. Normalisation

The normalisation stage left shifts the significand to obtain a leading "1" in the MSB, and adjusts the exponent by the normalisation distance. For true subtraction, the position of the leading "1" of the significand result is predicted from the input significands to within 1-bit using an LZA [17, 8] which is calculated concurrently with the significand addition. The normalisation distance is passed to a normalisation barrel shifter and subtracted from the exponent. The LZA circuit is arranged so the error is always larger than the required shift and correction circuits right shift the significand and add "1" to the exponent if the normalisation distance is too large. For true addition, the only possible post-shifting that can be performed is a single shift right if the significand addition and rounding operation produced a supernormal result. The multiplexer to perform this is shared with the LZA correction function.

The fractional significand can now be split off from the implied "1" and packed with the exponent and sign bit to produce the 64-bit result.

2.4. Implementation

A micrograph of the fabricated $0.5\mu m$ FP adder chip described above is shown in Figure 3 and the main functional blocks are highlighted. Most of the FP adder was constructed using static CMOS logic and it contains $33,000$ transistors. It was full custom designed using VHDL for high level design and verification with a suite of C test routines. The adder was functionally verified against the Berkeley IEEE-754 test vector set and sets of random vectors generated by a golden device (SUN UltraSPARC 60 [5]). This FP adder will be used in a parallel processor node [1] for fast matrix computations.

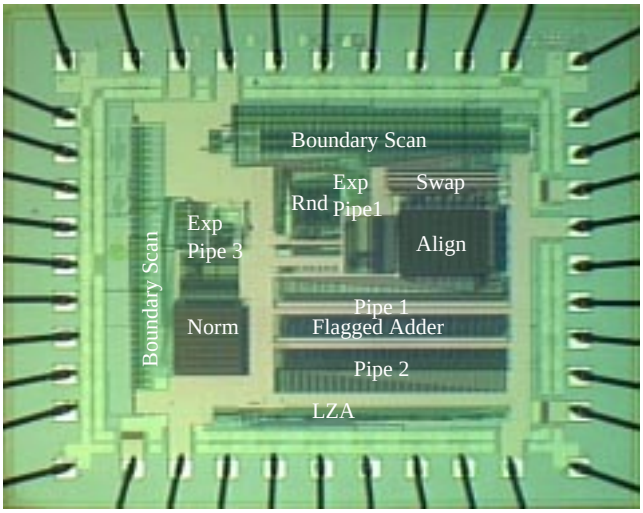

Figure 3. Micrograph of the $0.5\mu m$ CMOS FP adder chip.

3. A 2-cycle Accumulator Architecture

A new FP adder which supports 2-cycle accumulation whilst still being able to perform 3-cycle pipelined additions is shown in Figure 4. A correctly rounded, possibly un-normalised result is produced at the output of the second stage of the FP adder described in the previous section, together with the normalisation distance (which may be in error by 1-bit). This number may be either normalised in the third stage of an adder or accumulated with the next but one input using only a small amount of extra hardware (four multiplexers). This is useful in processors where many short accumulations are performed as it can reduce the pipeline start-up and finishing cost.

In the accumulation mode, the normalisation of the accumulator and the alignment shift operations can be overlapped so the critical path does not pass through both

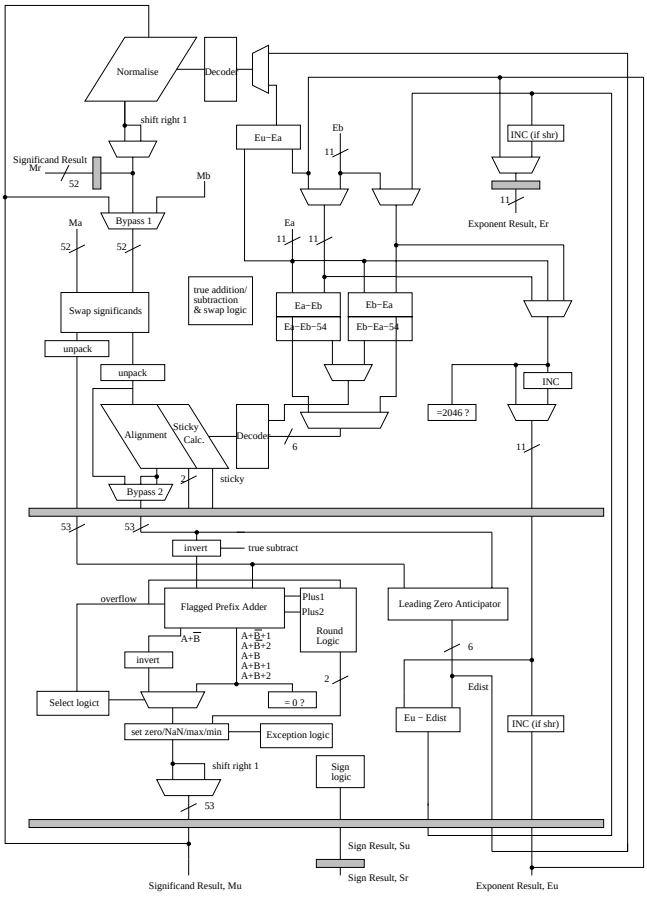

Figure 4. A 3-cycle FP adder which can be used as a 2-cycle accumulator.

shifters. The possibly un-normalised accumulator significand, M_u is fed back to the start of the adder along with its exponent, E_u, sign, S_u and a normalisation distance, E_{dist}. There are three cases to be considered to align the accumulator and addend significands:

- $E_a \geq E_u$: the normalisation distance, E_{dist} can be ignored and M_u can be shifted further to the right by the exponent difference ($E_a - E_u$). The M_u significand bypasses the normalise shifter through the bypass-1 multiplexer.

- $E_a < E_u < E_a + E_{dist}$: the M_u significand is shifted left by the exponent difference ($E_u - E_a$) but not enough to make it normalised. It then bypasses the alignment shifter through the bypass-2 multiplexer.

- $E_u \geq E_a + E_{dist}$: in this case, both significands need to be shifted. M_u must be normalised and M_a must be right shifted by the exponent difference less

the normalisation distance ($E_u - E_a - E_{dist}$). E_{dist} can be in error by 1-bit which can be detected by the multiplexer directly after the normalisation shifter and is applied to the bypass-2 multiplexer to correct the shifted significand.

The length of the critical path has increased by an 11-bit adder delay, however speculative calculations for $E_a - E_b$, $E_b - E_a$, $E_a - E_u$ and $E_u - E_{dist} - E_a$ and multiplexing the result would only increase the critical path by two multiplexer delays.

In non-accumulation mode, the adder has the same architecture and 3-cycle latency as the FP adder in previous section. The post-normalisation step is independent of the in-coming operands and the result is taken from M_r.

4. A 2-cycle Floating Point Adder Architecture

In this paper, an FP adder was designed which merged rounding with the significand addition phase. A 2-cycle FP adder can be constructed which has separate data paths depending on whether or not the data requires a large alignment shift or large normalisation shift as shown in Figure 5. This is a speculative FP adder architecture and uses more hardware than the previous design. A multiplexer at the output of the near and far data paths selects the correct result at the completion of the second pipeline stage. As noted in the introduction, some recent microprocessors have used this method to speed up FP addition [12],[5].

The far data path calculates the result for all true addition operations and true subtraction operations which have an exponent difference greater than or equal to 2. In this case true subtraction of the significands leads to a result which is always greater than one half and at most needs to be left shifted by one bit, whereas a true addition can only lead to the significand being shifted one place to the right if there is an overflow. This is done with multiplexers at the output of the far pipe-2 flagged prefix adder. The far data path also computes results for some cases where the exponent difference is "1" which are discussed below.

The near data path computes the true subtraction cases where the exponent difference is "0" and (in some cases) "1". These computations may need a large number of result significand left shifts for normalisation due to massive cancellation of the significands. The result of the near data path is calculated speculatively as it is not known until well into the first pipeline stage if the result will be needed.

The near pipe-1 stage uses a carry propagate adder to add the significands which forms the critical path. To start the addition as early as possible, thereby minimising the cycle time, it is only necessary to check if the exponent LSBs differ to speculate if the exponents differ by "1" and one of the significands is right shifted by 1-bit. If the exponent LSBs are equal and it turns out at the end of far pipe-1 that the exponents are equal, the significand result is exact, requires no rounding, and a normalisation shift then occurs in near pipe-2.

Sig.MSBs $M_a(51)$, $M_b(51)$	$E_a - E_b = 1$		$E_a - E_b = -1$	
	Range (g.t.,l.t.)	Norm. Shift	Range (g.t.,l.t.)	Norm. Shift
00	0.25, 1	2	0.25, 1	2
01	0, 0.75	54	0.75, 1.5	1
10	0.75, 1.5	1	0, 0.75	54
11	0.5, 1.25	1	0.5, 1.25	1

Table 1. Normalisation shifts needed for an exponent difference of 1.

If the exponent LSBs are different, it then needs to be determined which of the significands is to be right shifted by one bit. A speculative regime could calculate both $M_a - M_b/2$ and $M_b - M_a/2$ using two significand adders and multiplex the correct result once the exponent difference is known, which has been calculated in parallel with the significands.

A new method is proposed to determine which of the significands should be right shifted by one bit, thereby eliminating one of the adders and the multiplexer.

If we inspect the MSBs of the significands (before unpacking), there are four cases to consider which depend on the sign of the exponent difference as shown in Table 1.

Only one of the cases where the exponent bits differ is where massive cancellation of result bits can occur requiring large normalisation shifts. The other case where the exponent bits differ (eg. "10" for $E_a - E_b = 1$) and the "11" case result in at most 1-bit left shift and the "00" case results in at most 2-bits left shift. The cases where the MSBs are equal ("00" and "11") can be handled by the far data path providing the normalising multiplexer in far pipe-2 is modified to perform a 2-bit left shift. If the MSBs are not equal ("01" and "10"), only one of these two cases can lead to large normalisation shifts, so we always choose to speculatively calculate $M_a - M_b/2$ when M_a has a "0" in the MSB and M_b has an MSB equal to "1", or calculate $M_b - M_a/2$ when M_a has a "1" in the MSB and M_b has a "0" in the MSB. The operand with a "1" in the MSB is always right shifted and the case where the MSB bits are reversed is dealt with in the far data path. These results are always exact requiring no rounding thus simplifying the near path flagged prefix adder. Choosing which data path to use is summarised in Table 2.

A circuit to determine which operand to shift can be built with one CMOS complex gate and the need for an 11-bit exponent subtraction to determine the sign of the exponent difference has been avoided. The only extra hardware needed

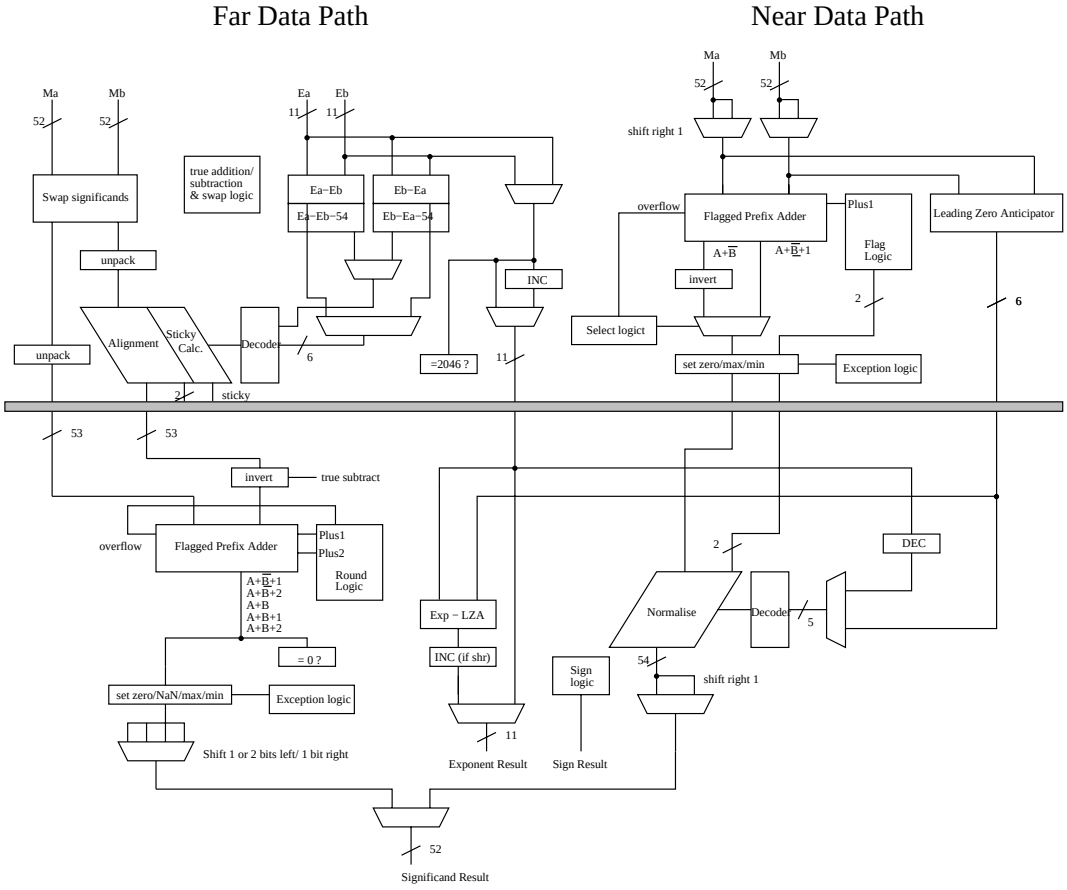

Figure 5. A 2-cycle floating-point adder.

Exp.LSBs A_{e0} B_{e0}	Sig.MSBs $A_m(51)$, $B_m(51)$	near/far data path	near data path alignment shift
same $(00, 11)$	-	near	no shift
differ $(10, 01)$	00	far	-
”	01	near	B_m right 1-bit
”	10	near	A_m right 1-bit
”	11	far	-

Table 2. Determining which significand to shift as a function of significand MSBs and exponent LSBs.

for the 2-cycle FP adder compared to the one shown in Figure 1 is an extra flagged prefix adder. Further speed improvements can be made in far pipe-1 by duplicating the alignment shifter and computing the two exponent differences in parallel with the alignment shifts [9].

5. Acknowledgement

This work was gratefully supported by the Australian Research Council and The Defence Science & Technology Organisation, Australia. Thanks to Kiet To for designing the boundary scan cells, Warren Marwood for providing test equipment, Mike Liebelt and Amos Omondi for helpful discussions and the anonymous referees for their helpful remarks.

References

[1] A. Beaumont-Smith, M. Liebelt, C. Lim, K. To, and W. Marwood. A digital signal multi-processor for matrix applications. In *Proc. 14th IREE Australian Microelectronics Conference*, pages 245–250, Sep. 1997.

[2] N. Burgess. The flagged prefix adder for dual additions. In *Proc. SPIE ASPAAI-7*, volume 3461, pages 567–575, San Diego, Jul. 1998.

[3] M. Farmwald. *On the Design of High Performance Digital Arithmetic Units*. PhD thesis, Stanford University, Aug. 1981.

[4] D. Goldberg. *Computer Architecture A Quantitative Approach*, chapter Appendix A. Morgan Kaufmann, 1990.

[5] D. Greenley et al. Ultrasparc: the next generation superscalar 64-bit sparc. In *Digest of Papers, COMPCON 95*, pages 442–451, Mar. 1995.

[6] Y. Hagihara, S. Inui, F. Okamoto, M. Nishida, T. Nakamura, and H. Yamada. Floating-point datapaths with online built-in self speed test. *IEEE J. Solid-State Circuits*, 32(3):444–449, Mar. 1997.

[7] C. Heikes and G. Colon-Bonet. A dual floating point coprocessor with and fmac architecture. In *ISSCC Dig. Tech. Papers*, pages 354–355, Feb. 1996.

[8] E. Hokenek and R. Montoye. Leading-zero anticipator (lza) in the ibm risc system/6000 floating-point execution unit. *IBM J. Res. Develop.*, 34(1):71–77, Jan. 1990.

[9] S. Knowles. Arithmetic processor design for the t9000 transputer. In *Proc. SPIE ASPAAI-2*, San Diego, 1991.

[10] L. Kohn and S. Fu. A 1,000,000 transistor microprocessor. In *IEEE Int'l Solid-State Circuits Conf. Dig. Tech. papers*, pages 54–55, 1989.

[11] Y. Kondo, Y. Koshiba, Y. Arima, M. Murasaki, T. Yamada, H. Amishiro, H. Mori, and K. Kyuma. A 1.2gflops neural network chip for high-speed neural network servers. *IEEE J. Solid-State Circuits*, 31(6):860–864, Jun. 1996.

[12] J. Kowaleski Jr., G. Wolrich, T. Fischer, R. Dupcak, P. Kroesen, T. Pham, and A. Olesin. A dual-execution pipelined floating point cmos processor. In *ISSCC Dig. Tech. Papers*, pages 358–359, Feb. 1996.

[13] R. Montoye, E. Hokenek, and S. Runyon. Design of the ibm risc system/6000 floating-point execution unit. *IBM J. Res. Develop.*, 34(1):59–70, Jan. 1990.

[14] S. Naffziger. A sub-nanosecond 0.5mm 64b adder design. In *ISSCC Dig. Tech. Papers*, pages 362–363, Feb. 1996.

[15] A. Nielsen, D. Matula, C. Lyu, and G. Even. Pipelined packet-forwarding floating point: II. an adder. In *Proc. IEEE 13th Int'l Symp. on Computer Arithmetic*, pages 148–155, Asilomar, California, Jul. 1997.

[16] S. Oberman, H. Al-Twaijry, and M. Flynn. The snap project: Design of floating point arithmetic units. In *Proc. IEEE 13th Int'l Symp. on Computer Arithmetic*, pages 156–165, 1997.

[17] V. Oklobdzija. An algorithmic and novel design of a leading zero detector circuit: Comparison with logic synthesis. *IEEE Transactions on VLSI Systems*, 2(1):124–128, Mar. 1994.

[18] A. Omondi. *Computer Arithmetic Systems, Algorithms, Architecture and Implementations*. Prentice Hall, 1994.

[19] W.-C. Park, S.-W. Lee, O.-Y. Kown, T.-D. Han, and S.-D. Kim. Floating point adder/subtractor performing ieee rounding and addition/subtraction in parallel. *IEICE Transactions on Information and Systems*, E79-D(4):297–305, Apr. 1996.

[20] S. Peng, S. Samudrala, and M. Gavrielov. On the implementation of shifters, multipliers, and dividers in vlsi floating point units. In *Proc. IEEE Int'l Symp. on Computer Arithmetic*, pages 95–102, 1987.

[21] N. Quach and M. Flynn. Design and implementation of the snap floating-point adder. Technical Report CSL-TR-91-501, Stanford University, Dec. 1991.

[22] N. Quach, N. Takagi, and M. Flynn. On fast ieee rounding. Technical Report CSL-TR-91-459, Stanford University, Jan. 1991.

[23] R. Simar Jr. Codevelopment of the tms320c6x velociti architecture and compiler. In *Proc. ICASSP'98*, Seattle Washington, May 1998.