Analysis and Implementation of a Novel Leading Zero Anticipation Algorithm for Floating Point Arithmetic Units

M. Olivieri, F. Pappalardo, S. Smorfa, and G. Visalli

***Abstract*— In multimedia Systems-on-Chips, the design of specialized IEEE-754-compliant floating point arithmetic units (FPU) is critical with respect to both operating speed and silicon area demand. Leading zero anticipation is a well-known issue in the implementation of high speed FPUs. We investigated a novel leading zero anticipation algorithm allowing us to significantly reduce the anticipation failure rate with respect to the state-of-the art. We embedded our technique into a complete FPU and compared its performance against existing solutions, definitely showing both area savings and total latency reduction.**

***Index Terms* — floating-point arithmetic, leading zero anticipation, floating-point unit, CMOS VLSI.**

I. Introduction

The increased interest in ASICs and SoCs for multimedia applications justifies the effort in the design of high-speed arithmetic units where either fixed or floating-point real number representations are exploited for calculations. In particular, high-throughput floating-point signed adders (FADD) lend themselves to a sophisticated design approach due to both the amount and complexity of the required operations (operands alignment, addition, renormalization and rounding). In case of effective subtraction (subtraction of operands weith the same sign or addition of operands with different signs), the number of leading digits in the output must be calculated to initiate the renormalization procedure. Since renormalization occupies the dominant portion of an FP addition, performing the leading digits calculation after the addition is ineffective and some reliable anticipation logic must be devised for parallel anticipation of the position of the first significant digit. For practical implementations, when both leading zeroes and ones may occur, either case can be handled by a separate anticipator [4][5]. Otherwise, the assumption that the smaller operand is always subtracted from the larger one allows to limit the anticipation of leading digits to leading zeroes [6]. Besides, the leading zeroes anticipatory (LZA) logic is not needed when the exponents of the operands differ by more than one, i.e. in the so-called far-path (FP): designs of FADDs with separate far-path and LZA-based near-path (NP) logics have been extensively discussed [2][9].

Leading zero anticipation requires that a binary string be generated to exhibit approximately the same number of leading zeroes as the final result. The string is processed by the leading zero counter (LZC) to work out the number of shifting positions needed for renormalization. Many different strategies can be adopted for leading zero anticipation, each one posing a trade-off between hardware complexity and correctness of prediction. In [10], the authors provide an overview of a number of alternative approaches. Solutions for the exact estimation of the number of leading zeroes have been proposed [3][11], but they all suffer form sizable hardware realizations. On the other hand, far simpler LZA logic can be developed at the expense of a 1-bit uncertainty in the leading digit detection; as a matter of fact, they are commonly used both in fixed (e.g. division) and floating-point (e.g. FADD) arithmetic blocks. As explained in [6][7], for all such inexact LZA techniques the anticipated leading digit position can be a single bit left of the correct one, so that an extra left shift might be needed for correctness of renormalization. There exist diverse error detection methods [10] to make up for failed anticipation. When error detection directly applies to the addition result, the extra shift can be easily absorbed in the last stage of a multistage realization of the renormalization shifter. Conversely, error detection can be carried out in parallel with the leading zero anticipation, so that the correction control signal can be used within the very first stage of the normalization shifter. Approaches where concurrent error detection and correction result into an overall error-free LZA circuit are illustrated in [8] [13].

Error detection and correction inevitably lead to higher hardware complexity and increased circuit area, which may represent a major shortcoming for designs strictly constrained by low-power consumption. In addition, most of the multimedia applications can tolerate the event of LZA misprediction with negligible perceptual drawbacks. In this scenario, advancement in the field of low-failure-rate LZA is crucial for the possibility of eliminating error correction circuitries in favor of simpler, cheaper and faster

Manuscript received October 9, 2001. (Write the date on which you submitted your paper for review.)

M. Olivieri and S. Smorfa are with the Dept. of Electronic Engineering, “La Sapienza” University of Rome, Rome, Italy (phone: 0039064820282; fax: 0039064742647; e-mail: {olivieri,smorfa} @die.uniroma1.it). F. Pappalardo and G. Visalli are with STMicroelectronics, Catania, Italy (e-mail: {francesco.pappalardo,giuseppe-ast.visalli}@st.com).

implementations. Here we propose a novel LZA where the efficiency of the leading digit prediction is enhanced by dedicated carry estimation logic. We investigated the performance of our method by measuring the occurrences of misprediction over diverse benchmarks and compared the results with other existing techniques. The proposed LZA has been implemented and included in a complete floating-point unit (FPU) that represents the FP extension for an STMicroelectronics microprocessor core. The HDL description of the whole unit has been synthesized in a 180nm and a 90nm CMOS technologies, and has been operated up to a frequency of 2.17 GHz. The area occupation and maximum latency are presented and compared with other state-of-the-art solutions.

In the following, Section II summarizes the principles of leading zero detection using anticipatory logic and introduces our method. Section III sheds insights into the performance of all approaches, from the point of view of the misprediction rate. Section IV illustrates the architecture of the new FPU and compares its actual speed and area with other works. Section V summarizes the conclusions.

II. Leading Zero Anticipation Approaches

A. Previous works

When an FPU performs effective subtraction, the final unrounded mantissa produced by the internal integer adder may exhibit leading zeroes. LZAs make use of the propagate (*T*), generate (*G*), and kill (*Z*) functions, defined as follows:

$$T = A \oplus B\,,\quad G = AB\,,\quad Z = \overline{A}\,\overline{B} \tag{1}$$

and calculated for each bit position of the adder inputs *A* and *B* after swapping, alignment and inversion. In the following, the bit positions will be numbered such that bit 0 is the most significant. A general expression for the leading zeroes indicator is:

$$\begin{cases} f_i = T_i \oplus \overline{Z}_{i+1} & 0 \le i \le n-1, \\ f_n = 1 \end{cases} \tag{2}$$

where *n+1* is the number of bits in the adder operands (1-bit-sign-extended *n*-bit operands). Under the assumption that the smaller operand is always subtracted from the larger one, i.e. when leading ones cannot occur, (2) yields:

$$\begin{cases} f_i = \overline{T}_i\overline{Z}_{i+1} & 0 \le i \le n-1 \\ f_n = 1 \end{cases} \tag{3}$$

as shown in [6]. The resulting LZA logic indicates the position of the leading digit with 1-bit of uncertainty, so that it can be either *i* or *i+1*. Chang et al. investigated the possibility of reducing the misprediction rate of (2) and (3) by looking ahead more than 1 bit right of a given position for a subtler estimation of the carry-in contribution at that position [7]:

$$\begin{cases} f_i = \overline{T}_i\overline{Z}_{i+1}\ldots\overline{Z}_{i+k} + T_i Z_{i+1} & 0 \le i \le n-1-k \\ f_i = \overline{T}_i\overline{Z}_{i+1}\ldots\overline{Z}_{n} + T_i Z_{i+1} & \text{n - k} \le i \le n-1 \\ f_n = 1 \end{cases} \tag{4}$$

In (4), k is a tunable parameter which determines the depth of the carry estimation procedure. It is apparent that (4) corresponds to (1) when *k=1*.

B. Our method

In this work, we propose an LZA definition that better exploits the carry-in C^{in}_i generation logic at any bit position *i* to further reduce the failure rate. As in [7], the principle of operation is to truncate the carry propagate chain and introduce a carry-in estimator, whose Boolean expression can be derived by inspecting the Karnaugh maps for C^{in}_i. Fig. 1.a shows them when the sole inputs A_{i+1}, B_{i+1}, and C^{in}_{i+1} are considered. All logic 1s in the Boolean function are covered if C^{in}_i is approximated as in Fig. 1.b:

$$\begin{cases} \tilde{C}^{in}_i = A_{i+1} + B_{i+1} = \overline{Z}_{i+1} & 0 \le i \le n\,, \\ \tilde{C}^{in}_{n+1} = 1 \end{cases} \tag{5}$$

which is independent of C^{in}_{i+1}. Using (5) for the carry-in leads to the LZA in (1). Fig. 2 shows the Karnaugh maps for C^{in}_i using A_{i+1}, B_{i+1}, A_{i+2}, B_{i+2} and C^{in}_{i+2} as inputs. The approximating function of C^{in}_i covering all logic 1s is thereby:

$$\begin{cases} \tilde{C}^{in}_i = A_{i+1}B_{i+1} + (A_{i+1} + B_{i+1})(A_{i+2} + B_{i+2}) = \\ = G_{i+1} + \overline{Z}_{i+1}\overline{Z}_{i+2} & 0 \le i \le n-1 \\ \tilde{C}^{in}_n = A_{n+1}B_{n+1} + (A_{n+1} + B_{n+1}) \\ \tilde{C}^{in}_{n+1} = 1 \end{cases} \tag{6}$$

(5) and (6) can be taken as 1st and 2nd-order estimators of the carry-in. By iterating the method, *m*th-order estimators can be recursively expressed as:

$$\begin{cases} \tilde{C}^{in}_{i,m} = G_{i+1} + \overline{Z}_{i+1}\,\tilde{C}^{in}_{i+1,m-1} & 0 \le i \le n\,. \\ \tilde{C}^{in}_{n+1,m} = 1 & \forall m \end{cases} \tag{7}$$

Using the carry-in estimators, the definition of the *m*th-order leading digit indicators can be generalized by the following:

$$\begin{cases} f_i = T_i \oplus \tilde{C}^{in}_{i+1,m} & 0 \le i \le n\,, \\ f_{n+1} = 1 \end{cases} \tag{8}$$

Performance of the indicators defined by equations (3), (4) and (8) will be discussed in Section III.

C. Extension to Leading-Ones Predictors

The hypothesis that that the smaller operand is always subtracted from the larger one can be dropped when a leading-ones anticipator is also exploited. The format of a general leading ones anticipator can be expressed as:

$$\begin{cases} f_i^{ones} = T_i \oplus \tilde{\tilde{C}}_{i+1,m}^{in} & 0 \le i \le n, \\ f^{ones}{}_{n+1} = 1 \end{cases} \quad (9)$$

and our method for calculating the carry-in estimator at a given order can be extended to the $\bar{C}_i^{in}$ estimator by Karnaugh maps inspection. This yields:

$$\begin{cases} \tilde{\tilde{C}}_{i,m}^{in} = \bar{G}_{i+1} \bullet \left(Z_{i+1} + \tilde{\tilde{C}}_{i+1,m-1}^{in} \right) & 0 \le i \le n. \\ \tilde{\tilde{C}}_{n+1,m}^{in} = 1 & \forall m \end{cases} \quad (10)$$

The anticipator (9) can be used in conjunction with (8); examples of the combined use of leading digit indicators can be found in [4][5][10].

III. Performance Investigation

A. Failure-rates Minimization

With the intent of comparing the effectiveness of our LZA with previous solutions, we performed functional simulations over three different sets of benchmarks. We referred to the effective subtraction of IEEE-754-compliant single-precision floating-points numbers [1]. The first set is an exhaustive benchmark covering all possible pairs of operands belonging to the NP (i.e., significands of floating-point numbers whose exponents are equal or differ by at most ±1) at a given operand width. The second set comprises 10^6 pairs of random generated operands. The third set is the Berkeley suite [14], aimed at testing some special cases of floating-point numbers (such as infinity and NaN).

We simulated several techniques, referred to as the following acronyms:

- *Szk* : Eq. (3), according to [6];
- *Ch2*: Eq. (4) with *k*=2, according to [7];
- *Ch3*: Eq. (4) with *k*=3, according to [7];
- *LZA2*: our method with 2nd order;
- *LZA3*: our method with 3rd-order.

Table I categorizes the percentages of leading zero anticipation failures over the number of tested operations. Ch2 and Ch3 are to be compared with LZA2 and LZA3 respectively. The results evidence that our C^{in} estimator significantly outperforms previous techniques according to both exhaustive or random performance analysis. Besides, although on average Ch2 performs better than Szk, there exist cases when (3) correctly predicts the leading digit position whereas (4) fails. Fig. 3.a and 3.b provide two 6-bit operand examples when either Szk or Ch2 methods respectively fail, while LZA2 yields the correct prediction (*S* denotes the true output). In fact, exhaustive investigation indicated that LZA*m* (*m* being the order of the C^{in} estimator) *never* fails when Szk and Ch2 yield the correct anticipation. In addition, increasing the estimator order always enhances the performance of LZA*m*, which does not apply to Ch2 and Ch3 (see Table I).

B. Implementation Cost

Referring to an AND-OR realization of the LZA*m* indicator, the number of required logic levels can be calculated to be *2m+1*. We developed and synthesized the VHDL RTL descriptions of *all* methods reported in Table I. Table II reports the resulting area occupation in 0.35 μm, 0.13 μm and 90 nm standard cell CMOS technologies. The reduction in the failure rate comes at the expense of about 50% area overhead when migrating from Ch2 to LZA3, but this is compensated by the elimination of correction units. Also, note that Table II actually presents worst case data, for most of the carry propagate logic used by LZA can be shared with fast adder structures (e.g. Brent-Kung) used inside the FPU [6][7]. Besides, when the LZA is embedded into a complete FPU, the induced area overhead is practically negligible when compared to the area of the whole unit. Finally, the format of our anticipators (8) lends itself to straightforward porting to FPUs where leading-ones are also admitted (see Section II.C), which does not apply to the simplified (3) and (4).

C. Optimal Sizing of Carry-Estimators

In the view of optimizing the use of our indicators, Fig. 4 charts the occurrences of *N* leading zeroes in the adder outputs over the random and Berkeley benchmarks (23-bit significand operands, 24-bit inputs to the adder). Values of *N* such that the corresponding occurrence of *N*-leading-zero outputs was approximately zero have not been displayed. Since the number of leading zeroes in most outputs ranges from 0 to 3, it is sensible to enhance the robustness of the LZA at the 3 most significant bit positions. We experimented a combined indicator (LZA-CMB) exploiting 3rd-order $C^{in}{}_i$ estimators in positions 0, 1, and 2, and 2nd-order estimators in positions 3 to 24. Correspondingly, inspection of Table I and II shows an interesting tradeoff between performance and area demand.

IV. The High Speed Floating-Point Arithmetic Unit

A. Overview

We designed a complete FPU including our LZA-CMB circuit, intended as an FP extension for STMicroelectronics microprocessor cores. The unit performs signed addition (FADD), subtraction (FSUB) and conversions from (FCONV_IS) and to (FCONV_SI) 32-bit integers (INT32). FP numbers are represented according to the IEEE-754 single-precision standard with normalized encoding only (denormalized inputs/outputs are flushed to zero). Overall, a 3-stage pipelined dual-path architecture has been used, where either the FP or NP are exclusively enabled based on the exponent difference and the type of calculation.

Fig. 5 sketches the top-level view of the NP, which is responsible for effective subtraction when |exponent difference| < 2; our LZA logic has been included therein. Operand swapping and alignment is performed in the first stage, along with inversion of the smallest operand (subtrahend). In order to speedup execution at the expense of some logic replication, the 4 possible operands configurations

are pre-computed in parallel and the proper operands pair is then fed into the second stage depending on the exponents difference and significands magnitudes. The second stage accommodates the 25-bit adder and the LZA-CMB/LZC logics. Finally, renormalization is implemented in the third stage by means of a multistage barrel shifter. As discussed in previous sections, the scope of our FPU allowed us to take advantage from the increased LZA precision to remove the error compensation circuitry. FCONV_IS is also accomplished by the NP and entails some preliminary data formatting (in the first stage; the significand is calculated as the 2's complement of the input for negative integers) and renormalization (in the third stage; to this end, the LZC circuits is also exploited).

Operations different from effective subtractions with constrained exponents differences are committed to the FP. In the first stage, the smaller operand is shifted for alignment and the possible final exponents are pre-computed (the possible exponents differ by 1 depending on the need for 1-bit renormalization in the final stage). In case of subtraction, the subtrahend is also inverted. In order to achieve significant speedup with respect to traditional architectures, the second stage exploits 4 separate adders to pre-calculate in parallel all possible contributions to the output significand. Different contributions would be used when rounding is needed or overflow occurs. As much as in the NP, Brent-Kung adders have been used for minimum delay. Injection-based rounding to nearest even (RNE) [15] is the sole supported rounding algorithm. The final significand is set in the third stage together with the output exponent; if necessary, 1-bit renormalization is also performed therewith.

As for FCONV_SI, it is executed by an extra unit and includes parsing the floating-point input, calculating the required number of shift positions and verifying the compatibility with the 32-bit integer range in the first stage. The actual shift is performed in the second stage, whereas the integer output is formatted in the third stage and saturated in case range underflow or overflow conditions occur.

B. Performance Evaluation

The new FPU has been synthesized in 90 nm CMOS standard cell technology; we also synthesized the same FPU architecture equipped with a standard LZA [6] with error correction. The results of the two designs are reported in Table III (design *A* and design *B*, respectively). The maximum frequency is determined by the maximum delay through the 2nd stage of either the FP for design *A* or the NP for design *B*. Our unit can be run at a frequency up to 2.17 GHz, though with some area overhead due to the logic replication in the FP. Moreover, with design *A* we achieved up to 8.5% increase in the maximum operating frequency and a 13% area reduction with respect to design *B*. It must be stressed that the latency of the adder's I/O path in the NP hides the maximum delay through the LZA/LZC pairs (see sixth and seventh rows in Table III). Therefore, our LZA circuit does not affect the critical path of the FPU, as requested for the effectiveness of the anticipation principle.

Table IV compares our NP design with two different NP state-of-the-art proposals for dual-path FPUs. The architecture in [12] achieves optimal delay performance by means of full-custom transistor-level-optimized design. With respect to the standard cell realization in [13], we attained an absolute 84% reduction in both area and critical path latency, and 37% area / 59% latency reduction when referring to an equivalent technology as in [13] (despite the insertion of pipeline registers and extra logic for additional operations, such as FCONV_IS).

V. Conclusion

We presented a new approach for leading zero anticipation in floating-point arithmetic units dedicated to multimedia algorithms, avoiding leading zero count correction. We demonstrated that specializing the carry estimation logic allows to significantly reduce the anticipation failure rate with respect to existing techniques. Our LZA circuit has been embedded into a complete FPU representing the floating-point extension for STMicroelectronics microprocessor cores. The unit, performing floating-point addition and subtraction as well as INT32-to-FP and FP-to-INT32 conversions, turns out to be definitely smaller and faster in the near-path with respect to other state-of-the-art solutions.

References

[1] Standard for binary floating-point arithmetic, IEEE Standard 754, 1985.
[2] H.P. Sit, et al., "Prenormalization for a floating-point adder," U.S. Patent 5010508, April 1991.
[3] G. Inoue, "Leading one anticipator and floating point addition/subtraction apparatus," U.S. Patent 5343413, August 1994.
[4] S. Britton, R. Allmon, and S. Samudrala, "Leading one/zero bit detector for floating-point addition," U.S. Patent 5317527, May 1994.
[5] M.S. Schmookler, and D. Mikan, "Two-state leading zero/one anticipator (LZA)," U.S. Patent 5493520, February 1996.
[6] H. Suzuki, H. Morinaka, H. Makino, Y. Nakase, K. Mashiko, and T. Sumi, "Leading-zero anticipatory logic for high-speed floating-point addition," *IEEE J. of Solid-State Circuits*, 1996, vol. 31(8), pp. 1157-1164.
[7] T. Chang, J. Huang, and S. Yang, "Leading-zero anticipatory logics for fast floating addition with carry propagation signal," *Proc. 40th Midwest Symposium on Circuits and Systems*, 1997, vol. 1, pp. 385-388.
[8] J.D. Bruguera, and T. Lang, "Leading-one predication with concurrent position correction," *IEEE Trans. Computers*, 1999, vol. 48(10), pp. 1083-1097.
[9] S. Oberman, and M. Roberts, "Leading one prediction unit for normalizing near-path subtraction results within a floating point arithmetic unit," U.S. Patent 6085208, July 2000.
[10] M.S. Schmookler, and K.J. Nowka, "Leading zero anticipation and detection – a comparison of methods," *Proc. 15th IEEE Symposium on Computer Arithmetic*, 2001, pp. 7-12.
[11] H. Sun, W. He, and M. Gao, "Designing leading zeros anticipatory logic based on production rules," *Proc. 5th International Conference on ASIC*, 2003, vol. 2, pp. 1260-1264. H. Sun, and M. Gao, "Unified bit pattern for leading-zero anticipatory logic for high-speed floating-point addition," *Proc. 3rd IEEE International Symposium on Signal Processing and Information Technology*, 2003, pp. 786-789.
[12] P. Seidel, and G. Even, "Delay-optimized implementation of IEEE floating-point addition," *IEEE Trans. Computers*, 2004, vol. 53(2), pp. 97-113.
[13] G. Zhang, Z. Qi, and W. Hu, "A novel design of leading zero anticipation circuit with parallel error detection," *Proc. IEEE*

International Symposium on Circuits and Systems, 2005, vol. 1, pp. 676-679.

[14] Berkeley Floating-Point Test Suite http://www.netlib.org/fp/ucbtest.tgz

[15] G. Even, and P. Seidel, "A Comparison of three Rounding Algorithms for IEEE Floating-Point Multiplication", *IEEE Trans. On Computers* , 2000, vol. 49(7), pp. 638-650.

TABLE I
FAILURE RATES

Benchmark	*Szk*	*Ch2*	*Ch3*	*LZA2*	*LZA3*	*LZA-CMB*
exhaustive [a]	37.48	24.98	34.35	18.73	9.36	12.48
random	0.51	0.33	0.45	0.26	0.13	0.16
Berkeley	10.74	9.84	9.84	9.84	9.84	9.84

All data are expressed as percentages #failures/#tests × 100.

[a] Due to computational complexity, exhaustive exploration has been run for 16-bit operands.

TABLE II
AREA RESULTS OF LZA APPROACHES (IEEE-754 SINGLE-PRECISION)

LZA	*CMOS 0.35μm*	*CMOS 0.13μm*	*CMOS 90nm*
Szk	4860 μm^2	908 μm^2	494 μm^2
Ch2	7344 μm^2	1180 μm^2	642 μm^2
Ch3	6804 μm^2	1230 μm^2	670 μm^2
LZA2	9126 μm^2	1585 μm^2	862 μm^2
LZA3	9558 μm^2	1787 μm^2	971 μm^2
LZA-CMB	9288 μm^2	1636 μm^2	889 μm^2

TABLE III
COMPLETE FPU DESIGN DATA (90nm CMOS)

Performance Parameter	*Design A*	*Design B*
Total Area	58200 μm^2	67000 μm^2
NP Area	12100 μm^2	18400 μm^2
Max. Delay (2nd stage)	0.46 nsec	0.5 nsec
Max. Operating Frequency	~ 2.17 GHz	~ 2 GHz
LZA+LZC Delay (NP)	0.23 ns	0.3 ns
Adder Delay	0.3 ns	0.33 ns

TABLE IV
COMPARISON WITH PREVIOUS *NP* DESIGNS

Performance Parameter	*This Work*		*[12]*	*[13]*
Technology	90nm	180nm	180nm	*technology-independent*
Pipeline stages	3	3	1	2
Area	12100	48400	76272	-
Max. Stage Delay	0.38	1.0	2.43	15.3 FO4
Max. Total Delay	0.92	2.4	2.43	30.6 FO4

Areas expressed in μm^2; delays expressed in ns. FO4 denotes the delay of an inverter with fanout 4 in the target technology.

C_i^{in} \ $A_{i+1}B_{i+1}$	00	01	11	10
0	0	0	1	0
1	0	1	1	1

(a)

C_i^{in} \ $A_{i+1}B_{i+1}$	00	01	11	10
0	0	1	1	1
1	0	1	1	1

(b)

Fig. 1. (a) Karnaugh map for C^{in}_i as a function of A_{i+1}, B_{i+1}, and C^{in}_{i+1} (b) Corresponding approximation for 1st order estimate.

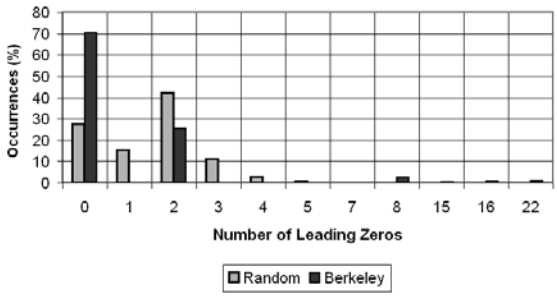

Fig. 4. Leading Zeroes Distribution.

$A_{i+1}B_{i+1}$ \ $A_{i+2}B_{i+2}$	00	01	11	10
00	0	0	0	0
01	0	0	1	0
11	1	1	1	1
10	0	0	1	0

$C^{in}_{i+2} = 0$

$A_{i+1}B_{i+1}$ \ $A_{i+2}B_{i+2}$	00	01	11	10
00	0	0	0	0
01	0	1	1	1
11	1	1	1	1
10	0	1	1	1

$C^{in}_{i+2} = 1$

Fig. 2. Karnaugh map for C^{in}_i as a function of A_{i+1}, B_{i+1}, A_{i+2}, B_{i+2}, and C^{in}_{i+2}.

```
A:      0110101          A:      0110110
B:      1100011  +1      B:      1110000  +1
-------------------      -------------------
S:      0011001          S:      0100111
Szk:    0101001          Szk:    0101001
Ch2:    0011001          Ch2:    0001011
LZA2:   0011001          LZA2:   0101111
```

(a) (b)

Fig. 3. (a) Failed anticipation with Szk; (b) Failed anticipation with Ch2.

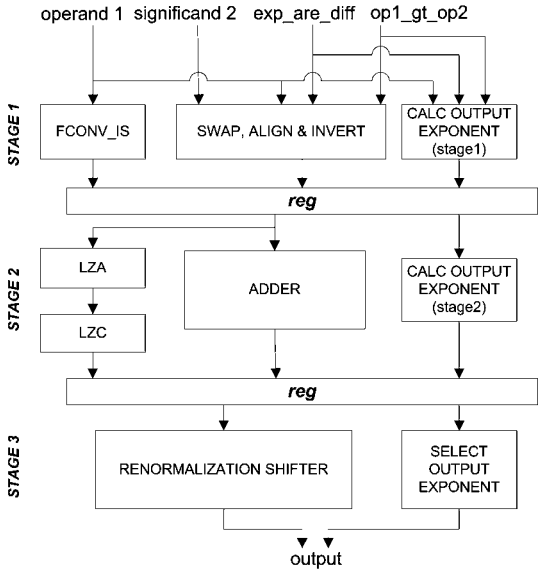

Fig. 5. Block view of the near-path.