Leading Zero Anticipation and Detection -- A Comparison of Methods

Martin S. Schmookler[1] and Kevin J. Nowka[2]
[1]IBM Server Development and [2]IBM Austin Research Laboratory
Austin, Texas USA
martins@austin.ibm.com, nowka@austin.ibm.com

Abstract

Design of the leading zero anticipator (LZA) or detector (LZD) is pivotal to the normalization of results for addition and fused multiplication-addition in high-performance floating point processors. This paper formalizes the analysis and describes some alternative organizations and implementations from the known art. It shows how choices made in the design are often dependent on the overall design of the addition unit, on how subtraction is handled when the exponents are the same, and on how it detects and corrects for the possible one-bit error of the LZA.

1. Introduction

Leading zero anticipators predict the location of the most significant bit location of the result of a floating-point addition directly from the inputs to the adder. This determination of the leading digit position is performed in parallel with the addition step so as to enable the normalization shift to start as soon as the addition completes. Many different solutions to the problem of designing an LZA have appeared in publications and patents. They have varying degrees of complexity, and some operate only on restricted cases. This paper describes what appear to be the simplest solutions for both the general and the restricted cases. It also includes a design that has not been previously published except in a patent [1], but which is used in several commercial processors.

The typical LZA consists of the generation of a string of bits having approximately the same number of leading zeros as the sum output. An LZD is then employed to encode the result. Several methods of designing the LZD are available, and the best choice often depends on the adder design and on how the string of bits is created.

LZDs are frequently used in fixed point arithmetic units also. A Count Leading Zeros (CLZ) instruction is often part of the fixed point instruction set, and the counting of leading digits of the divisor may be needed for some fixed point divide algorithms. Techniques that are known for speeding up LZDs can also be used for the encoding of an LZA. Therefore, this paper includes brief descriptions of two methods for efficiently obtaining a leading zero count.

The LZA can also detect the cases when the result of addition is all zeros. This too is a function which is useful in both fixed point and floating point units. Therefore, some discussion of zero result predictors is included as well.

The earliest description of an LZA known to the authors is by Kershaw, et al [2][3] which shows a Manchester carry adder circuit with a second precharged circuit used for the detection of the leftmost significant digit. It works for both leading zeros when the result of a subtraction is positive, and for leading ones when the result is negative. This basic algorithm is also used in the T9000 Transputer described by Knowles [4].

An LZA described by Hokenek and Montoye [5] also handles the general case of leading ones or zeros. Because this method is more complex and slower than efficient implementations of the Kershaw method, we do not describe this design in detail in this paper.

Since then, Britton et al [6] and Suzuki et al [7] have shown that a much simpler circuit can be used when one can assume that the subtraction result will be positive. Further simplification is obtained when one also assumes that the exponents differ by one, as shown by [8] and [9].

Most of the LZAs which are described are inexact. They only examine the inputs from left to right, ignoring a possible carry from the right for each bit that it predicts to be part of the leading string of zeros. Several papers [10] [11] [12] have also been published describing exact LZAs which do take into account carries from the right, but they generally result in excessive complexity and delay. However, one exact LZA [13] is described briefly because it is simple, and has delay comparable to that of the adder.

0-7695-1150-3/01 $10.00 © 2001 IEEE

An alternative to the exact LZA is to an error indicator in parallel with the LZA computation. The Kershaw LZA includes a circuit which uses the carries from the right to generate a single error signal for the LZA which can be used to adjust the controls to the last stage of a multistage normalizing shifter. The circuit is relatively simple, and the error signal can be developed in parallel with the earlier stages of the normalizer. Thus, when one includes the circuits for the error signal and adjustment of the shift controls, the result is an exact LZA.The principal concepts for calculating this error signal are also included in this paper.

The remainder of this paper describes the methods for detecting leading digits, encoding a count of the leading digits, detecting a zero-value result, and correcting the error in the inexact LZAs. We describe generalized leading digit detection and detail optimizations possible for restricted cases.

2. General leading digit detection and anticipation

For an arbitrary binary number, k-bits of leading zero can be represented as the string of digits $0^k 1x^*$, where the superscript represents k instances of the digit 0, x is either zero or one, and * indicates zero or more instances of the digit x. Likewise, k-bits of leading one can be represented as $1^k 0x^*$. Leading zero detection thus involves a determination of the position of the first non-zero digit, or equivalently the first transition from a zero digit i to a one digit i+1. Leading one detection involves the location of the first transition from a one digit to an adjacent zero digit.

In most of the literature, the term *leading zeros* refers to a starting string of zeros prior to the first one, while *leading ones* refers to a starting string of ones prior to the first zero. However, there may be some confusion since several papers also use the term *leading one predictor* for determining the first one after a starting string of zeros. Therefore, in this paper, we avoid use of that term.

Leading zeros occur when the result of a subtraction is positive, and leading ones occur when the result is negative. LZAs make use of the propagate (T), generate (G), and kill (Z) functions for each bit position of the adder inputs A and B after swapping, alignment, and inversion have taken place. These functions are defined as:

$$T = A \oplus B,\ G = AB,\ Z = \bar{A}\bar{B}$$

Leading zeros occur when the starting sequence has the pattern T*GZ*. If there are n bit positions before the first mismatch, then the sum will have either (n-1) or n leading zeros. Similarly, the number of leading ones can be found when the starting sequence is T*ZG*.

For addition or subtraction with 2's complement signed numbers, leading zeros may also occur with a starting sequence of Z*, and leading ones may occur with a starting sequence of G*.

A starting sequence of Z* may also occur in effective addition of floating point denormalized operands. If an LZA is to be used for both effective addition and subtraction, then it would be useful to prefix the sequence with a T for subtraction and a Z for addition. Also, we can append a low order Z for an input carry of zero, and a low order G for an input carry of one, as sometimes needed for subtraction.

2.1. Detection of first leading digit -- general case

Kershaw, et al [2][3] recognized that each digit can be evaluated to determine if this digit can possibly be the first leading digit by examination of this digit and its two neighbors, one to the left and one to the right. Knowles [4] formalized the solution by providing a truth table for setting an indicator f_i. If the bits are numbered such that bit 0 is the most significant, then, the indicator is equal to one when:

$$f_0 = \bar{T}_0 T_1 \qquad (1)$$
$$f_i = T_{i-1}(G_i\bar{Z}_{i+1} \vee Z_i\bar{G}_{i+1}) \vee \bar{T}_{i-1}(Z_i\bar{Z}_{i+1} \vee G_i\bar{G}_{i+1}),\ i > 0$$

If the indicator is set in position i and no other digit of greater significance has its indicator set, then the leading digit is either i or i+1.

Essentially the same result appears in a recent paper by Bruguera and Lang [14].

2.2. Separate detection of leading zeros and ones

If the detection of leading zeros and ones are done separately, then the indicators only need to examine bits i and i+1. For the leading zeros case, the indicator, f_i^{zeros} is equal to one, when

$$f_i^{zeros} = T_i \oplus \bar{Z}_{i+1},\ i \geq 0 \qquad (2)$$

If the indicator is set in position i and no other digit of greater significance has its indicator set, then the leading digit is either i or i+1.

Likewise for the leading ones case, the indicator, f_i^{ones} is equal to one, when

$$f_i^{ones} = T_i \oplus \bar{G}_{i+1},\ i \geq 0 \qquad (3)$$

If the indicator is set in position i and no other digit of greater significance has its indicator set, then the leading

digit is either i or i+1.

The indicators defined in equ. (2) and (3) are used in the LZA by Schmookler and Mikan [1]. In that design, the indicators are ORed from the left to create two monotonic strings of zeros followed by ones. The two strings are then ANDed together bit-wise to create a single monotonic string whose first one predicts the bit position of the most significant bit.

2.3. Detection of first leading digit -- restricted cases

The indicators defined in equ. (2) and (3) can be simplified further when the detection is restricted to only leading zeros or leading ones. For example, when the circuit for detection of leading zeros does not need to consider cases where leading ones might result, then the leading zero indicator can be simplified to

$$f_i^{zeros} = \bar{T}_i \, \bar{Z}_{i+1} \ , \ i \geq 0 \qquad (4)$$

as shown by Suzuki et al [7]. In that paper, a comparison of the operands is performed to ensure that only the smaller operand is complemented during subtraction. Other designs where this could be applied would be where separate adders are provided for use when the exponents are equal. One adder calculates A-B and the other calculates B-A, and the result from the adder producing a carry out is selected. Each adder then needs only a leading zero detector using indicators of the form shown above.

An LZA based on equ. (4) is used in another recent paper by Bruguera and Lang [15].

Another variation appears in a patent by Britton et al [6]. In this design, separate leading zero and leading one detectors are used, and the adder output carry selects between them. The leading zero detector uses indicators defined as in equ. (4), and the leading one detector uses indicators as defined in equ. (5) shown below:

$$f_i^{ones} = \bar{T}_i \, \bar{G}_{i+1} \ , \ i \geq 0 \qquad (5)$$

Further simplification results for a case which is even more restricted, as described in [8] and [9]. Some floating point adders provide separate dataflow paths for "far" and "near" cases. The far path is used for either effective addition or for subtraction of operands whose exponents differ by more than one. No LZA is needed for the far path. In the near path, separate LZAs are used for subtraction of operands whose exponents are equal and for subtraction of operands whose exponents differ by one. This allows the detection of the number of leading zeros to start in parallel with swapping, aligning and inverting the operands. When the exponents differ by one, the presumed smaller operand is shifted right one place and then inverted. Since the operands must be normalized, the function in the first bit must be G, and therefore the number of leading zeros is determined by the number of following bit positions that are Zs. Therefore, the leading zero indicator in each following bit position is $f_i^{zeros} = \bar{Z}_{i+1}$.

2.4. An exact LZA

A conceptually simple exact LZA described in [13] is integrated with the adder. To handle both positive and negative results, two separate adders are used, one assuming the first operand is larger, the other assuming the second operand is larger. The output carry from the first adder is used to select the result which is positive. Each adder includes its own LZA which is also selected.

Since each adder may assume that its result is positive, its LZA only needs to consider leading zeros. The adder design uses carry select, so that for each group of bits, two sets of conditional sums are generated, one set assuming input carry of zero, the other assuming input carry of one. The internal carries then select the appropriate sums as they are evaluated. With each group of conditional sums, a conditional count of leading zeros is determined for the group. These conditional counts are then also selected by the internal carries. This description is a simplification of the actual design, which must also take into account the hierarchy of the adder and also generate the high order bits of the leading zero count from larger groups of bits.

2.5. Comparing cost and delay

In this section, the LZA described by equ. (1) is referred to as Kershaw, (the earliest reference), the LZA described by equ. (2) and (3) is referred to as Schmookler, and the LZA described by equ. (4) and (5) is referred to as Britton.

Only Kershaw and Schmookler cover both leading zeros and ones, without using any carry signals from the adder. From the equations, it is apparent that Kershaw would have one or two more gate levels of delay for just the indicators, and a few more total gates as well. However, Schmookler then requires separate ORing of signals from the left for leading zeros and for leading ones, so the costs are more comparable. Then, Schmookler also requires the resulting strings to be ANDed, so the total delays may also be about the same.

Now comparing Kershaw with Britton, although the

indicator circuits are much smaller and faster with Britton, both the ORing and encoding of the shift signals must be duplicated for the two cases, before the adder carry signal is available for selection. Therefore, the cost of Britton may actually be slightly greater, and its delay is dependent on the speed of the adder. In the actual circuit implementations that are shown, Britton shows several enhancements for reducing the delay. Both use precharged chains of NFET pass gates for propagating the leading zero signal from the high order bits to the lower order bits, to accomplish the ORing. However, Britton uses a regenerative feedback circuit in each bit position to help pull the chain low. Britton also illustrates how a wide word can be broken up into smaller chains which operate in parallel to provide some lookahead.

When one only needs to consider leading zeros, it is apparent that the LZA used by Suzuki would provide lower cost and less delay than Kershaw.

3. Encoding count of leading digits

There are two basically distinct methods of obtaining an encoded count of the leading digits. One method includes the creation of a monotonic string of zeros followed by ones. The other method uses a hierarchical tree structure.

3.1. Leading digit counting through monotonic string production

The restriction that no other digit of greater significance with an active indicator imposes a priority encoding function on the anticipator. The priority encoding involves the generation of the ORing of all indicator bits of greater significance. The Boolean inverse of this value is ANDed with the indicator to signify that the position i contains the first leading digit:

$$F_i = \sum_{j=0}^{i} f_j \tag{6}$$

$$L_i = \overline{F}_{i-1} \wedge f_i \tag{7}$$

This ORing function creates a monotonic string in which the digit i represents the ORing of all less significant indicators. Once this string is created, the indicator f_i is ANDed with the inverse of the monotonic string in position i-1 to determine the position which is within one digit of the most significant digit of the result.

The creation of the monotonic string can be accomplished through the use of Manchester carry techniques[2] [3] or through hierarchical or look-ahead techniques [4].

The monotonic string method is used in several LZAs. The Power and Power2 processors employ the well-known LZA designed by Hokenek and Montoye [5]. Five separate monotonic strings are generated, including strings for leading ones, leading zeros and the case where all bit positions are Ts. These strings are ones followed by zeros, where the first zero indicates the location of the most significant bit position. Therefore, they are bit-wise ORed together to obtain a single string. The Power3 processor, and also several PowerPC processors such as some which are used in the Power Macintosh, use the LZA by Schmookler [1], which creates two monotonic strings as we previously described in section 2.2. Thus, the creation of monotonic strings in both of these designs is essential to combining the several strings into a single string.

In Kersaw [2][3], generating the count from a monotonic string is dictated by the use of precharged chains. One small circuit integrates the adder and LZA functions together. It uses a boot-strapped Manchester carry chain for the adder, which propagates the carries from right to left, and it uses a similar precharged chain to propagate the F_i signal from left to right under control of the local propagate signals to generate the monotonic F_i string and the 1-of-32 coded string L. The carry signals and the L_i are also used to create an error signal at each position, e_i. The OR of these e_i signals indicates that a 1-bit correction is needed in both the shifter and the exponent. The creation of an error signal in this way also required generation of the monotonic string.

In the T9000 described by Knowles [4], the LZA is logically similar to that of Kershaw, but with more standard lookahead techniques similar to the carry skip techniques used in their adder. The use of ORing to create the monotonic strings is due to its simplicity.

In order to get the leading zero count, either simple AND-OR functions of the F_i signals or simple ORing of particular L_i signals from equ. (6) and (7) permit easy and fast encoding of the count. For example, for an eight-bit sum, the shift amount which is determined by the binary encoding of the location of the leading significant digit can be formed by:

$$\begin{aligned} SA_0 &= \overline{F}_3 F_7 \\ &= L_4 \vee L_5 \vee L_6 \vee L_7 \\ SA_1 &= \overline{F}_1 F_3 \vee \overline{F}_5 F_7 \\ &= L_2 \vee L_3 \vee L_6 \vee L_7 \\ SA_2 &= \overline{F}_0 F_1 \vee \overline{F}_2 F_3 \vee \overline{F}_4 F_5 \vee \overline{F}_6 F_7 \end{aligned} \tag{8}$$

$$= L_1 \vee L_3 \vee L_5 \vee L_7$$

3.2. Leading digit counting with tree structure

The other well-known method for LZC design consists of a tree structure. For example, the string of n inputs may first be partitioned into n/2 pairs of adjacent bits. For each pair, a 2-bit leading zero count is generated, and the high order bit also indicates when both bits are zeros. At the next level, adjacent pairs are combined, a mux circuit selects the count from one of the pairs, and a new high order bit is appended to the count which also indicates that both pairs are all zeros. This scheme is continued for $\log_2(n)$ levels. Some speedup can be obtained by detecting larger groups of all zeros and using larger multi-way muxes. This type of binary tree structures is described more fully for a leading zero detector by Oklobdzija [16][17] and for the LZA by Suzuki [7]. The similarity of the two implementations is shown in a short correspondence by Oklobdzija [18].

The first method that was described using monotonic strings can be significantly faster than the hierarchical tree structure if one uses a circuit topology which permits fast wide ORs (or ANDs with negative polarity signals). Manchester carry circuits at one time provided such benefits, but long chains are less attractive with low-voltage technology. More conventional dynamic circuits, however, are well suited to use of wide ORs.

On the other hand, with dynamic circuits, the tree structure method can also be sped up by using 4-bit or even 8-bit groups to reduce the hierarchy by a factor of two or three. The wide ORs can also be used in a hybrid structure to reduce the number of levels.

4. Early zero result detection

Early detection of a zero result is most important for fixed point arithmetic units, which must set condition bits indicating if a result is greater than, less than, or equal to zero. The detection must be done in parallel with the addition or subtraction, to enable fast conditional branching. A solution by Weinberger [19] describes a rather complicated expression which uses the T, G and Z functions for each bit position, but which is faster than the adder itself. A simpler solution is presented in Vassiliades [20][21] which is equivalent to the solution described here for leading zero detection in which all f_i^{zeros} indicator bits are zeros. Thus, a simple ORing of these bits, or equivalently, the value of F_{n-1}^{zeros} would detect a non-zero result for an n-bit adder. We noted earlier that for addition, Z_n=1, so the general T^* G Z^* sequence can produce a zero result with 2's complement signed numbers as used in fixed point arithmetic. For subtraction, however, since G_n=1, the only sequence that can produce a zero result is T^n, which corresponds to both inputs being identical prior to inverting one of them. Both of these cases are handled properly by the use of F_{n-1}^{zeros}. It should also be pointed out that although the Vassiliades method also lends itself to leading zero detection, the Weinberger method does not. Nevertheless, it was the only known solution for many years.

For floating point, if a full LZA is used, then F_{n-1}^{zeros} provides an attractive way to determine a zero result. Otherwise, since the T^* G Z^* sequence cannot produce leading zeros for effective addition, a simpler circuit may be chosen. For effective addition, a zero result can only occur when both operands are zeros, therefore, ORing the $\bar{Z}_i$ signals would detect a non-zero result. For effective subtraction, both operands must be identical, so ORing the $\bar{T}_i$ signals would detect a non-zero result.

5. Handling the error in anticipators

LZAs as described are inexact; they may have up to one bit of error in the count. This can be detected at the end of the normalization shift, and if there remains a leading zero, the result can be shifted one more bit position in a 2-to-1 multiplexor. A slightly faster method is to do this extra shift in the last stage of the normalization shifter. The shifter usually contains one or more stages of coarse shifting, and the last stage does the fine shifting. For example, if the coarse shifters do all shifts which are multiples of four bits, then the fine shifter would normally only shift from zero to three bit positions. However, to allow for the LZA error, the fine shifter would be modified to shift up to four bit positions. The high order four bits from the coarse shifter would be examined to determine the correct shift amount. A fast circuit for doing this uses the predicted control signals. For example, if the LZA predicts a fine shift of two, then it would select a shift of two if bit 2 is a one, otherwise it selects a shift of three.

A similar method is to avoid the calculation of the least significant digits of the leading digit count. This eliminates a large amount of complexity in the anticipator. The circuit described above for selecting the fine shift controls would be replaced by a four-bit LZD. Since that essentially has the delay of a 4-way NAND, it is not much slower. However, one now needs only an LZA which computes the number of leading digits modulo-4. The indicators f_i, F_i, and L_i only need to be generated for each block of four bits, resulting in savings in both circuits and delay. This method is used in the early IBM RS/6000 processors described by Montoye and Hokenek [22] [5].

The error can also be detected at each bit position [2][3] through an examination of the carry-in to that position. For a leading position i the error indicator is:

$$e_i = L_i T_{i+1}(C_{i+1} \oplus (T_i \vee \overline{(A_i \oplus T_{i-1})})) \quad (9)$$

where A_i is the addend digit and C_{i+1} is the carry-in to the (i+1)th digit.

The global error correction signal, e, indicates when an additional leading digit must be removed, where,

$$e = \sum_i e_i \ . \quad (10)$$

Quach [23] has shown that an equivalent error indicator can be formed by

$$e_i = L_i(T_{i-1} \oplus A_i \oplus \overline{C}_i) \quad (11)$$

where, C_i is the carry-in to the ith digit.

6. Summary

Use of leading zero anticipators or detectors is an established method of reducing the delay of floating-point addition. We have presented several algorithms and implementations which have proven to be fast and efficient, and we have shown how other choices made in the design of the adder, including circuit technology considerations, can guide in the selection of the best method for a particular floating-point processor. We have included both algorithms which assume that the result of an effective subtraction must be positive and those which cover the general case of floating-point addition. Because they are closely related to leading zero anticipation, we have described alternative design for leading zero detectors and count leading zero implementations. We have described alternative methods with dealing with the one-bit position error inherent in leading zero anticipators.

References

[1] M. Schmookler and D. Mikan, "Two-state Leading Zero/One Anticipator (LZA), US Patent #5493520, Feb. 1996.

[2] R. Kershaw, L. Bays, R. Freyman, J. Klinikowski, C. Miller, K. Mondal, H. Moscovitz, W. Stocker, L. Tran, "A Programmable Digital Signal Processor with 32-bit Floating-Point Arithmetic", IEEE Solid State Circuits Conference, Digest of Papers, 1985, pp. 92-93.

[3] W. Hays, R. Kershaw, L. Bays, J. Bodie, E. Fields, R. Freyman, C. Garen, J. Hartung, J. Klinikowski, C. Miller, K. Mondal, H. Moscovitz, Y. Rotblum, W. Stocker, L. Tran, "A 32-bit VLSI Digital Signal Processor", IEEE Journal of Solid State Circuits, October 1985, pp. 998-1004.

[4] S. Knowles, "Arithmetic Processor Design for the T9000 Transputer", SPIE, v. 1566, 1991, pp.230-243.

[5] E. Hokenek and R. Montoye, "Leading-Zero Anticipator (LZA) in the IBM RISC System/6000 Floating Point Execution Unit", IBM Journal of Research and Development, Jan. 1990, pp. 71-77.

[6] S. Britton, R. Allmon, S. Samudrala, "Leading One/Zero Bit Detector for Floating Point Operation", US Patent #5317527, May 1994.

[7] H. Suzuki, H. Morinaka, H. Makino, Y. Nakase, K. Mashiko, T. Sumi, "Leading-zero Anticipatory Logic for High-speed Floating Point Addition", IEEE Journal of Solid State Circuits, August 1996, pp. 1157-1164.

[8] H.P. Sit, et. al., "Prenormalization for a floating-point adder", US Patent #5010508, April 1991.

[9] S. Oberman and M. Roberts, "Leading one prediction unit fo normalizing close path subtraction results within a floating point arithmetic unit", US Patent #6085208, July 2000.

[10] R. Maher III, "Circuit for Simultaneous Arithmetic Calculation and Normalization Estimation", US Patent #5040138, August 1991.

[11] K. Ng, "Exact Leading Zero Predictor for a Floating Point Adder", US Patent #5204825, February 1993.

[12] G. Inoue, "Leading one anticipator and floating point addition/subtraction apparatus," US Patent #5343413, August 1994.

[13] G. Gerwig and M. Kroener, "Floating Point Unit in standard cell design with 116 bit wide dataflow", IEEE Symposium on Computer Arithmetic. 1999. pp 266-273.

[14] J. Bruguera and T. Lang "Leading-One Prediction with Concurrent Position Correction" IEEE Transactions on Computers, v. 48, No. 10, October 1999, pp. 298-305.

[15] J. Bruguera and T. Lang "Leading-One Prediction Scheme for Latency Improvement in Single Datapath Floating-Point Adders" Proceedings International Conference on Computer Design, October 1998, pp. 298-305.

[16] V. Oklobdzija, "An Implementation Algorithm and Design of a Novel Leading Zero Detector Circuit", 26th IEEE Asilomar Conference on Signals, Systems, and Computers, 1992, pp. 391-395.

[17] V. Oklobdzija, "An Algorithmic and Novel Design of a Leading Zero Detector Circuit: Comparison with Logic Synthesis", IEEE Transactions on VLSI Systems, v. 2, no. 1, 1993, pp. 124-128.

[18] V. Oklobdzija, "Comments on Leading-zero Anticipatory Logic for High-speed Floating Point Addition", IEEE Journal of Solid State Circuits, February 1997, pp. 292-293.

[19] A. Weinberger, "High-speed Zero Sum Detection", 4th Symposium on Computer Arithmetic, 1975.

[20] S. Vassiliadis and M. Putrino, "Condition Code Prediction for Fixed-point Arithmetic Units", International Journal of Electronics, June 1989, pp. 887-890.

[21] S. Vassiliadis, M. Putrino, A. Huffman, B. Feal, G. Pechanek., "Apparatus and Method for Prediction of Zero Arithmetic/Logic Results", US Patent #4947359, August 1990.

[22] E. Hokenek and R. Montoye, "Second Generation RISC Floating Point with Multiply-Add Fused", IEEE Journal of Solid State Circuits, v. 25, no. 5, October 1990, pp. 1207-1212.

[23] N. Quach and M. Flynn, "Leading One Prediction -- Implementation, Generalization, and Application", Technical Report CSL-TR-91-463, Stanford University, March 1991.