FERMILAB-Pub-92/312

A Full Custom, High Speed Floating Point Adder

J.R. Hoff and G.W. Foster

Fermi National Accelerator Laboratory
P.O. Box 500, Batavia, Illinois 60510

November 1992

Submitted to *Transactions on 1992 Nuclear Science Symposium*

A Full Custom, High Speed Floating Point Adder

J.R. Hoff, G.W. Foster

Fermi National Accelerator Laboratory[1]

Abstract

This document describes the concept, design and implementation of a high-speed floating point adder for use by the Solenoidal Detector Collaboration (SDC) at the Superconducting Super Collider (SSC). The adder uses a unique floating point format, described herein, and is implemented using Orbit Semiconductor's 1.2um n-well process. Simulations indicate that the device will operate at 63 MHz.

I. INTRODUCTION

One of the primary advantages of the proposed SDC Calorimeter is the digitization of incoming signals at each beam-crossing. This allows for a wide variety of data manipulations in the digital domain rather than error-prone attempts of the same in the analog domain. Furthermore, this manipulation can be performed in custom designed digital electronics at high data rates

One of the aforementioned manipulations is to sum the digitized outputs of Twenty-five Thousand photomultiplier tubes from the SDC Calorimeter. It is for this and for several other similar potential applications that the AdderChip was developed.

The AdderChip is a full custom IC capable of adding together two floating point numbers each with 8 bit mantissa and 4 bit exponents. Both the Exponents and the Mantissa can be assumed positive in this particular design. The AdderChips would be arranged in a multi-tiered fashion with sums of previous AdderChips feeding the inputs of other AdderChips producing what is called the AdderTree with the ultimate output being the sum of many inputs.

What follows will be a short description of the Floating Point format used followed by the design description. Finally, an Appendix is included which contains the schematic and full custom designs of all of the circuits and subcircuits used in the Chip.

II. THE FLOATING POINT FORMAT

The AdderChip's floating point format is roughly based on the standard scientific format. That is to say,

$$N = (\text{Mantissa}) \times 2^{(\text{Exponent})} \qquad (1)$$

Thus, if the Mantissa were 10100000_2 and the Exponent were 0011_2, then N would be equal to 10100000000_2 or 1280.

Also, it is assumed that the Mantissa has been shifted to the left or to the right and the Exponent increased or decreased commensurately in order to place a 1 in the most significant bit of the Mantissa.

This format works well for a large number of situations, but improvements can be made. First of all, if it is known that the most significant bit of the Mantissa must be a 1, then it is possible to assume the existence of that bit and gain an extra bit of accuracy. In other words if the real mantissa were 110010101_2, it could be stored as 10010101_2 under the assumption that 100000000_2 must be added to 10010101_2 in order for the results to be correct. This is known as a Hidden Bit.

Secondly, and more critically, the requirement that there be a leading 1 (Hidden or present) eliminates the possibility of a true 0, and, since the smallest possible Exponent in this design is 0 (all Exponents are assumed non-negative), this means that the smallest possible number would be 128 (or 256 if Hidden Bits were used).

The solution is to use a Conditional Hidden Bit format. A true Zero requires both a Zero mantissa and a Zero exponent, so, in the Conditional Hidden Bit format, Hidden Bits are always assumed to be present *unless the Exponent is Zero*. If the Exponent is Zero, then the Mantissa contains *no* Hidden Bits.

$$N = \text{Mantissa (if Exponent} = 0) \qquad (2)$$

$$N = ((\text{Mantissa}+256) \times 2^{(\text{Exponent-1})}) \text{ (if Exponent} > 0), \qquad (3)$$

Exponent	Mantissa	Value
0	0	0
0	1	1
0	255	255
1	0	256
1	255	511
4	1	1028
11	255	$FF800_{16}$

This format yields errors ranging from as little as 1 part is 511 to as great as 1 part in 255 and is exact for counts less than 512.

1. Work supported by U.S. Department of Energy under contract No. DE-AC02-76CH03000.

By way of comparison, standard scientific format yields errors of 1 part in 255 and is incapable of representing counts less than 128. Most importantly, it yields 24 bits of Dynamic Range with 8 to 9 bits of accuracy

III. THE DESIGN

The AdderChip was required to add two 12 bit floating point numbers using a two-staged pipeline operating at approximately 63MHz. The first task was to determine what functions went into each pipeline stage. In the addition of two floating point numbers (see Figure 1), the exponents must be compared to each other, the Mantissa must be shuffled relative to one another as dictated by the comparison of Exponents, the Mantissa must then be added, and if Mantissa carries out, then the larger of the two Exponents must be increased by one and the Mantissa Sum must be left shifted. These steps are, for simplicity sake, referred to as Exponent Comparison, the Mantissa Shuffle, the Mantissa Addition, and the Renormalization.

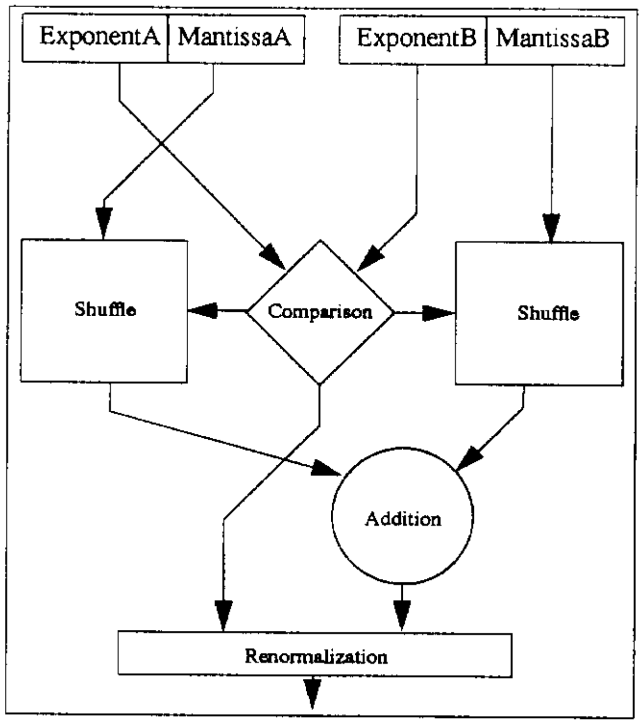

Fig. 1 The AdderChip Flow Diagram

The use of Conditional Hidden Bits adds complexity to the design, but it does not change the basic data flow presented above.

At 63MHz, each Tick has approximately 16ns to perform its task. Since Exponent Comparison, Mantissa Shuffle, Mantissa Addition and Renormalization must occur in that order, the question of how to subdivide the Addition Flow into the First and Second Ticks reduces to how much of the Addition Flow can be completed in the first 16ns. Whatever is left over must be completed in the Second Tick. Therefore, it was rather simple to determine that the Exponent Comparison and Mantissa Shuffle would be accomplished in the FirstTick and Mantissa Addition and Renormalization would be accomplished in the SecondTick.

IV. THE FIRST TICK

A. Exponent Comparison

In this most recent AdderChip design, a means of parallel Exponent Comparison and Mantissa Shuffle was attempted. In this scheme, the initial stages of the Shuffle are begun as soon as partial information on the Exponent Comparison becomes available.

It is well known that in the Binary system, the magnitude of a given number is represented as follows:

$$N_2 = B_N 2^N + B_{N-1} 2^{N-1} + \ldots + B_3 2^3 + B_2 2^2 + B_1 2^1 + B_0 2^0 \quad (4)$$

where each B_X is either a Zero or a One. For the most recent version of the AdderChip, a new numbering system has been invented which has been called, for lack of a better term, Trinary. Magnitudes in this system are expressed in a similar fashion

$$I_{Trin} = B_N 2^N + B_{N-1} 2^{N-1} + \ldots + B_3 2^3 + B_2 2^2 + B_1 2^1 + B_0 2 \quad (5)$$

However, in the Trinary system,

$$B_X \subset \begin{bmatrix} 0 & 1 & -1 \end{bmatrix} \quad (6)$$

This permits the following representations, in Trinary,

$$1101 - 0111 = 10(-1)0 = 6_{10} \quad (7)$$

To apply this to a four stage Mantissa Shuffle circuit,

- a) if $B_X = 1$, then right shift the mantissa of the smaller exponent
- b) if $B_x = 0$, then do not shift either mantissa
- c) if $B_X = -1$, then left shift the mantissa of the smaller exponent.

The significance of the Trinary Numbering System is that if it is known which Exponent is larger, then it is possible to do a all-bits parallel comparison of the two Exponents. No carrying or borrowing is necessary for subtractions, and thus much time is saved.

Finally, if Exponent A is equal to Exponent B, then neither Mantissa should be shuffled. This is, of course, obvious, but important nonetheless.

As was stated earlier, there still is no *a'priori* knowledge about the relative magnitudes of the two Exponents. However, neither Mantissa gets shuffled *either* when Exponent A is Larger than Exponent B and Bit N of Exponent A equals Bit N of Exponent B *or* when Exponent A is equal to Exponent B. So, from a Logic Design standpoint, the two states are equivalent. Therefore, a simple scan from the Most Significant Bits of the two Exponents to the Least Significant bits of the two Exponents will reveal the larger of the two numbers at the first Bit Pair that do not match. Whichever Exponent has a One in the same Bit location as the Most Significant Zero of the other Exponent is the larger number. All Bits of greater significance than this Bit simply leave both Mantissas unshuffled up to that point, All Bits of less significance than this Bit are made aware of the relative magnitudes of the two numbers.

In simulation, the worst case occurred when one Exponent was equal to 1000_2 and the other Exponent was equal to 0111_2. This was due to the fact that the "who's greater" signals needed to propagate from the Most Significant Bit to the Least Significant Bit, and change every result from "I'm Bigger" and "Don't Shift" to "He's Bigger" and "Shift Up".

B. The Mantissa Shuffle

As indicated in the previous subsection, the Mantissa Shuffle is a four sectioned design in which each of the four sections are virtually identical. In effect, each section is a three-to-one multiplexor which accepts a binary number as its input and shifts it up, down or not at all depending on the output of the appropriate Exponent Comparison Bit. The fact that ShiftUp, ShiftDown, and DontShift are designed to be mutually exclusive in all cases makes the Multiplexor easier to design. All that is really needed is a two level and-or scheme.

V. THE SECOND TICK

The SecondTick accepts as its inputs the Shuffled Mantissas and the Larger Exponent. It performs the Mantissa Addition and then the Renormalization. Its output is a twelve bit floating point number in the same Conditional Hidden Bit format as the two inputs.

A. The Mantissa Addition

The Mantissa Addition is accomplished though a nine bit Binary Carry Lookahead Adder (BCLA).

It is it is well known that

$$A_i \oplus B_i \oplus Carry_{i-1} = Sum_i \tag{8}$$

Furthermore,

$$(A_i \wedge B_i) \vee (A_i \wedge Carry_{i-1}) \wedge (B_i \wedge Carry_{i-1}) = Carry_i \tag{9}$$

In other words, there would be a carry into the next more significant bit addition if either bit A and bit B were 1 or either bit A were a 1 and there was a carry into this bit addition or B were 1 and there was a carry into this addition. In the design of adders, these two possibilities under which there is a carry into the next bit have special names. When both bit A and bit B are 1, they are said to generate there own carry. When either A or B are 1 and the carry input to this bit is a 1,A or B are said to propagate their carry input into the next bit.

The adder complexity can be reduced if two functions are defined based on the aforementioned two possibilities. The Carry Generate function is defined to be

$$g_i = A_i \wedge B_i \tag{10}$$

The Carry Propagate function is defined to be

$$p_i = A_i \oplus B_i \tag{11}$$

Thus, the Sum equation is redefined to be

$$p_i \oplus Carry_{i-1} = Sum_i \tag{12}$$

and the Carry equation is redefined to be

$$g_i \vee (p_i \wedge Carry_{i-1}) = Carry_i \tag{13}$$

The goal of Binary Carry Lookahead is to provide a regular (i.e.repeatable and easily VLSI designed) means of generating sums in a fashion such that the total propagation delay is sub-linear. Specifically, the total propagation delay increases logarithmically.

The reduction in delay is accomplished by splitting apart the carry propagation circuitry so that as much as possible occurs in parallel. Let's assume for the moment that there is a 4-bit adder that is to be used exclusively as an adder and that no preceding adder stage is feeding its CarryIn. In other words, C_{in} is always 0. Then,

$$Cout_0 = g_0 \tag{14}$$

$$Cout_1 = g_1 \vee p_1 g_0 \tag{15}$$

$$Cout_2 = g_2 \vee p_2 (g_1 \vee p_1 g_0) \tag{16}$$

$$Cout_3 = g_3 \vee p_3 (g_2 \vee p_2 (g_1 \vee p_1 g_0)) \tag{17}$$

A pattern begins to emerge here, and it is exploited by the BCLA. For example,

$$Cout_2 = g_2 \vee p_2 \overbrace{(g_1 \vee p_1 g_0)}^{G_{01}} \tag{18}$$

and,

$$Cout_3 = \overbrace{(g_3 \vee p_3 g_2)}^{G_{23}} \vee \underbrace{(p_3 p_2)}_{P_{23}} \overbrace{(g_1 \vee p_1 g_0)}^{G_{01}} \tag{19}$$

Thus, if we define two equations,

$$G_{xy} = g_y \vee p_y g_x \tag{20}$$

and,

$$P_{xy} = p_x \wedge p_y \tag{21}$$

Then the CarryOut equations become

$$Cout_0 = g_0 \tag{22}$$

$$Cout_1 = g_1 \vee p_1 (g_0) = G_{01} \tag{23}$$

$$Cout_2 = g_2 \vee p_2 G_{01} = G_{012} \tag{24}$$

$$Cout_3 = G_{23} \vee P_{23} G_{01} = G_{0123} \tag{25}$$

and the Sum equations become

$$Sum_i = p_i \oplus G_{((i-1)(i-2)\ldots(0))} \tag{26}$$

Brent and Kung define an operator, "o", which replaces the G and P equations above

$$.\,(g, p) \circ (g', p') = (g \vee (p \wedge g'), p \wedge p') \tag{27}$$

This is no new information, it is simply presented so that the reader can better understand the next figure. It is this format that realizes the goals of the BCLA. First, it is regular. Only two subcircuits are needed - gen to generate p_i and g_i from A_i and B_i, the incoming numbers to be added, and O to generate Ps and Gs from p and g. Second, it has a logarithmic time increase with the number of gates. This is easily seen from the figure because Cout0 passes through 0 gates (after p and g are generated) whereas Cout1 passes through 1 gate and Cout2 and Cout3 pass through 2 apiece. The following figures will hopefully clarify any remaining questions about BCLA theory.

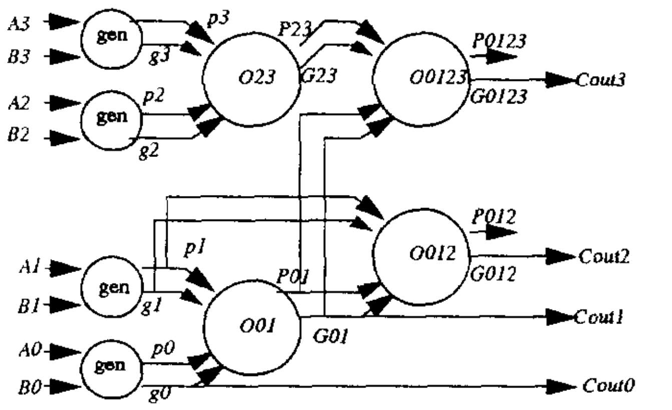

Fig. 2 .Binary Carry Lookahead Generation

The various subcircuits were implemented and replicated. The resulting Binary Carry Lookahead Adder adds the Nine Bit Mantissas in approximately 6-8ns.

B. Renormalization

Essentially, if there is a carry out from the Mantissa Addition, then the resulting Mantissa needs to be right shifted one place and the Exponent needs to be increased by one. The shifting of the Mantissa is quite simple. Internally, the adder is nine bits wide. If there is a Carry Out, the Upper 8 Bits are the proper output. If there is no Carry Out, the Lower 8 Bits are the proper output. In the case of the Exponent renormalization, however, waiting for the Carry Out to determine whether or not to Add a 1 to the Exponent would take too much time. Thus, while the Mantissas are being added, a One is added to the Exponent, and both the original Exponent and the increased Exponent are held awaiting the Carry Out.

VI. SUPPORT CIRCUITRY

The AdderChip used straight forward Master-Slave flip-flops for the register stages. They have no preset or clear capabilities, so the output of the chip for the first two clock ticks might be meaningless gibberish.

The Clock circuitry was drastically altered for this design. In previous versions, large Clock Buffers distributed the signal around, but in this version, a Clock Tree distributes the signal in an evenly delayed fashion. Two extra pins were added to the design, ClockDelayed and ClockUndelayed, to show the delay from the beginning of the Tree to its furthest branches.

Finally, several registers were tapped off of the signals between the FirstTick and the Second Tick to allow for greater visibility of the internal operations.

VII. FABRICATION AND RESULTS

The AdderChip was fabricated in a 1.2μ n-well process. Testing thus far indicates proper functionality in both fall-through (no clocks necessary, registers transparent) and clocked modes.

A problem has been discovered with certain sequences of numbers. This is due to the propagating nature of the Exponent Comparison. A new approach simular in Theory, but slightly different in approach has been proposed to alleviate this problem. A new version with this new Exponent Comparison is forthcoming.

Testing of the AdderChip continues both at Fermilab and at the Superconducting Super Collider in Texas.

VIII. ACKNOWLEDGEMENTS

I would like to acknowledge the work of M. Larwill and C. Needles who built the test station for the AdderChip and performed the tests of the AdderChip listed above. I would also like to acknowledge the work of J.Dorenbosch, P. Kibuule, and S. Karuturi of the SSC Laboratory who also tested the AdderChip.

IX. BIBLIOGRAPHY

1. Brent, R.P. and Kung, H. T., A Regular Layout For Parallel Adders, IEEE Transactions on Computers, VOL. C-31, No. 3, March, 1982