Fermi National Accelerator Laboratory

FERMILAB-TM-1779

A Full Custom, High Speed Floating Point Adder

J. Hoff and B. Foster

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

Operated by Universities Research Association Inc. under Contract No. DE-AC02-76CHO3000 with the United States Department of Energy

Disclaimer

A Full Custom, High Speed Floating Point Adder

Jim Hoff, Bill Foster
Fermi National Accelerator Laboratory

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,

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

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.

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

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

To demonstrate,

Table 1:

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. 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.

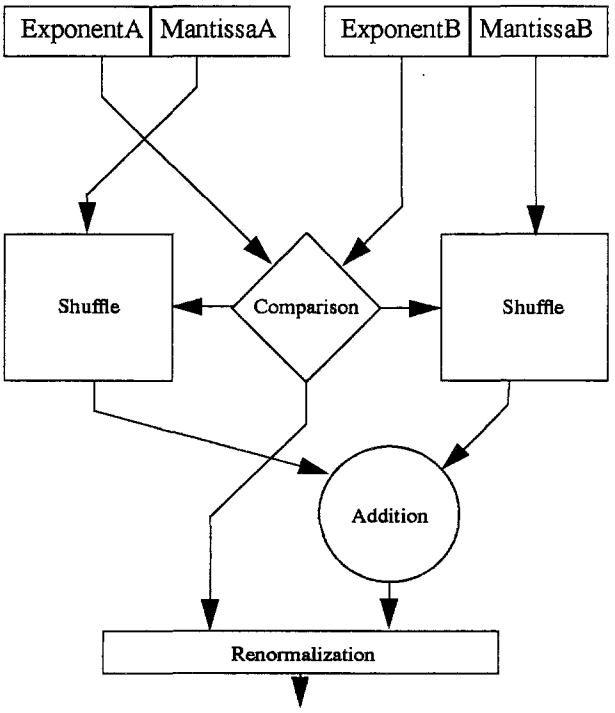


Fig. 1 The AdderChip Flow Diagram

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 the initial AdderChip designs, the Exponent Comparison circuitry did simultaneous subtractions of Exponent A from Exponent B and Exponent B from Exponent A. If the former was positive, then Mantissa B was shuffled by the Result of the former subtraction. If the latter was positive, then Mantissa A was shuffled by the Result of the latter subtraction. This style of Exponent Comparison was necessary because there was (and still is) no *a'priori* knowledge of which Exponent was larger. Unfortunately, this took up a great deal of space in the design and it resulted in a fixed delay of approximately 10ns between the presentation of the two numbers and the start of the appropriate Mantissa Shuffle.

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.

To explain how this was done, some background information is necessary. In the Binary number 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. This number system has an advantage as far as version 1 of the AdderChip was concerned because the results of the simultaneous Exponent subtractions, which were expressed in Binary, could be applied directly to a four-staged Mantissa Shuffle section. The first stage either shifted by 2^3 or it did not. The second stage shifted either shifted by 2^2 or it did not. The third stage either shifted by 2^1 or it did not. Finally the forth stage either shifted by 2^0 or it did not. Each decision to shift or not to shift was made based on the Nth bit of the subtraction. For example, if the 3rd bit was a One, then the Mantissa was shuffled by 2^3 in the 3rd Stage of the Mantissa Shuffle. If this bit was Zero, no shift was made by the third Mantissa Shuffle Stage.

For the most recent version of the AdderChip, a new numbering system has been invented and given the almost presumptuous name of 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,

$$B_X \subset [0 \ 1 \ -1] \quad (6)$$

Thus, in Binary,

$$1101 - 0111 = 0110 = 6_{10} \quad (7)$$

but in Trinary,

$$1101 - 0111 = 10(-1)0 = 6_{10} \quad (8)$$

To apply this to a four stage Mantissa Shuffle circuit,

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

The significance of the Trinary Numbering System is that if it is known which Exponent is larger, then it is possible to do a bit-by-bit comparison of the two Exponents. In other words, Exponent A is larger than Exponent B and Bit N of Exponent A is a One and Bit N of Exponent B is a Zero, then Mantissa B should be Shuffled to the Right by 2^N. If Exponent A is larger than Exponent B and Bit N of Exponent A is a Zero and Bit N of Exponent B is a One, then Mantissa B should be Shuffled to the Left by 2^N. If Exponent A is larger than Exponent B and Bit N of Exponent A is equal to Bit N of Exponent B, then the Mantissas shouldn't be shuffled at all.

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 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. For example, consider the following figure.

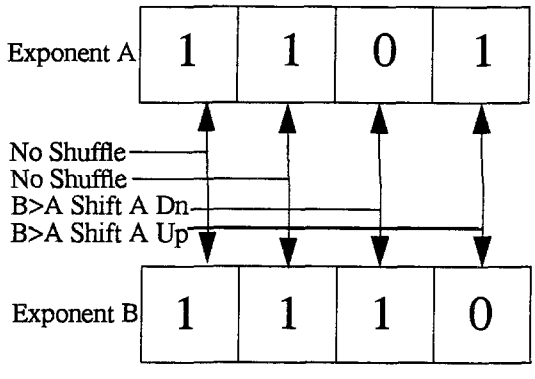


Fig. 2 Exponent Comparison

The Logic to perform this comparison is called MeHimLogic, and it uses the following inputs.

- a) HBI = 1 if He (the other Exponent) is Larger
- b) IBI = 1 if I'm Larger
- c) HI = His Bit being compared
- d) MI = My Bit being compared

The circuitry generates the following outputs

- a) HBO is set to One if I determine that He is Larger
- b) IBO is set to One if I determine that I am Larger
- c) ShiftUp is set to One if the Mantissa is to Shift Left
- d) ShiftDown is set to One if the Mantissa is to Shift Right
- e) DontShift is set to One if the Mantissa is not to Shift.

This circuitry is generated for each of the four bits and then replicated for both Exponents. The inputs are connected such that HI of Exponent A is connected to MI of Exponent B, etc. Also, ShiftUp, ShiftDown, and DontShift must be mutually exclusive. The following are the equations for the MeHimLogic.

$$\text{ShiftDown} = \overline{(\text{IBI})} \wedge (\text{HI}) \wedge \overline{(\text{MI})} \quad (9)$$

$$\text{ShiftUp} = (\text{HSI}) \wedge \overline{(\text{HI})} \wedge (\text{MI}) \quad (10)$$

$$\text{DontShift} = (\text{IBI}) \vee \overline{((\text{HI}) \oplus (\text{MI}))} \vee ((\text{MI}) \wedge \overline{(\text{HBI})}) \quad (11)$$

$$\text{HBO} = (\text{HBI}) \vee (\overline{(\text{IBI})} \wedge (\text{HI}) \wedge \overline{(\text{MI})}) \quad (12)$$

$$\text{IBO} = (\text{IBI}) \vee (\overline{(\text{HI})} \wedge (\text{MI}) \wedge \overline{(\text{HBI})}) \quad (13)$$

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 HBO and IBO signals were 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". It is possible that this particular situation will not be as radically different from the other possible scenarios in a real implementation. In any case, this Worst Case Situation still permits operation at the specified speed of 63MHz.

One final note is that the larger of the two exponents is passed on to the SecondTick. Either of the two HBO signals or either of the two IBO signals from the last Comparison Stage can be used to flip a two-to-one multiplexor and choose the appropriate Exponent.

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 as presented in Fig. 2.

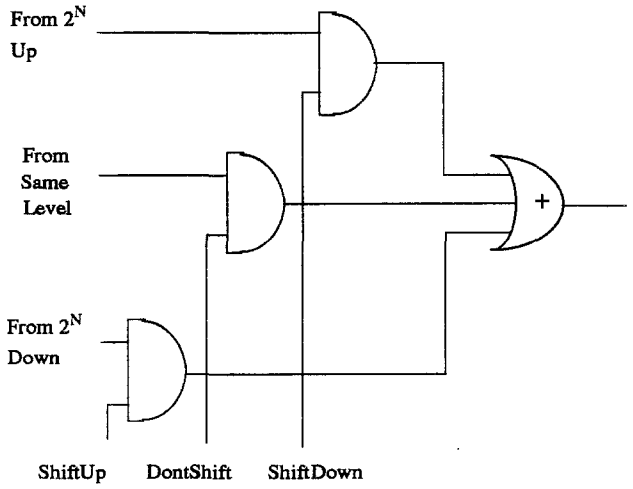


Fig. 3 One Mantissa Shuffle Stage

The unfortunate side effect of the redesign of the Exponent Comparison Circuitry is that now the Mantissa Shuffle Stage must be larger. Previously, only down shifts were necessary, so once a significant bit passed below the Least Significant Bit of the Shuffled output, it could be forgotten. However, in the present design, the possibility exists that the next stage might be a ShiftUp, and so, $2^{(N-1)}+2^{(N-2)}+...+2^0$ bits below the Least Significant Bit of the Shuffled Output must be maintained at each stage (where N is the current bit level). This, of course, adds a considerable amount of additional space in the design, but the increase in speed makes up for this.

C. Hidden Bits

As stated earlier, Hidden Bits add complexity to the design without effecting the data flow. To recall, the Conditional Hidden Bit format is as follows:

$$\text{if Exp} = 0 \; N = Mantissa \tag{14}$$

$$\text{if Exp} > 0 \; N = (Mantissa + 256)\, x2^{(Exponent-1)} \tag{15}$$

Thus, the first necessity of the Hidden Bits is to determine whether or not the Exponent is Zero. The next step is to add 256 to the Mantissa if the Exponent is not Zero or add nothing if the Exponent is Zero. This is actually quite simple. Externally, the AdderChip adds two numbers with 8 bit Mantissas. Internally, it adds two numbers with 9 bit Mantissas. If the Exponent equals Zero, then that ninth bit (in the most significant bit location) is set to a Zero. Otherwise, it is set to a one.

The final complexity comes from the Mantissa Shuffle. It is obvious that if both Exponents are 0, the no Hidden Bits are added and neither Mantissa is shifted. This presents no problem. Similarly, if both Exponents are greater than Zero, then both have Hidden Bits added and the smaller number is shifted by the difference between the Exponents. This presents no problem because

$$Exp_A - Exp_B = (Exp_A - 1) - (Exp_B - 1) \tag{16}$$

In other words, Equation 15 requires that the magnitude of a number with an Exponent greater than Zero be scaled by $2^{(\text{Exponent} - 1)}$. However, if both Exponents are greater than Zero, then the Mantissas will be shuffled relative to one another properly without any extra effort. However, if Exponent A is Zero and Exponent B is not, then there is a problem. The shifting of Mantissa A with respect to Mantissa B is based upon a direct comparison of their Exponents. However, Exponent B is not really the Exponent of the Number represented by the pair Exponent B/Mantissa B. Exponent B must be reduced by one for the Addition to be accurate. However, if Exponent B is reduced by one, then its original value must be restored before it is output from the AdderChip. This is gets quite complicated unless one realizes that if Exponent A is a Zero and Exponent B is not, then Mantissa A was shifted to the right by one place too many. Thus, a simple circuit can be designed to Left Shift either Mantissa in the event that one Exponent is Zero and the other is not. This circuit unfortunately must sit directly in the data flow path and it requires about 2-4ns to execute.

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 perhaps simplest to start with an example of binary addition, ignoring 2s compliment format. The numbers in this example are simple, positive binary number.

```
  11010  <-Carries
  00101    A
+ 01101    B
-------
  10010    Sum
```

Fig. 4 An Example of Binary Addition

It is easily seen from the example that

$$A_i \oplus B_i \oplus Carry_{i-1} = Sum_i \tag{17}$$

In other words, each bit of the Sum is a 1 if an odd number of its respective A, B and Carry bits are a 1. Furthermore

$$,(A_i \wedge B_i) \vee (A_i \wedge Carry_{i-1}) \wedge (B_i \wedge Carry_{i-1}) = Carry_i \quad (18)$$

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

$$g_i = A_i \wedge B_i \quad (19)$$

The Carry Propagate function is defined to be

$$p_i = A_i \oplus B_i \quad (20)$$

Thus, the Sum equation is redefined to be

$$p_i \oplus Carry_{i-1} = Sum_I \quad (21)$$

and the Carry equation is redefined to be

$$g_i \vee (p_i \wedge Carry_{i-1}) = Carry_i \quad (22)$$

While this does help to reduce the complexity, these equations are still subject to what is known as the ripple carry effect. Bit 0, the least significant bit, has all of its input present immediately. However, Bit 1 must wait until Bit 0 completes its Carry Out before it has all of its inputs. Similarly, Bit 2 must wait for Bit 1, and so on. If each adder bit is exactly identical to all others, then it can be expected that there will be a consistent delay, say t_d, between the carry input and the sum and carry output. This means that if the two addends arrive at time T=0, then Sum_0 is available at T=t_d; Sum_1 is available at T=2t_d; and Sum_7 is available after T=8t_d. The total propagation delay increases linearly with the number of bits to be added.

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,

$$Cout_0 = g_0 \quad (23)$$

$$Cout_1 = g_1 \vee p_1 g_0 \quad (24)$$

$$Cout_2 = g_2 \vee p_2 (g_1 \vee p_1 g_0) \quad (25)$$

$$Cout_3 = g_3 \vee p_3 (g_2 \vee p_2 (g_1 \vee p_1 g_0)) \quad (26)$$

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

$$Cout_2 = g_2 \vee p_2 \overbrace{(g_1 \vee p_1 g_0)}^{G_{01}} \quad (27)$$

and,

$$Cout_3 = \overbrace{(g_3 \vee p_3 g_2)}^{G_{23}} \vee \underbrace{(p_3 p_2)}_{P_{23}} \overbrace{(g_1 \vee p_1 g_0)}^{G_{01}} \quad (28)$$

Thus, if we define two equations,

$$G_{xy} = g_y \vee p_y g_x \quad (29)$$

and,

$$P_{xy} = p_x \wedge p_y \quad (30)$$

Then the CarryOut equations become

$$Cout_0 = g_0 \quad (31)$$

$$Cout_1 = g_1 \vee p_1 (g_0) = G_{01} \quad (32)$$

$$Cout_2 = g_2 \vee p_2 G_{01} = G_{012} \quad (33)$$

$$Cout_3 = G_{23} \vee P_{23} G_{01} = G_{0123} \quad (34)$$

and the Sum equations become

$$Sum_i = p_i \oplus G_{((i-1)\,(i-2)\ldots(0))} \quad (35)$$

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

$$.(g,p)\,o\,(g',p') = (g \vee (p \wedge g'), p \wedge p') \quad (36)$$

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.

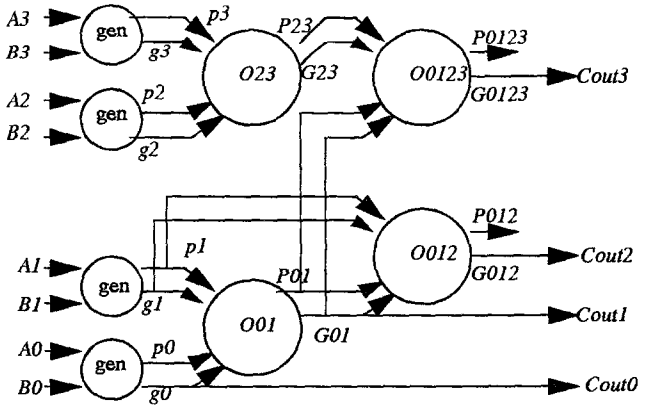


Fig. 5 Binary Carry Lookahead Generation

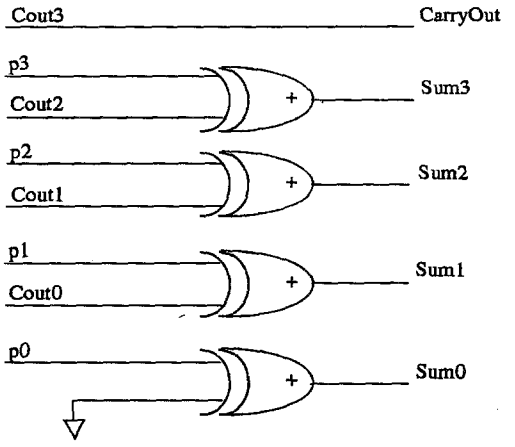


Fig. 6 Addition Process given Carry Lookahead Generation

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. The following is a list of the preliminary testing results

- a) The time delay along the clock tree was shown to be 3.5ns.
- b) In clocked mode the chip appears to actually add at 200Mhz.
- c) In fall-through mode the ADDER chip can easily add at 150Mhz.
- d) In **fall through** mode the time from Datain rising edge to the Mantissa out data valid was between **37ns to 44ns.**
- e) **Rise time** of the Clockout undelayed was **5.85ns** and the **Fall time** was **5.2ns**.

There was one minor design error found which related to the overflow bit and the Mantissas. This problem will be corrected in future designs.

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

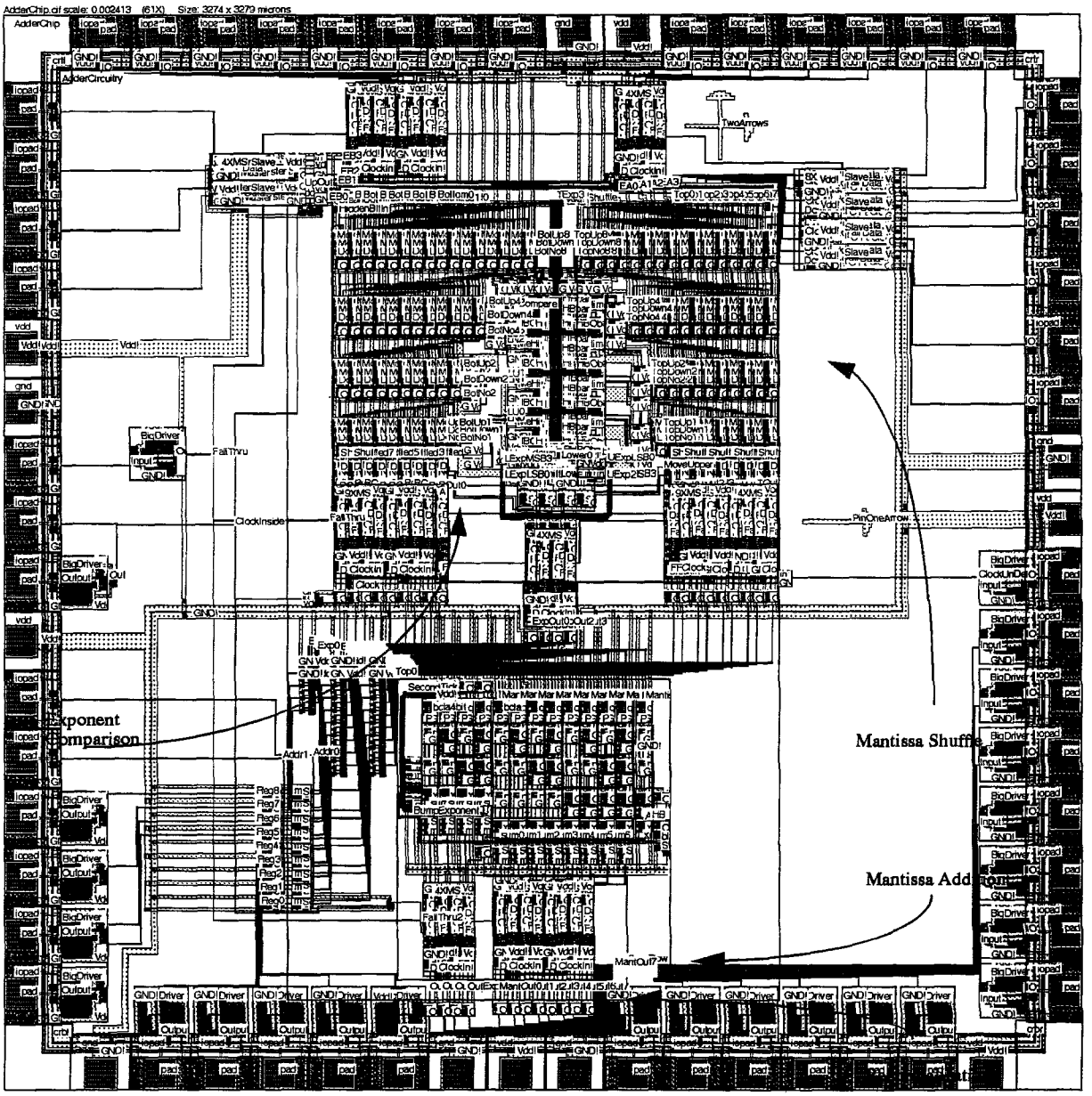
VIII. ACKNOWLEDGEMENTS

I would like to acknowledge the work of Marc Larwill and Cecil Needles who designed and built the test station for the AdderChip and performed the tests of the AdderChip listed above.

Appendix

Circuits and Layouts

Fig. A.1 The AdderChip



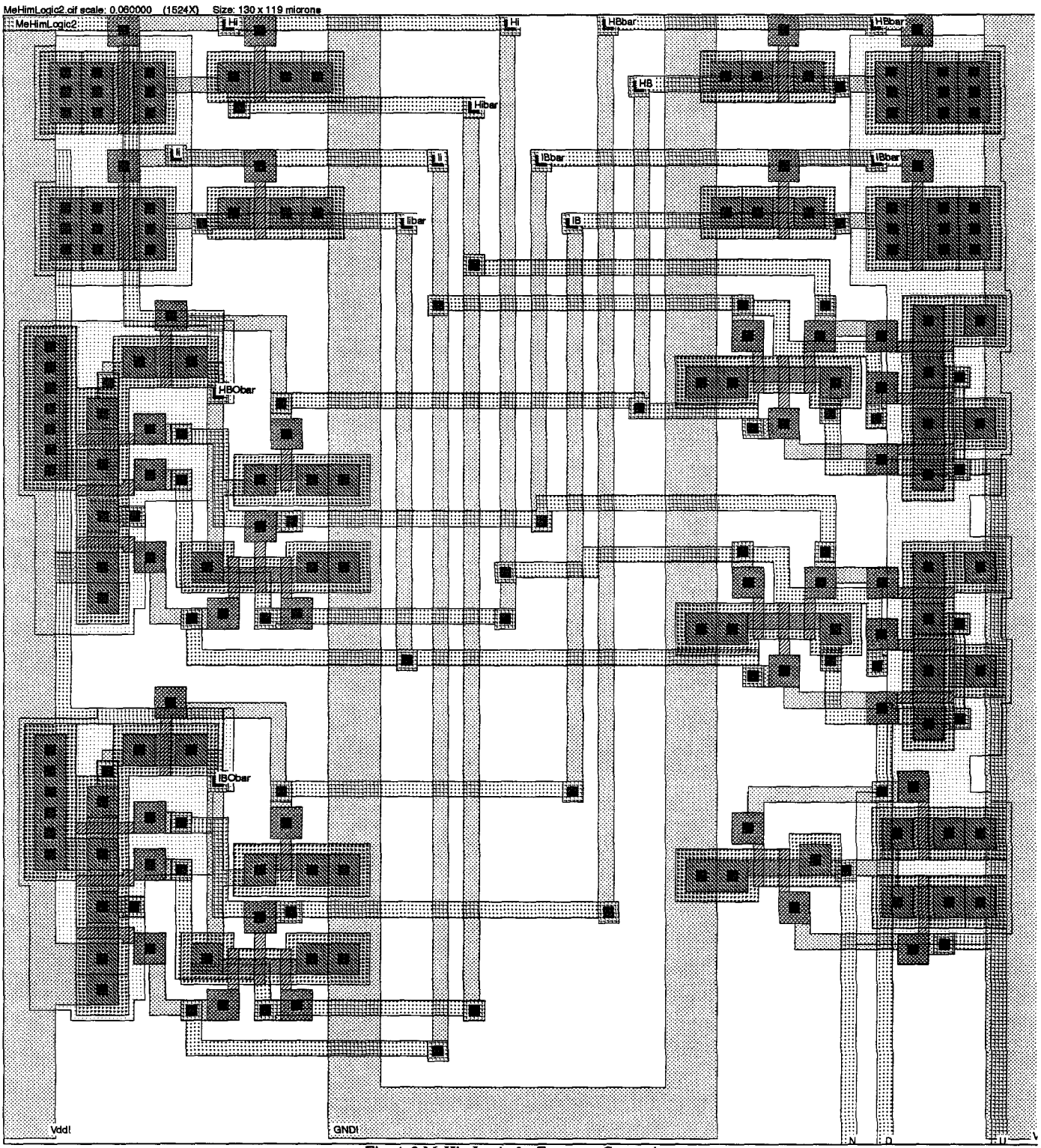


Fig. A.2 MeHimLogic for Exponent Comparison

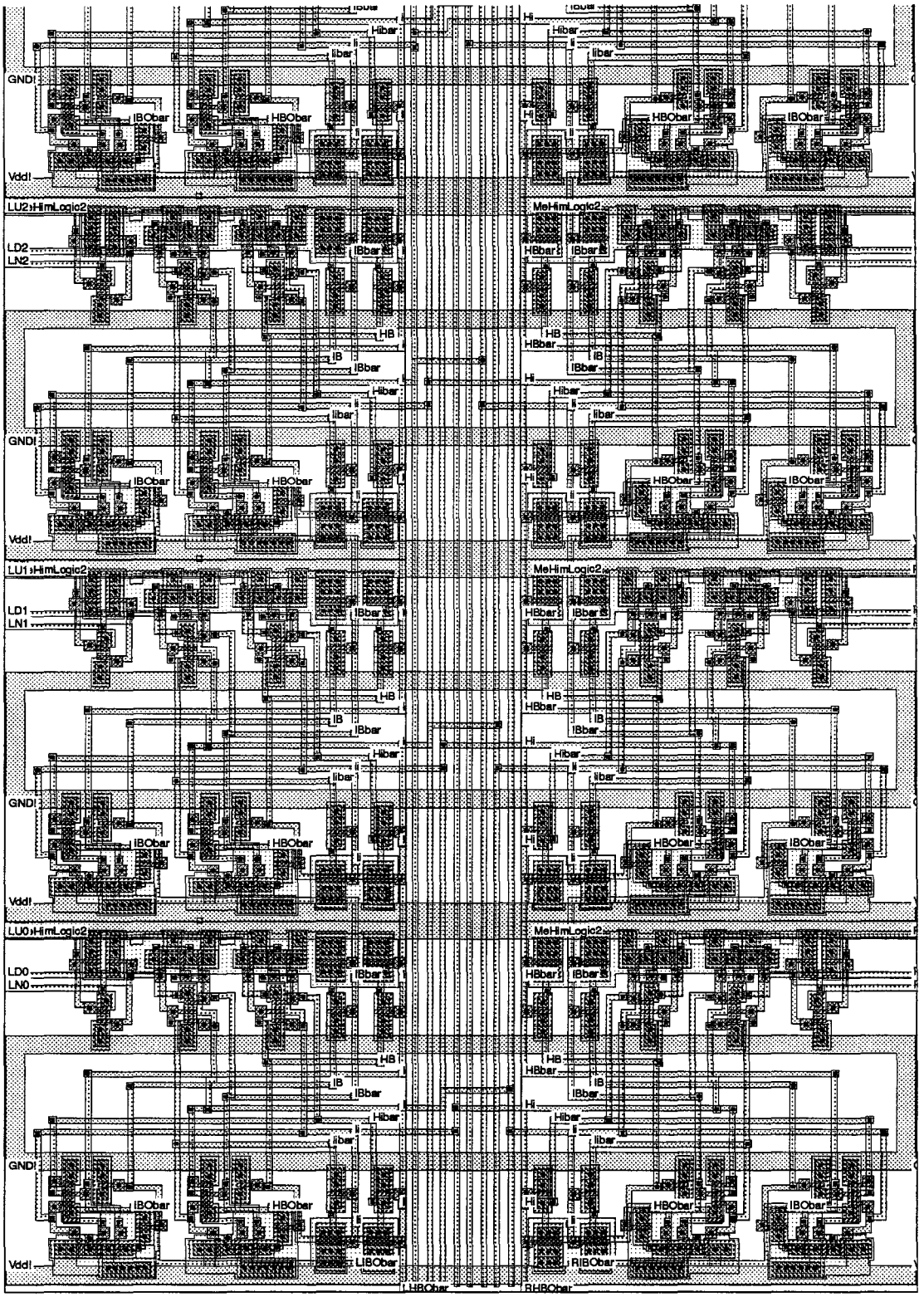


Fig. A.3 Exponent Comparison Circuitry

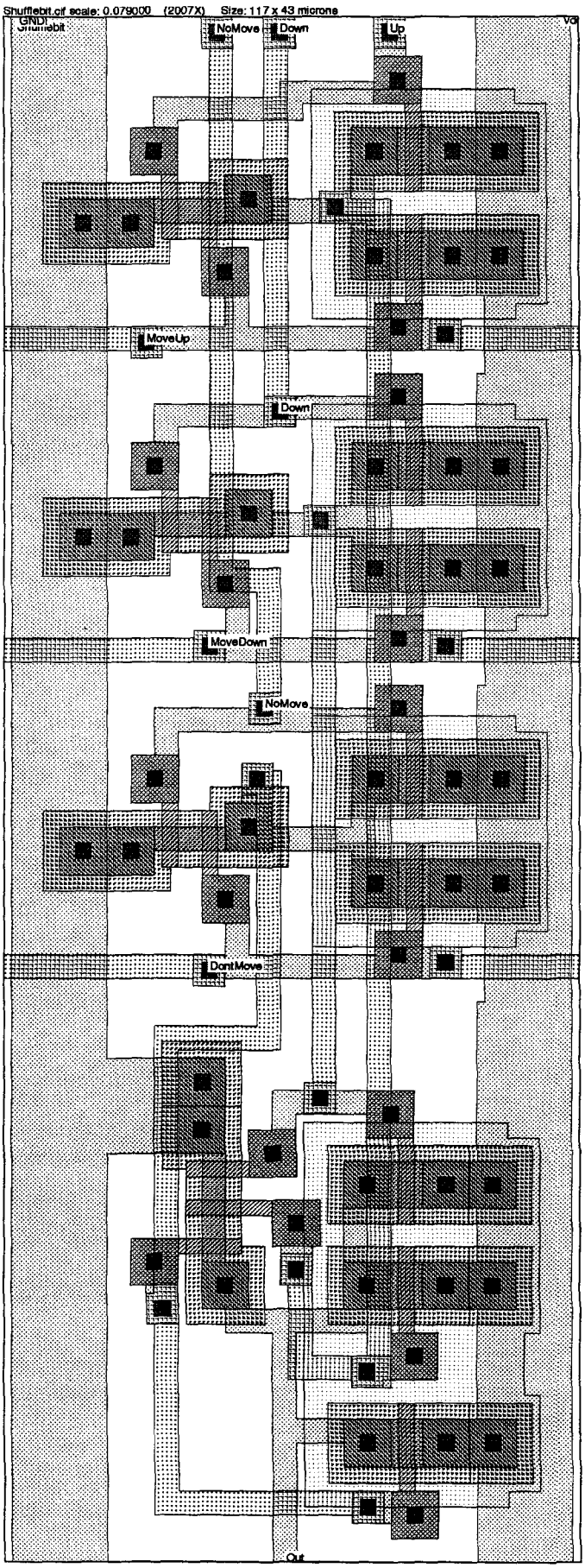


Fig. A.4 A Single Bit Shuffle Stage

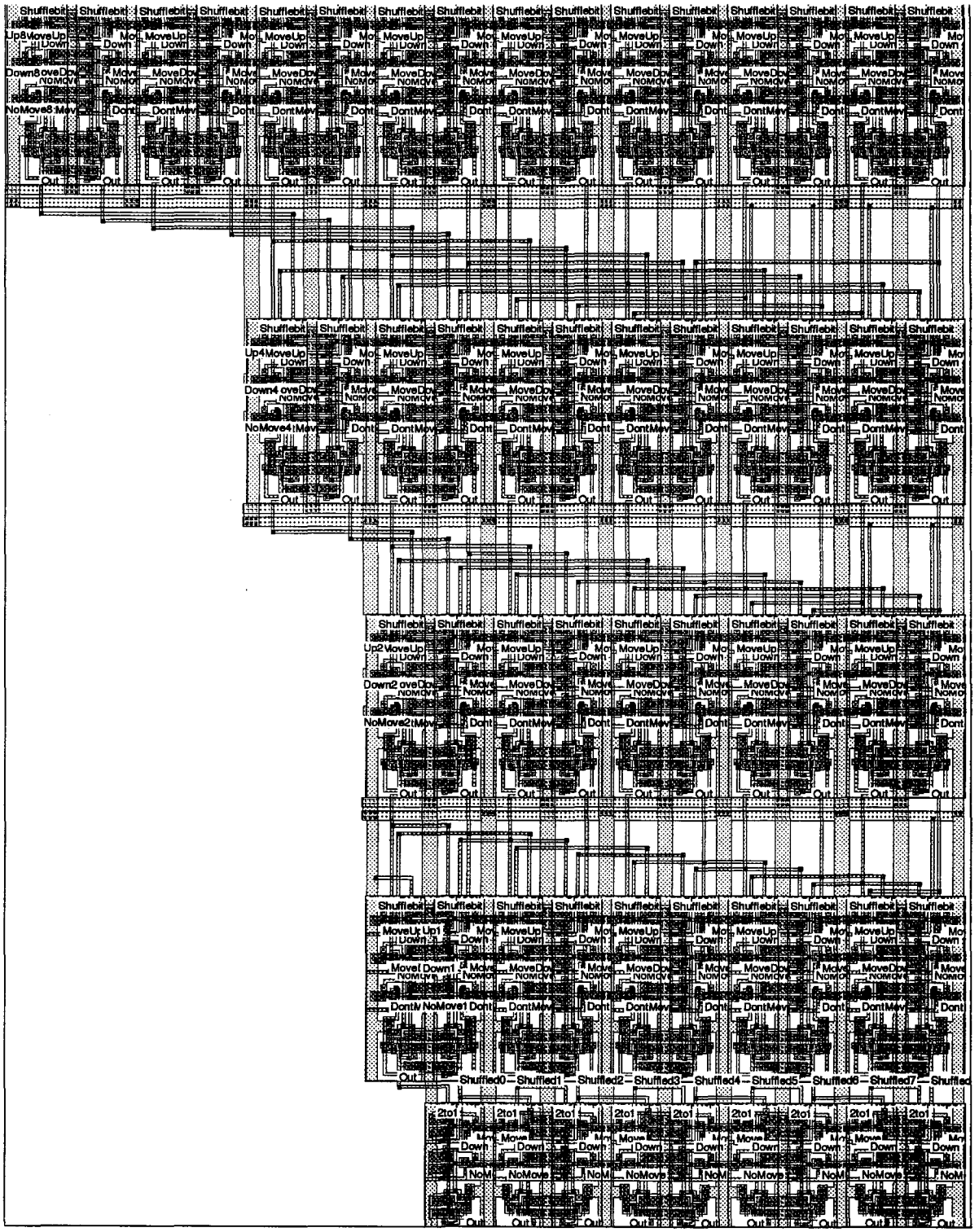


Fig. A.5 Shuffle Circuitry

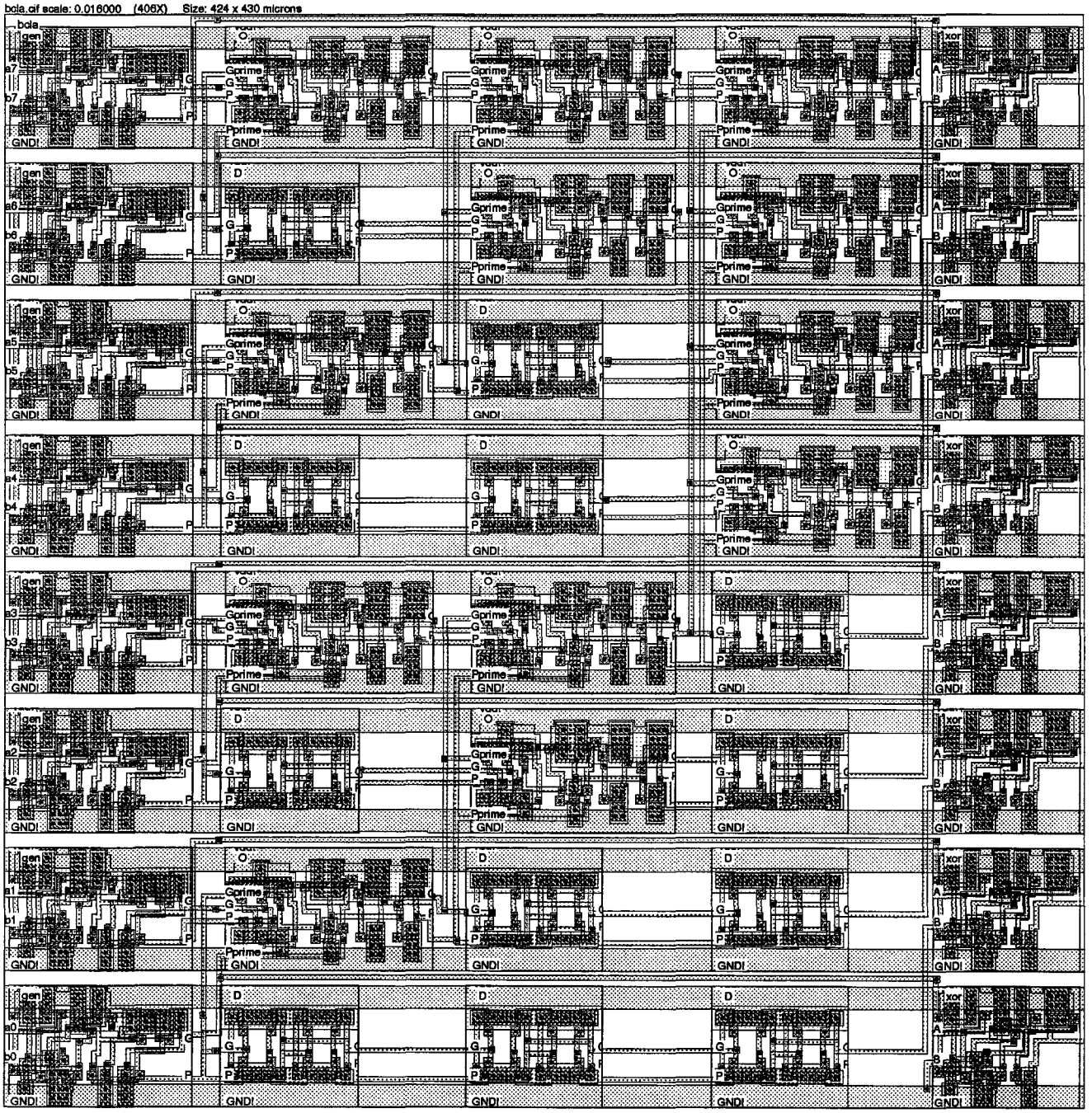


Fig. A.6 The Binary Carry Lookahead Adder (Mantissa Addition)

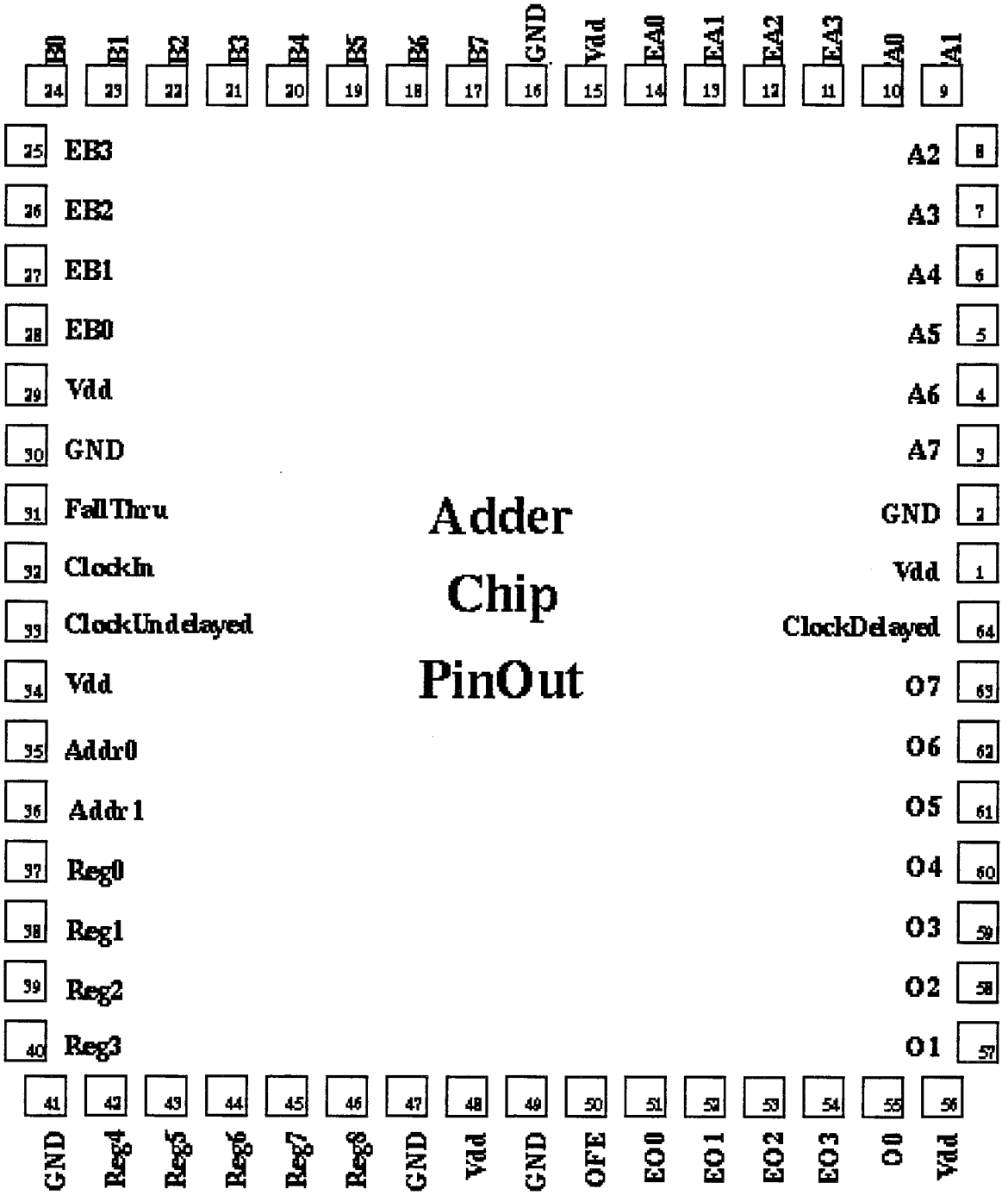


Fig. A.7 The AdderChip Pinout