

# Low-Power Single-Precision IEEE Floating-Point Unit

by

Sheetal A. Jain

Submitted to the Department of Electrical Engineering and Computer  
Science

in partial fulfillment of the requirements for the degree of

Master of Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2003

© Sheetal A. Jain, MMIII. All rights reserved.

The author hereby grants to MIT permission to reproduce and  
distribute publicly paper and electronic copies of this thesis document  
in whole or in part.

Author .....  
Department of Electrical Engineering and Computer Science  
May 21, 2003

Certified by .....  
Krstel Asanovic  
Assistant Professor  
Thesis Supervisor

Accepted by .....  
Arthur C. Smith  
Chairman, Department Committee on Graduate Students



# Low-Power Single-Precision IEEE Floating-Point Unit

by

Sheetal A. Jain

Submitted to the Department of Electrical Engineering and Computer Science  
on May 21, 2003, in partial fulfillment of the  
requirements for the degree of  
Master of Engineering

## Abstract

Floating point adders are area and power intensive, but essential in high performance systems. The Software-Controlled Architectures and Low Energy (SCALE) project requires a low-power single-precision IEEE floating-point adder cluster. Two adder architectures, one containing a single longer computational path, and one containing two shorter parallel computational paths were implemented using minimal area modules. Inputs to the parallel computational paths were registered, and only enabled when that computational path was valid, reducing switching activity. Energy measurements were made of the dual path adder with and without inhibit control and the single path adder, to determine the most energy efficient design.

Thesis Supervisor: Krste Asanovic

Title: Assistant Professor



## Acknowledgments

I would like to thank Professor Krste Asanovic for his guidance and the opportunity to work on this project. I would also like to acknowledge the members of the Assam group for their encouragement and unfailing patience in answering my innumerable questions, especially Ken Barr, Chris Batten, Ronny Krashinsky and Mike Zhang. Albert Ma wrote all the scripts used to translate Verilog RTL to a Spice file, and set up the Testfloat test harness. I would like to especially acknowledge Albert for his guidance in navigating the worlds of synthesis, TestFloat and IEEE floating-point, without which this project could not have been finished.



# Contents

<b>1</b>	<b>Introduction</b>	<b>13</b>
1.1	Organization . . . . .	14
<b>2</b>	<b>Floating Point Unit Specifications</b>	<b>15</b>
2.1	IEEE Floating-Point Design Criteria . . . . .	15
2.1.1	Numerical encoding . . . . .	15
2.1.2	Rounding Modes . . . . .	17
2.1.3	Exceptions . . . . .	17
2.2	SCALE Floating-Point Design Criteria . . . . .	19
2.2.1	Rounding Modes . . . . .	19
2.2.2	NaNs . . . . .	19
2.2.3	Exceptions and Denormalized Numbers . . . . .	20
2.2.4	Comparison Unit . . . . .	21
2.2.5	Timing . . . . .	21
<b>3</b>	<b>FP Addition Architectures</b>	<b>23</b>
3.1	FP Addition Algorithms . . . . .	23
3.1.1	Single path adder . . . . .	23
3.1.2	Dual path adder . . . . .	24
3.1.3	Rounding . . . . .	24
3.1.4	Exceptions . . . . .	25
3.2	Low-Power Adder Architectures . . . . .	26
3.2.1	Simplification of Rounding Modes . . . . .	26

3.2.2	Reduction in significand bandwidth . . . . .	26
3.2.3	Low-power triple path adder . . . . .	26
<b>4</b>	<b>Floating Point Adder Design</b>	<b>29</b>
4.1	Customized Integer Adders . . . . .	29
4.2	Single Path Adder . . . . .	30
4.2.1	Exponent and Significand Decoding . . . . .	30
4.2.2	Swap Unit . . . . .	33
4.2.3	Sticky Shift Unit . . . . .	33
4.2.4	Significand Addition . . . . .	34
4.2.5	Normalization Unit . . . . .	34
4.2.6	Rounding Unit . . . . .	35
4.2.7	Output Select . . . . .	35
4.3	Dual Path Adder . . . . .	37
4.3.1	N-path . . . . .	37
4.3.2	R-path . . . . .	40
4.3.3	Output Select . . . . .	43
<b>5</b>	<b>Implementation and Testing</b>	<b>45</b>
5.1	Test Strategy . . . . .	45
5.2	Toolflow . . . . .	46
5.3	Pipelining the Dual Path Adder . . . . .	48
5.4	Pipelining the Single Path Adder . . . . .	49
<b>6</b>	<b>Results</b>	<b>51</b>
6.1	Energy Measurement Methodology . . . . .	51
6.2	Energy Dissipation Results . . . . .	52
<b>7</b>	<b>Conclusion</b>	<b>53</b>

# List of Figures

2-1	Representation of Single-Precision IEEE FP Number . . . . .	15
4-1	Adder with Unequal Size Inputs . . . . .	30
4-2	Single Path Adder: Block Diagram . . . . .	31
4-3	Single Path Adder: Datapath Diagram . . . . .	32
4-4	Dual Path Adder: Block Diagram . . . . .	38
4-5	Dual Path Adder: N-path . . . . .	39
4-6	Dual Path Adder: R-path . . . . .	41
5-1	Synthesis Tool Flow . . . . .	47



# List of Tables

2.1	Single-Precision FP Number Representation to Value Mapping . . . .	16
2.2	Results in the event of an untrapped overflow . . . . .	18
2.3	Encoding of Rounding Modes . . . . .	19
2.4	Encoding of Comparison Predicates . . . . .	21
4.1	Exponent/Significand Decoder Signal Summary . . . . .	33
4.2	Swap Unit Signal Summary . . . . .	33
4.3	Normalization Unit Signal Summary . . . . .	35
4.4	Rounding Unit Signal Summary . . . . .	35
4.5	N-path Signal Summary . . . . .	37
4.6	Exponent Increments in the R-Path and Single-Path Datapath . . . .	42
4.7	R-path Signal Summary . . . . .	42
5.2	Dual Path Adder Critical Path Delay . . . . .	48
5.4	Single Path Adder Critical Path Delay . . . . .	49
6.1	Area and Energy Dissipation Results . . . . .	52



# Chapter 1

## Introduction

High performance embedded systems require a low-power implementation of floating-point operations. As Moore's law continues to hold, more transistors are packed into a smaller area, providing more potential computational power at a lower price over the same area. As a result embedded systems are becoming ubiquitous, and increasingly complex. Ever-increasing performance is demanded of such computing devices as cell phones, laptops, PDAs and digital cameras. A software implementation of floating-point operations is thus inadequate, and a hardware floating-point unit is required. While speed and area are still important design considerations, power dissipation has come to the forefront, since the amount of power dissipated in a circuit increases both with clock speed and number of transistors. Therefore the floating-point unit on an embedded system must be optimized for minimal power dissipation.

The SCALE (Software-Controlled Architectures and Low Energy) project proposes a general purpose, high performance, configurable architecture for embedded computing. A major SCALE design goal is providing software with fine grain control over system power usage, using compile time knowledge to reduce energy dissipation at run-time. A SCALE chip consists of an array of tiles connected via a fast communication network. The proposed SCALE tile architecture contains a control unit, local tile memory, a network switch connection and an array of parallel lanes, where each lane contains multiple execution clusters[4].

The goal of this thesis is the implementation of a single precision IEEE compliant

floating-point adder cluster for the SCALE architecture. The primary design goal is minimal power dissipation.

A floating-point adder design with a single computational path was first implemented in Verilog RTL and tested using Testfloat. This design was then modified to have two parallel computational paths which made different assumptions about the inputs, simplifying the logic inside the paths. This dual path design determines which path is the correct one by the time each path has finished computing, and propagates the result from the correct path onto the output. After functionality was verified, both designs were synthesized to a standard cell library. By setting timing constraints during synthesis, the critical path in each design was determined. Based on this information and the desired cycle time, both designs were pipelined into three stages. The first stage of the dual path adder contains the common logic for both computational paths and determines which computational path will provide the desired answer. Inputs to the other computational path are not enabled on the next cycle, and thus internal nodes will not switch state, eliminating power dissipation due to switching activity within that path. A version of the dual path design that did not inhibit inputs to the computational paths was also implemented. The three designs were reverified with Testfloat and resynthesized. The synthesized designs were placed and routed, and underwent parasitic extraction to a transistor level model. A Spice simulator was used to verify functionality and make power measurements.

## 1.1 Organization

This thesis is structured as follows. Chapter 2 provides an overview of the IEEE floating-point standard and discusses SCALE specific design criteria. Chapter 3 discusses known algorithms and low-power optimizations for floating-point addition. Chapter 4 describes the two floating-point adder designs implemented. Chapter 5 describes how the designs were implemented and tested. Chapter 6 presents area, time and energy data about the designs. Chapter 7 concludes the paper and suggests directions for future research.

# Chapter 2

## Floating Point Unit Specifications

IEEE Standard 754 is the de-facto functional definition of floating point arithmetic. This chapter presents the specifications of the standard relevant to floating point addition. Interpretations of the standard's options and other design criteria for the SCALE floating-point adder are also presented.

### 2.1 IEEE Floating-Point Design Criteria

The IEEE standard defines floating-point number formats, floating point arithmetic operations, conversions between other number formats, and floating-point exceptions [3].

#### 2.1.1 Numerical encoding

The standard defines 32-bit single-precision and 64-bit double precision numbers. A single-precision FP number is encoded by a 1-bit sign, an 8-bit exponent and a 23-bit significand, as shown in Figure 2-1.

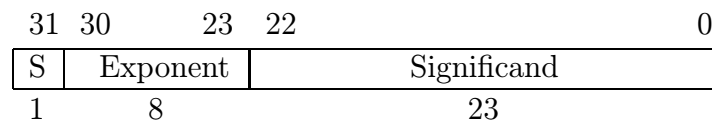


Figure 2-1: Representation of Single-Precision IEEE FP Number

Exponent values of 0xFF and 0x00 are reserved to encode special numbers such as zero, denormalized numbers, infinity and NaNs. The mapping from an encoding of a single-precision floating-point number to the number's value is summarized in Table 2.1.

Sign	Exponent	Significand	Value	Description
S	0xFF	0x00000000	$(-1)^S \infty$	Infinity
S	0xFF	$F \neq 0$	NaN	Not a Number
S	0x00	0x00000000	0	Zero
S	0x00	$F \neq 0$	$(-1)^S 2^{-126} 0.F$	Denormalized Number
S	$0x00 < E < 0xFF$	F	$(-1)^S 2^{E-127} 1.F$	Normalized Number

Table 2.1: Single-Precision FP Number Representation to Value Mapping

### Normalized Numbers

For all ordinary numbers, the exponent field represents the exponent of a number plus a bias of 127. A sign bit of 1 indicates a negative number. The highest bit of the significand is always assumed to be 1, and therefore the representation only encodes the lower 23 bits. This highest bit is often referred to as a hidden bit. These are known as normalized numbers since the digit to the left of the binary point is 1.

### Denormalized Numbers

Denormalized numbers are defined as all numbers with an exponent field equal to zero and a nonzero significand. The highest (or hidden) bit of the significand is assumed to be zero. Since the digit to the left of the binary point is 0, these numbers are not normalized. Denormalized numbers fill in the representation gap between zero and the smallest normalized number.

### NaNs

A NaN is indicated by an exponent equal to 0xFF and a nonzero significand. The sign bit for a NaN is not interpreted. A NaN has no hidden bit. The standard defines Signaling (SNaN) and Quiescent (QNaN) NaNs but does not specify how they are

represented. The convention is for all QNaNs to have the highest bit of the significand field set to 1. In the absence of a trap, any operation on a SNaN produces a QNaN. Any operation on one or more QNaNs produces one of the input QNaNs as a result.

## **Infinity**

Infinity is represented by an exponent of 0xFF and a significand of zero.

## **Zero**

Zero is represented by an exponent and significand of zero.  $-0$  and  $+0$  always compare equal, but the required sign bit is defined for the zero results of an operation. Any operation on two nonzero operands that produces a zero result has a sign of  $+$  in all rounding modes except round toward  $-\infty$ . The effective addition of two zeros of different sign follows the same convention. The effective addition of two zeros with the same sign propagates that sign to the output in all rounding modes.

### **2.1.2 Rounding Modes**

Four user-selectable rounding modes are defined: round to nearest even (RNE), round toward  $+\infty$  (RP), round toward  $-\infty$  (RM), and round toward 0 (RZ). The default mode, RNE, rounds the intermediate result to the nearest representable number, choosing an even number if the result is equidistant from the two nearest numbers. The rounding mode affects the results of most arithmetic operations, the sign of zero sums, and thresholds for the overflow and underflow exceptions.

### **2.1.3 Exceptions**

The standard defines five types of exceptions: invalid operation, division by zero, overflow, underflow and inexact. An exception is signaled by setting a status flag and/or taking a trap. The returned result in the case of an exception depends on whether the corresponding trap is enabled and on the rounding mode.

## Invalid Operation

The given operation cannot be performed on the operands. For an adder, these are the effective subtraction of infinities and the input of a signaling NaN. In the absence of a trap, the result is a quiescent NaN.

## Division By Zero

Divisor is zero. In the absence of a trap, the result is an appropriately signed  $\infty$ .

## Overflow

The rounded result of the operation is larger in magnitude than the largest possible finite number that can be represented. The representation of the largest possible finite single-precision number is E=0xFE, F=0x7FFFFFFF or equivalently,  $3.6875 \times 10^{19}$ . Table 2.2 summarizes the results of an overflow according to the rounding mode and the sign of the intermediate result.

Rounding Mode	Sign	Result
Round to zero	S	$(-1)^S 3.6875 \times 10^{19}$
Round to nearest even	S	$(-1)^S \infty$
Round toward $+\infty$	+	$+\infty$
Round toward $+\infty$	-	$-3.6875 \times 10^{19}$
Round toward $-\infty$	+	$+3.6875 \times 10^{19}$
Round toward $-\infty$	-	$-\infty$

Table 2.2: Results in the event of an untrapped overflow

## Underflow

In absence of a trap handler, an underflow exception is raised when both the conditions *tininess* and *loss of accuracy* are detected. Tininess occurs when a nonzero result computed as though exponent range and precision were unbounded would lie strictly between  $\pm 2^{-126}$ , and may be detected either before or after rounding. Loss of accuracy may be detected identically to the inexact exception or as a difference between the delivered result and what would have been computed if the exponent

range was unbounded. An untrapped underflow exception does not affect the result of an operation.

### **Inexact**

An inexact exception is raised when the rounded result is not exact or an overflow exception is raised and not trapped. The inexact exception does not affect the result of an operation.

## **2.2 SCALE Floating-Point Design Criteria**

The floating-point unit will operate on single-precision numbers. Since the SCALE architecture makes use of the MIPS ISA, the encoding of rounding modes, the default NaN, and the interpretation of the underflow exception are chosen so as to be MIPS-compliant[5].

### **2.2.1 Rounding Modes**

The SCALE floating-point unit supports all four rounding modes. The encodings for the rounding modes are shown in Table 2.3 and are taken from the MIPS ISA.

Rounding Mode	Encoding
Round to Nearest Even	00
Round to Zero	01
Round to +Infinity	10
Round to -Infinity	11

Table 2.3: Encoding of Rounding Modes

### **2.2.2 NaNs**

If a NaN is generated as an output where neither input is a NaN, the output NaN is 0x7F7FFFFFFF, to be compliant with the MIPS ISA. If an input is a SNaN, the output

will always be the QNaN formed by flipping the highest bit of the significand field. If an input is a QNaN, the output will be that QNaN.

### 2.2.3 Exceptions and Denormalized Numbers

The SCALE floating-point adder supports three of the four relevant exceptions: invalid operation, overflow, and inexact. Two exceptions, denormal in and denormal out, have been added to indicate a denormalized number as input and output, respectively.

The adder signals an exception by setting a corresponding output flag at the same time the sum is delivered. The adder does not support trap handlers for the exceptions. The surrounding hardware or a software handler may provide trap handlers, as well as modify or propagate exception flags and the output sum to the software interface of the floating-point unit.

Input and output denormalized numbers are flushed to zero. Although denormalized numbers could be handled by the adder with very little extra overhead in area and time, the same is not true of the SCALE floating-point multiplier. The increased cost in area, time and power necessary to support denormalized numbers in the multiplier was deemed prohibitive. For the sake of consistency, the adder does not handle denormalized numbers and therefore is not fully IEEE compliant. Should IEEE compliance be desired in the overall system, a software handler will be able to access the floating-point unit inputs, output sum and exception flags, and produce the required result.

The underflow exception is signalled by detecting tininess after rounding and detecting loss of accuracy as an inexact result, in order to be compliant with MIPS. Tininess after rounding is defined as the computation of a nonzero result that lies between  $\pm 2^{-126}$ . However, in the case of addition, any such result would be a representable denormalized number. Therefore tininess is already signalled by the denormal out exception. Loss of accuracy is already signalled by the inexact exception. The underflow exception signal can be generated by other hardware or by a software handler, depending on how IEEE compliance is to be handled in the overall system.

Therefore, the adder does not implement the underflow exception.

## 2.2.4 Comparison Unit

The SCALE floating-point adder contains a comparison unit and outputs a 2-bit value to indicate which of the four predicates, less than, equal, greater than or unordered, is true. The encodings for the predicates are shown in Table 2.4. x

Predicate	Symbol	Encoding
Less Than	<	00
Equal	=	01
Greater Than	>	10
Unordered	?	11

Table 2.4: Encoding of Comparison Predicates

## 2.2.5 Timing

The SCALE floating-point adder should run at 200 MHz when the Verilog model is synthesized using the standard cell library. It will therefore be pipelined into at most four stages with a 5 ns cycle time.



# Chapter 3

## FP Addition Architectures

This chapter describes algorithms that detect exception conditions during floating-point addition, perform the addition of two normalized numbers, and round the resulting sum. This chapter also presents previously published approaches to low-power floating-point adder design.

### 3.1 FP Addition Algorithms

Two standard floating-point addition architectures are presented in algorithmic form. Both algorithms are presented as if operating on single-precision normalized inputs. For ease of notation, the following quantities are defined:

Let  $\{s_a, e_a, f_a\}$  and  $\{s_b, e_b, f_b\}$  be the signs, exponents and significands respectively of two input floating-point numbers  $a$  and  $b$ . Let  $SOP$  denote whether the operation is a subtraction (1) or addition (0).

#### 3.1.1 Single path adder

1. Compute the effective operation  $seff = SOP \oplus s_a \oplus s_b$ .
2. Calculate  $\delta_e = |e_a - e_b|$  and identify the input containing larger significand.
3. Assign  $(e_1, \{1, f_1\})$  and  $(e_2, \{1, uf_2\})$  to be the exponents and significands of the inputs with the larger absolute value and smaller absolute value, respectively.

4. Shift  $uf_2$  right by  $\max(\delta_e, 24)$  places to align it with  $f_1$ , producing  $f_2$ .
5. Compute the significand sum  $f\_sum = f_1 \pm f_2$  according to  $seff$ .
6. Normalize the result by shifting  $f\_sum$  until the highest bit of the significand is 1 and assign it to  $f\_norm$ .
7. Increment or decrement  $e_1$  by the shift amount of the prior step to produce the exponent of the result.
8. Round  $f\_norm$ . Renormalize the significand and increment the exponent if necessary.

This is the basic implementation of a floating-point adder. It has the advantage of low area, and the disadvantage of high latency.

### 3.1.2 Dual path adder

The design is split into a near (N) path and a far (R) path. The near path assumes the effective operation is a subtract and the absolute value of the exponent difference is less than 2. The far path assumes the opposite. Each path computes the operation in parallel, and the correct output is selected at the end.

The computation carried out in each path is almost identical to that of a single path adder. However, in the near path  $uf_2$  need only be shifted by at most one place to the right, which eliminates a step that executes in logarithmic time. In the far path  $f\_sum$  will only need to be shifted by at most one place to normalize the result, again eliminating a logarithmic time step[9, 8].

A dual path adder is faster than a single path adder, but takes more area and consumes more power.

### 3.1.3 Rounding

The rounding logic is identical in the single path and dual path adders. Rounding is implemented as follows:

After  $uf_2$  is shifted to align it with  $f_1$ , it is computationally expensive, as well as unnecessary to keep all the lower bits of  $f_2$ . To ensure accurate computation and rounding, it is sufficient to keep 3 extra bits of  $uf_2$  that are shifted beyond the width of the significand. These lowest three bits are known as the guard, round, and sticky bits respectively, and are carried throughout the computation. The sticky bit is the OR of all bits of  $f_2$  that have been shifted out of range. Within the rounding unit, the guard bit indicates whether to round down or round up in either round infinite mode. Round to zero can be accomplished by truncation. Round to nearest even mode is accomplished by rounding up if the guard bit is set, and then pulling down the lowest bit of the output if the round and sticky bits are zero. It has been shown that the four rounding modes can be reduced to a single rounding mode by adding a 3-bit rounding mode specific injection signal to the normalized sum and truncating the output. For example, in round to nearest even mode, the injection would be 100, so that if the guard bit was set, there would be a carry in to the lowest bit of the significand representation, incrementing the significand by one[7].

### 3.1.4 Exceptions

The overflow exception is set by checking if the exponent sums in the normalization or rounding stages overflow. Since the maximum possible exponent for a normalized number is 0xFE, the sum result must also be less than 0xFF.

Since SCALE does not implement the underflow exception, the conditions necessary to detect it are not described.

The inexact exception is signalled when the result is not exact, i.e., cannot fit into the representation. Therefore it is raised when there is an overflow or when either the guard, round or sticky bit is set.

The denormal out exception is signaled when the exponent sum underflows (becomes negative or zero) in the normalization stage.

The invalid operation and denormal in exceptions are trivial.

## 3.2 Low-Power Adder Architectures

Approaches to low-power adder design fall into two categories: the elimination of features and microarchitectural optimizations. The simplification of rounding modes and the reduction of significand bandwidth fall into the former category. Schemes that selectively enable computational paths, such as Pillai et al.'s low-power triple path adder would fall into the latter category.

### 3.2.1 Simplification of Rounding Modes

Supporting all four rounding modes as specified by the IEEE standard can be quite expensive. Round to nearest even mode requires an extra addition as well as a possible normalization shift. Both round towards  $+\infty$  and  $-\infty$  have the same requirements and thus no energy would be saved by only eliminating one of these three modes. However, limiting rounding to round to zero mode will save energy, as it eliminates both control logic, an extra addition, a normalization shift and a mux. The penalty is a reduction in the accuracy of calculations and possibly misleading results. For certain applications, where less accuracy is required, this optimization is worthwhile, but it is not suitable for SCALE.

### 3.2.2 Reduction in significand bandwidth

The area, and by extension the power consumption of a floating-point adder is highly dependent on the width of the significand. Reducing the width of the significand representation can save significant amounts of power [10]. This approach is suitable for specialized applications when precision may be sacrificed.

### 3.2.3 Low-power triple path adder

This architecture contains 3 datapaths, of which only one is operational at a time. Two of the paths are identical in functionality to the far and near paths described in the dual path adder. The third path is a bypass path which takes care of infinite

inputs, NaN, and the case when the exponent difference is greater than the width of the significand field. For each computation, signals are only propagated to the path that computes the correct result. The inputs to the other two paths are inhibited by clock gating, eliminating power dissipation via switching on these paths. Power is also reduced by implementing pseudo Lazy Zero Anticipation (LZA) logic instead of a Count Leading Zeros module and merging rounding with significand addition. For single-precision formats, it is estimated that this scheme cuts power consumption by 90% when compared to a similarly configured floating-point adder that implements full LZA [6].

The dual path design presented in this thesis uses the same concept of inhibiting computational paths, but implements and enables the computational paths differently.



# Chapter 4

## Floating Point Adder Design

A single path and dual path adder were both implemented and have identical functionality. The single path adder provides a base line for measuring power consumption. The dual path adder attempts to minimize power consumption by reducing switching activity. Once it is determined whether the result of an input will come from the R-path or N-path, inputs are not propagated down the unused path. Therefore on a given cycle, power dissipation due to switching activity is eliminated on the unused path. In addition, for each integer add operation in the datapath, an adder customized for that operation is implemented. These customized integer adders are smaller in area than a multipurpose integer adder, and will dissipate less power.

### 4.1 Customized Integer Adders

A multipurpose integer adder performs addition and subtraction, as well as outputting control signals that indicate if the result is negative, equal to zero, or overflows. For each adder or subtractor in a datapath, the multipurpose adder was modified to create a custom module that only performed the necessary operation and output the necessary control signals. In the N-path of the dual path adder, the significant adder always performs a subtract. Removing the addition functionality of the adder eliminates a 25 bit mux. If two unsigned numbers are to be added, the result can never be zero. Removing the test for zero equality eliminates ORing all the bits of

the result together.

The custom adder that saved the most area in comparison to a multipurpose adder operates on inputs of unequal width. Datapaths in both floating-point adders contain integer additions between such operands. In particular, the addition used to perform rounding has an operand of less than 4 bits and an operand of at least 25 bits. The adder module takes in the length of each input as a parameter. The logic is simplified for the higher order bits of the larger input, reducing both area and latency. In datapath diagrams, these adders are represented as shown in Figure 4-1.

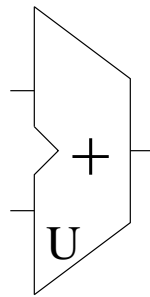


Figure 4-1: Adder with Unequal Size Inputs

## 4.2 Single Path Adder

The overall block diagram for the single path adder is shown in Figure 4-2.

The control unit calculates all exceptions, the effective operation *seff*, and the output sign *sign*, as well as various control signals for the datapath. The datapath contains an exponent and significand decoder, a swap unit, a shifter, an adder, a normalization unit, a rounding unit and an output selection unit. All modules except the exponent and significand unit are shown on the datapath diagram in Figure 4-3.

### 4.2.1 Exponent and Significand Decoding

The exponent and significand decoder produces signals used by the control unit to determine if the inputs are NaN, infinite, zero or denormalized. The signals, their descriptions and the logic equations used to calculate them are listed in Table 4.1.

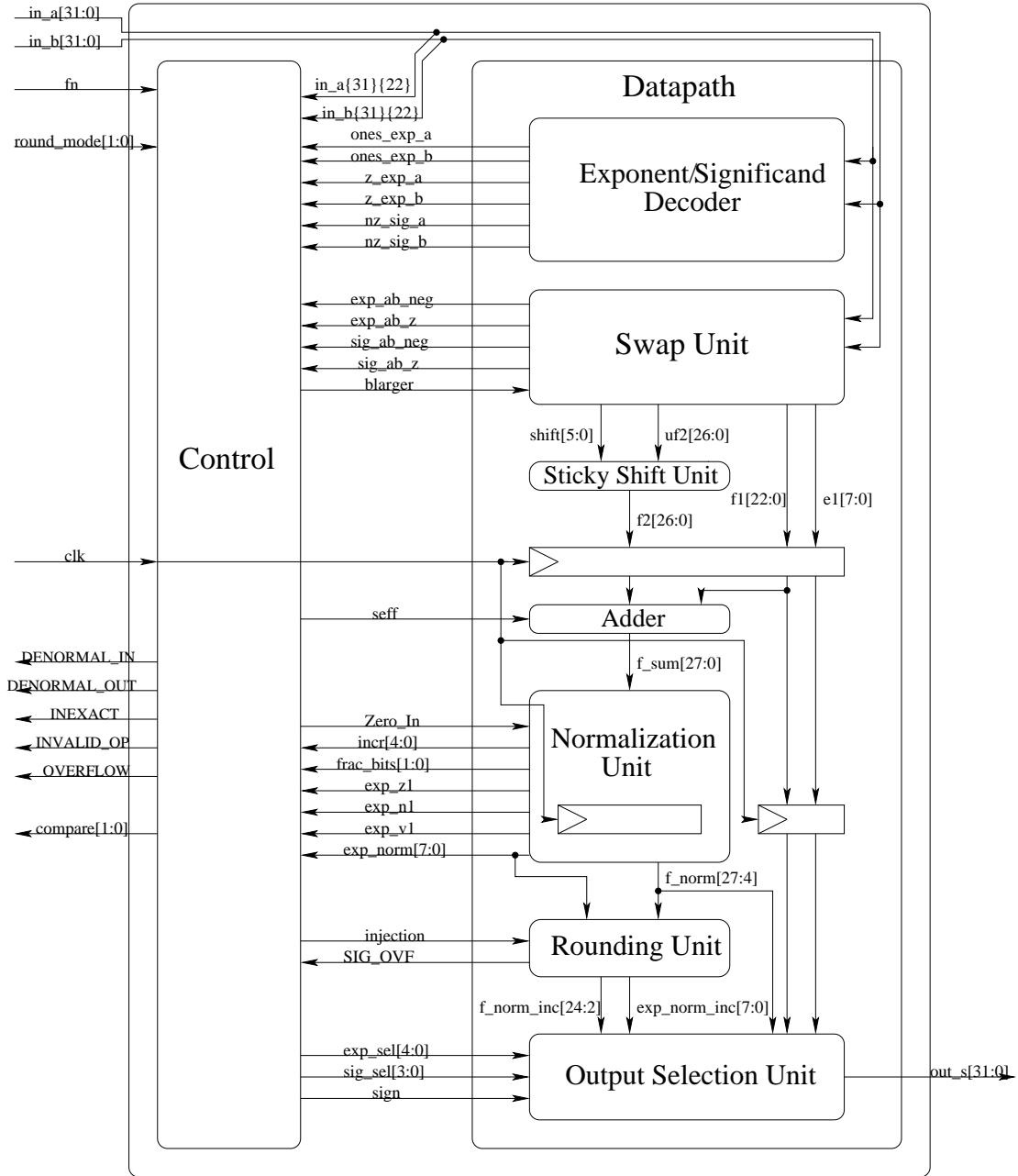


Figure 4-2: Single Path Adder: Block Diagram

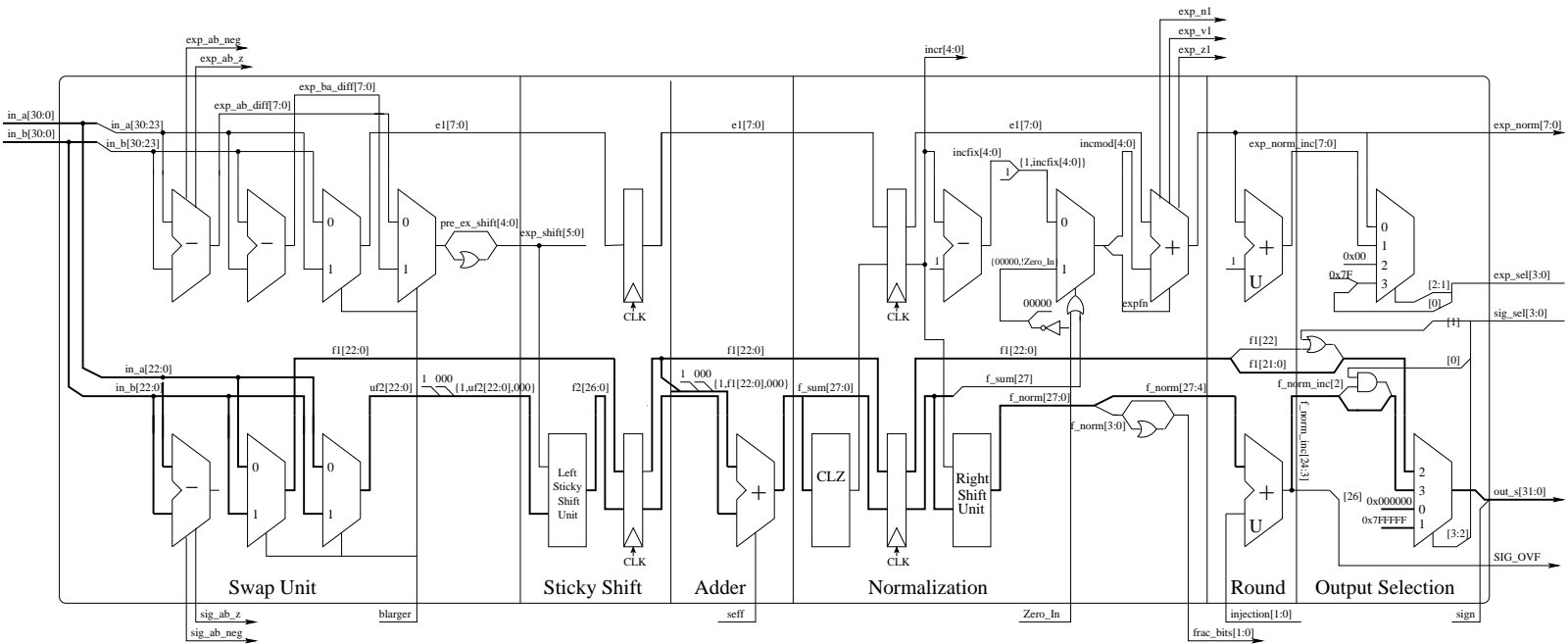


Figure 4-3: Single Path Adder: Datapath Diagram

Signal	Meaning	Calculation
$z\_exp\_a$	Exponent Field of $in_a$ all zeros	NOR of $in_a[30:23]$
$z\_exp\_b$	Exponent Field of $in_b$ all zeros	NOR of $in_b[30:23]$
$ones\_exp\_a$	Exponent Field of $in_a$ all ones	AND of $in_a[30:23]$
$ones\_exp\_b$	Exponent Field of $in_b$ all ones	AND of $in_b[30:23]$
$nz\_sig\_a$	Significand Field of $in_a$ not all zeros	OR of $in_a[22:0]$
$nz\_sig\_b$	Significand Field of $in_b$ not all zeros	OR of $in_b[22:0]$

Table 4.1: Exponent/Significand Decoder Signal Summary

### 4.2.2 Swap Unit

The components of the swap unit are shown in the datapath diagram in Figure 4-3. A summary of relevant signals in the swap unit appears in Table 4.2. Two adders compute the differences  $exp\_ab\_diff$  and  $exp\_ba\_diff$ , and the control unit inputs  $exp\_ab\_neg$  and  $exp\_ab\_z$ . An adder computes the control unit inputs  $sig\_ab\_neg$  and  $sig\_ab\_z$ . The control unit uses these signals to compute signal  $blarger$ , which is then used to select  $pre\_exp\_shift$ ,  $e_1$ ,  $f_1$  and  $uf_2$ . Since  $pre\_exp\_shift$  is 8 bits wide, its top 4 bits are ORed together to produce the 5-bit shift amount  $exp\_shift$ .

Signal	Meaning	Calculation
$exp\_ab\_diff$	$e_a - e_b$	$in_a[30:23] - in_b[30:23]$
$exp\_ba\_diff$	$e_b - e_a$	$in_b[30:23] - in_a[30:23]$
$exp\_ab\_neg$	$e_a < e_b$	Byproduct of $in_a[30:23] - in_b[30:23]$
$exp\_ab\_z$	$e_a = e_b$	Byproduct of $in_a[30:23] - in_b[30:23]$
$sig\_ab\_neg$	$f_a < f_b$	Byproduct of $in_a[22:0] - in_b[22:0]$
$sig\_ab\_z$	$f_a = f_b$	Byproduct of $in_a[22:0] - in_b[22:0]$
$f_1$	Significand of larger input	Mux $in_a[22:0]$ , $in_b[22:0]$ with $blarger$
$uf_2$	Unshifted significand of smaller input	Mux $in_a[22:0]$ , $in_b[22:0]$ with $blarger$
$e_1$	Exponent of larger input	Mux $in_a[30:23]$ , $in_b[30:23]$ with $blarger$
$pre\_exp\_shift$	$ e_a - e_b $	Mux $exp\_ab\_diff$ , $exp\_ba\_diff$ with $blarger$
$exp\_shift$	Shift amount for $uf_2$ to align with $f_1$	$pre\_exp\_shift$ with top bits ORed together

Table 4.2: Swap Unit Signal Summary

### 4.2.3 Sticky Shift Unit

This custom shift unit takes the 23-bit significand  $uf_2$  and shifts it right by  $exp\_shift$  bits to align it with  $f_1$ . Before shifting,  $uf_2$  is concatenated with a single 1 to the

left, and 3 zeros to the right. The 1 represents the hidden bit of the significand not present in the significand field. The places of the zeros represent the added guard, round and sticky bits. As bits are shifted out, they are ORed with the sticky bit.

#### 4.2.4 Significand Addition

The significand addition unit contains an adder which computes the significand sum  $f\_sum$  according to  $seff$ . The hidden significand bit is inserted before the highest bit of  $f_1$  and three zeros are placed after the lowest bit of  $f_1$ . Therefore  $f\_sum = \{1, f_1, 000\} \pm f_2$ .

#### 4.2.5 Normalization Unit

The components of the normalization unit are shown in the datapath diagram in Figure 4-3. A summary of relevant signals in the normalization unit appears in Table 4.3. The Count Leading Zeros (CLZ) module counts the number of leading zeros in the result  $f\_sum$  and stores it in  $incr$ . The shift unit takes in  $f\_sum$  and shifts it right by  $incr$  bits to produce the result  $f\_norm$ . An adder takes  $e_1$  and adds or subtracts exponent increment  $incmod$  according to  $expfn$ , producing  $exp\_norm$ .

$f\_sum$  is a 28-bit quantity which decomposes into an overflow bit, the unit bit, 23 significand bits, and the 3 extra (guard, round, sticky) bits. Hence if  $f\_sum[27]$  is set, the exponent needs to be incremented by one, and 00001 should be muxed into  $\{expfn, incmod\}$ . However, if  $Zero\_In$  is set, the exponent should not be incremented, and 00000 is muxed into  $\{expfn, incmod\}$ . In all other cases, the exponent needs to be decremented by  $incr-1$ , so  $\{1, incfix[4:0]\}$  is muxed into  $\{expfn, incmod\}$ .

$f\_norm$  is a 28-bit quantity which decomposes into the unit bit, 23 significand bits and 4 extra bits (guard, round, sticky1, sticky2). In order to make rounding decisions, it suffices to know if the guard bit is set and if any of the round or sticky bits are set. Therefore the round and sticky bits are ORed together, concatenated with the guard bit, and placed in control unit input  $frac\_bits$ .

Signal	Meaning	Calculation
<i>expfn</i>	Exponent Operation, Subtract=1	Muxed by <i>Zero_In</i> AND <i>f_sum</i> [27]
<i>exp_n1</i>	$exp\_norm < 0$	Byproduct of $e_1 \pm incmod$
<i>exp_v1</i>	$exp\_norm > 0xFF$	Byproduct of $e_1 \pm incmod$
<i>exp_z1</i>	$exp\_norm = 0$	Byproduct of $e_1 \pm incmod$
<i>exp_norm</i>	Normalized exponent	$e_1 \pm incmod$
<i>f_sum</i>	Significand sum	$f_2 \pm \{1, f_1, 000\}$
<i>frac_bits</i>	Control unit input for rounding	$\{f\_norm[4], \text{OR of } f\_norm[3:0]\}$
<i>incfix</i>	Intermediate exponent increment	<i>incr</i> -1
<i>incmod</i>	Exponent increment	Muxed by <i>Zero_In</i> AND <i>f_sum</i> [27]
<i>incr</i>	Number of leading zeros in <i>f_sum</i>	CLZ Unit
<i>f_norm</i>	Normalized result	<i>f_sum</i> rightshifted by <i>incr</i> bits
<i>Zero_In</i>	$in_a = 0$ or $in_b = 0$	From Control unit

Table 4.3: Normalization Unit Signal Summary

## 4.2.6 Rounding Unit

The components of the rounding unit are shown in the datapath diagram in Figure 4-3. A summary of relevant signals in the rounding unit appears in Table 4.4. The control unit supplies the signal *injection*, which is set according to the rounding mode. The result *f\_norm\_inc* contains the rounded result. If the sum overflows, output *SIG\_OVF* is set so the control unit can select incremented exponent *exp\_norm\_inc* as the output exponent.

Signal	Meaning	Calculation
<i>injection</i>	Rounding increment	From Control Unit
<i>f_norm_inc</i>	Rounded result	$f\_norm + injection$
<i>exp_norm_inc</i>	Incremented exponent	$exp\_norm + 1$
<i>SIG_OVF</i>	Significand addition has overflowed	$f\_norm\_inc[26]$

Table 4.4: Rounding Unit Signal Summary

## 4.2.7 Output Select

The control unit provides the signals *exp\_sel* and *sig\_sel* to select the output exponent and significand respectively. It also provides the output sign, which is assigned to *out\_s*[31]. The components of the output select unit are shown in the datapath

diagram in Figure 4-3.

A significand of all zeros is selected if the output is zero or infinity. A significand of 0x7FFFFFFF is selected if the output is a SNaN or the maximum possible representable value. The significand  $f_1$  is selected if only one input was zero or the output is a QNaN. If the output is a QNaN, then  $out\_s[22:0]$ , the highest bit of the significand field, is set to 1. The output  $f\_norm\_inc$  is selected otherwise. If the rounding mode is RNE, and  $frac\_bits=10$ , then  $out\_s[0]$ , the lowest bit of the significand field, is pulled low.

The zero exponent is selected if the output is zero. An exponent of 0xFF is selected if the output is NaN or infinity. An exponent of 0xFE is selected if the output is the largest possible finite number. If the significand sum overflowed in the rounding stage and none of the inputs were zero, then the exponent becomes  $exp\_norm\_inc$ . Otherwise the exponent is  $exp\_norm$ .

## 4.3 Dual Path Adder

The block diagram for the dual path adder is shown in Figure 4-4. The control unit calculates all exceptions and the output sign *seff*, as well as various control signals for the datapath. The datapath contains an exponent and significand decoding unit, a swap unit, the N-path, the R-path and an output select unit.

The exponent and significand decoder is identical to that described in the single path adder. The swap unit is also identical to that described in the single path adder, with the exception that it produces an extra signal *large\_diff*, which indicates if the exponent difference is greater than 1. This is produced by ORing together the top 7 bits of *pre\_exp\_shift*.

The pipeline registers after the swap unit only pass inputs to the N-path and R-path modules if the enable signals *is\_N* and *is\_R* respectively, are high. Control unit signals that are inputs to the N-path and R-path modules are also registered and not propagated to the modules unless the appropriate enable signal is high.

### 4.3.1 N-path

A datapath diagram for the N-path module is shown in Figure 4-5. A summary of relevant signals appears in Table 4.5.

Signal	Meaning	Calculation
<i>f_sum_N</i>	Significand sum	$\{1, f_2[22:0], 0\} - f_2[24:0]$
<i>incr_N</i>	Number of leading zeros in <i>f_sum_N</i>	CLZ unit
<i>f_norm_N</i>	Normalized result	Shift <i>f_sum_N</i> left <i>incr</i> places
<i>f_norm_inc_N</i>	Rounded result	$f\_sum\_N + inject\_N$
<i>f_res_N</i>	Output significand	$\{f\_norm\_inc\_N[23:2], !sig\_sel\_N \& f\_norm\_inc\_N[1]\}$
<i>exp_norm_inc_N</i>	Incremented Normalized Exponent	$exp\_norm\_N + 1$
<i>SIG_OVF_N</i>	Significand Addition Overflow	$f\_norm\_inc\_N[25]$
<i>inject_N</i>	Rounding Increment	From Control Unit
<i>exp_n1_N</i>	$exp\_norm\_N < 0$	Byproduct of $e_1 - incr$
<i>exp_v1_N</i>	$exp\_norm\_N > 0xFF$	Byproduct of $e_1 - incr$
<i>exp_norm_N</i>	Normalized exponent	$e_1 - incr$
<i>frac_bits_N</i>	Control unit input for rounding	$f\_norm\_N[0]$

Table 4.5: N-path Signal Summary

The N-path module performs the functions of the sticky shift unit, significand

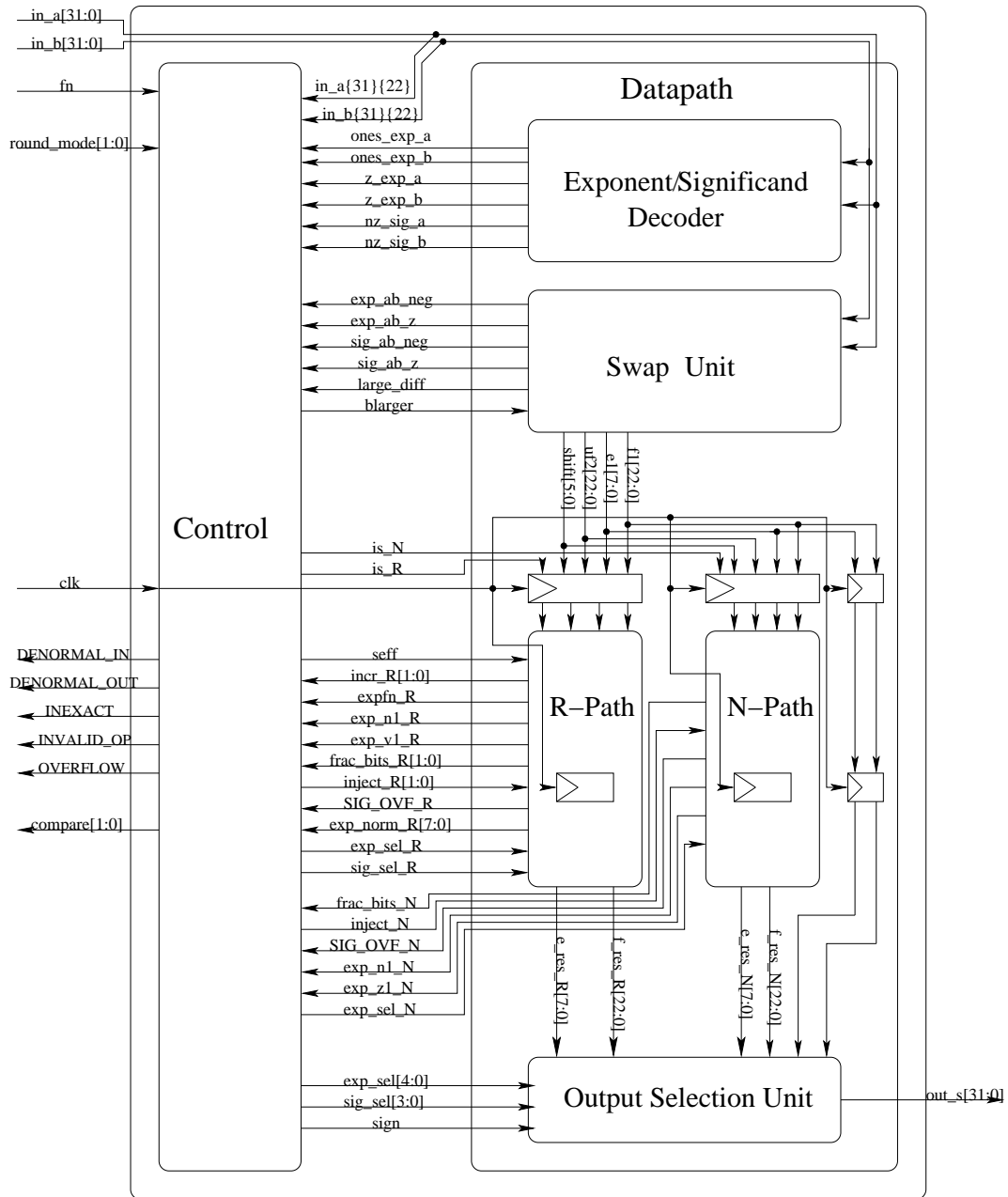
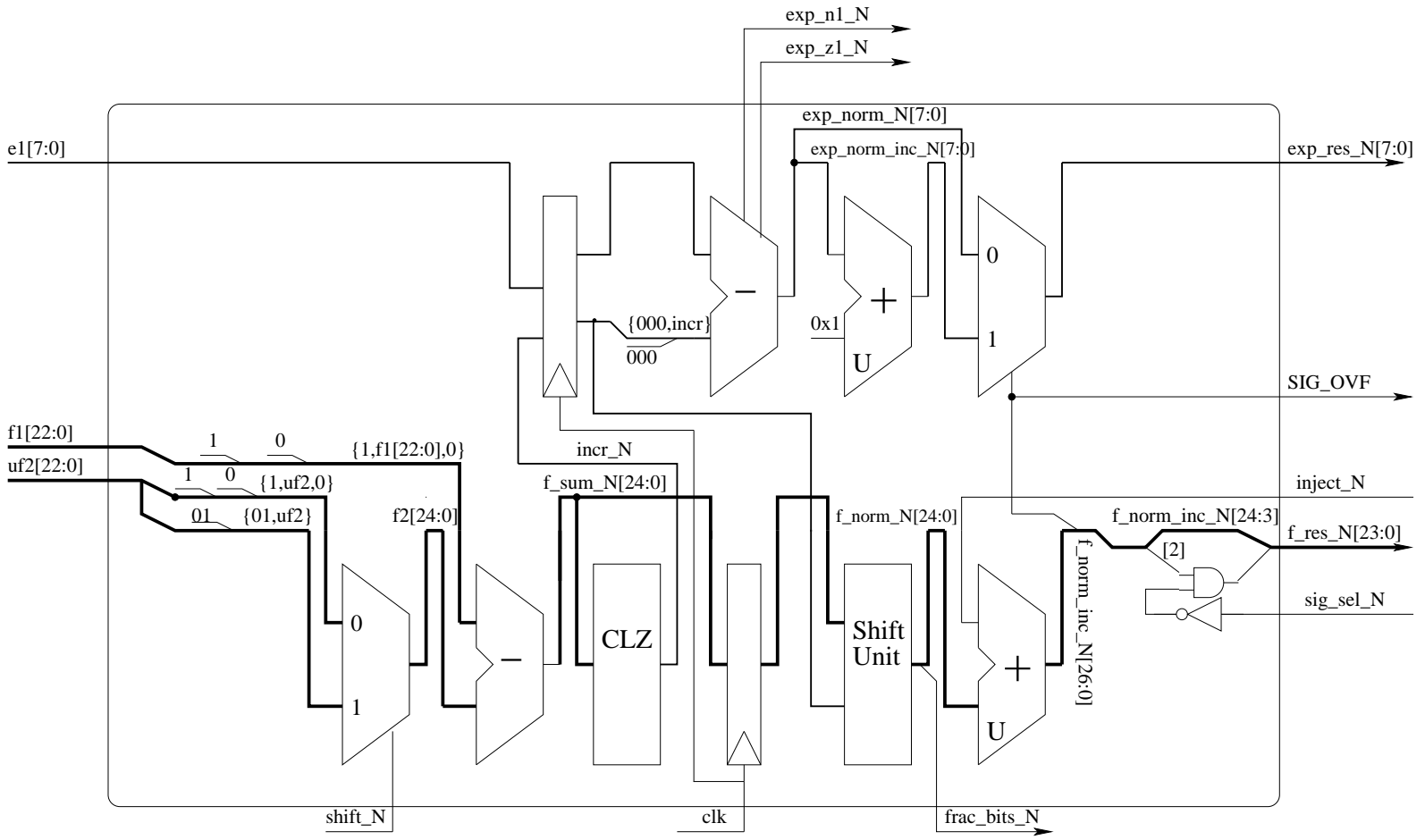


Figure 4-4: Dual Path Adder: Block Diagram

Figure 4-5: Dual Path Adder: N-path



adder, normalization unit and round unit from the datapath of the single path adder. The implementation of these units inside the N-path module differs from the single path adder in the following ways:

Since the exponent difference of the inputs is no greater than one, a 2-way mux suffices to align  $uf_2$  with  $f_1$ , and only a single extra guard bit needs to be concatenated with  $uf_2$  prior to shifting. This eliminates the sticky shift unit.

Since the significand operation is a subtraction, the result  $f\_sum\_N$  is only 25 bits wide and contains no overflow bit. The number of leading zeros  $incr\_N$  thus represents both the number of places by which  $f\_sum\_N$  must be shifted to normalize it and the increment by which the exponent  $e_1$  must be decremented. This eliminates an 8-bit adder and 6-bit mux. The subtraction cannot overflow so only  $exp\_z1\_N$  and  $exp\_n1\_N$  are output to the control unit.

Since there is a single guard bit,  $frac\_bits\_N$  and  $injection\_N$  are 1-bit signals. After rounding by injection the output significand  $f\_res\_N$  is equal to  $f\_norm\_inc\_N[23:1]$ .  $sig\_sel\_N$  pulls down  $f\_res\_N[0]$  when in RNE mode. The signal  $SIG\_OVF\_N$  is set to  $f\_norm\_inc\_N[25]$  and is used to mux either  $exp\_norm\_N$  or  $exp\_norm\_inc\_N$  from  $exp\_res\_N$ .

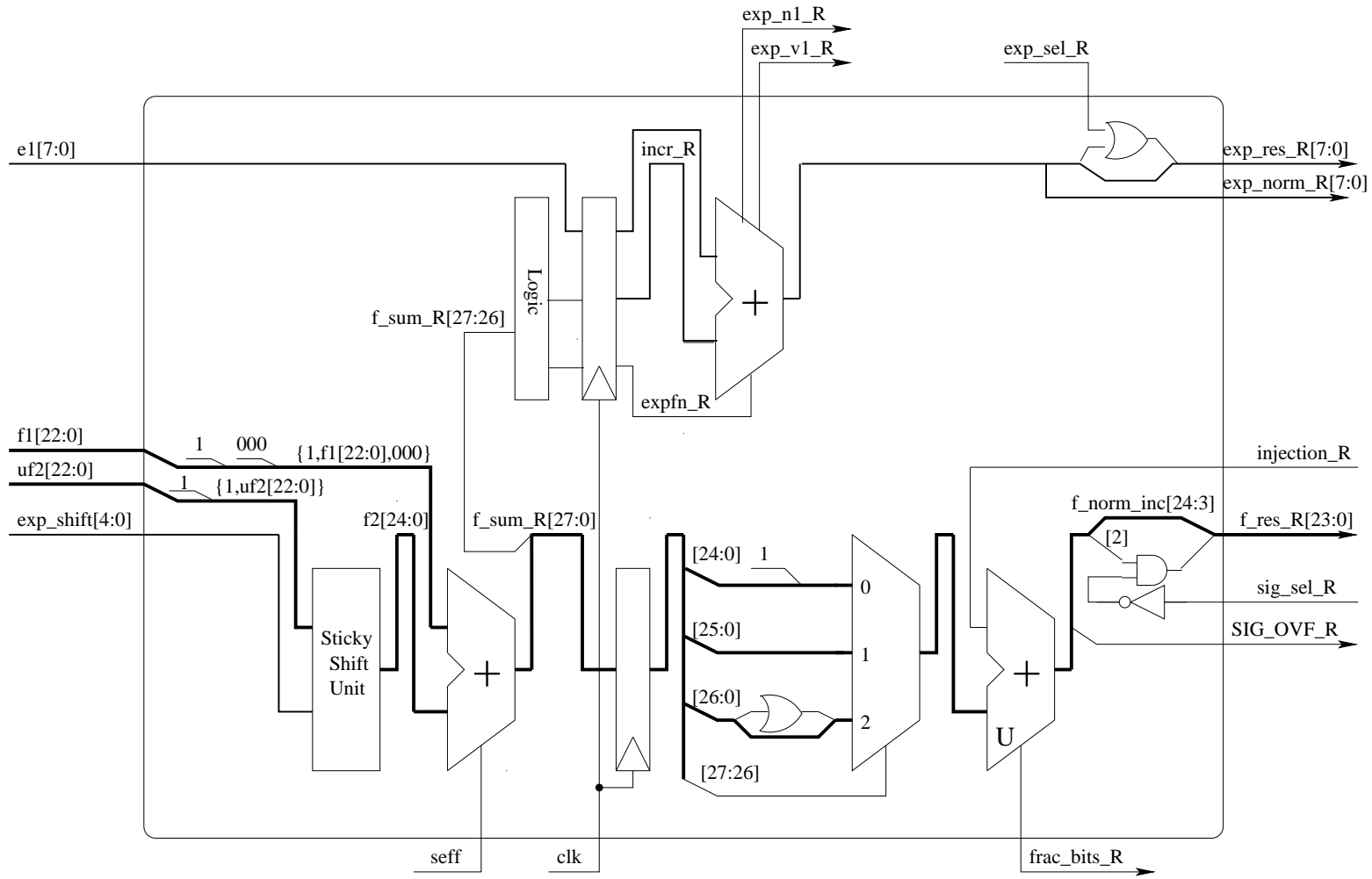
### 4.3.2 R-path

A datapath diagram for the R-path module is shown in Figure 4-6. A summary of relevant signals appears in Table 4.7. The R-path module performs the same functions as the N-path module from the datapath of the single path adder. The implementation of these units inside the R-path module differs from the single path adder in the following ways:

Since if the operation is a subtraction, the exponent difference is greater than one, the number of places by which the 28-bit significand sum  $f\_sum\_R$  must be shifted can be no greater than 2. Since there are no more than 3 leading zeros, there are only 4 possible values for the final exponent field:  $e_1 - 1$ ,  $e_1$ ,  $e_1 + 1$  and  $e_1 + 2$ .

Table 4.6 shows the increments added to  $exp_1$  to calculate  $exp\_norm\_inc$  and  $exp\_norm$  in the single path adder. The R-path adds the increment in the third

Figure 4-6: Dual Path Adder: R-path



<i>incr_R</i>	Single Path Increment		Dual Path Incr.
	<i>exp_norm</i>	<i>exp_norm_inc</i>	<i>exp_norm_R</i>
0	+1	+2	+2
1	+0	+1	+1
2	-1	+0	-1

Table 4.6: Exponent Increments in the R-Path and Single-Path Datapath

column of the table to  $e_1$  to calculate  $exp\_norm\_R$ . The desired exponent result will be identical to either  $e_1$  or the calculated value of  $exp\_norm\_R$  for the 7 highest bits. It is thus possible to generate the appropriate exponent result by selecting either  $e_1$  or  $exp\_norm\_R$  and modifying the lowest bit. This logic is computed by the control unit and is dependent on  $SIG\_OVF\_R$ ,  $incr\_R$ ,  $exp\_norm\_R[0]$  and  $e_1[0]$ . This scheme eliminates 2 8-bit adders, and a 6 and 8-bit mux.

The exponent operation  $expfn$  and number of leading zeros  $incr$  are computed by simple logic operations as shown in Table 4.7. A 3-way mux performs the normalization shift, with  $f\_sum[27:26]$  as select inputs. This eliminates the CLZ module and a shifter. After rounding by injection the output significand  $f\_res$  is equal to  $f\_norm\_inc[24:2]$ .  $sig\_sel$  pulls down  $f\_res\_R[0]$  when in RNE mode.  $SIG\_OVF\_R$  is equal to  $f\_norm\_inc\_R[26]$ .

Signal	Meaning	Calculation
$f\_sum\_R$	Significand sum	$\{1, f_2[22:0], 000\} - f_2[26:0]$
$incr\_R$	Number of leading zeros in $f\_sum\_R$	$f\_sum\_R[27], !f\_sum\_R[27]$
$expfn\_R$	Exponent Operation, Subtraction=1	NOR of $f\_sum\_R[27:26]$
$f\_norm$	Normalized result	Shift $f\_sum\_R$ with 3-way mux
$f\_norm\_inc\_R$	Rounded result	$f\_sum\_R + inject\_R$
$f\_res\_R$	Output significand	$\{f\_norm\_inc\_R[24:3], !sig\_sel\_R \& f\_norm\_inc\_R[2]\}$
$exp\_res\_R$	Output exponent	$\{exp\_norm\_R[7:1], exp\_norm\_R[0] \& exp\_sel\_R\}$
$SIG\_OVF\_R$	Significand Addition Overflow	$f\_norm\_inc\_R[26]$
$inject\_R$	Rounding Increment	From Control Unit
$exp\_n1\_R$	$exp\_norm\_R < 0$	Byproduct of $e_1 - incr$
$exp\_v1\_R$	$exp\_norm\_R > 0xFF$	Byproduct of $e_1 - incr\_R$
$exp\_norm\_R$	Normalized exponent	$e_1 - incr\_R$
$frac\_bits\_R$	Control unit input for rounding	$\{f\_norm[2], \text{OR of } f\_norm[1:0]\}$

Table 4.7: R-path Signal Summary

### 4.3.3 Output Select

The output selection unit is similar to that of the single path adder. The control unit provides the selection signals *exp\_sel* and *sig\_sel* and the output sign. The significant selection logic differs from that of the single path only in that instead of signal *f\_norm\_inc*, it appropriately chooses either *f\_res\_N* or *f\_res\_R*, and that the logic to pull down *out[0]* in RNE mode has been moved into the R-path and N-path modules.

The exponent is selected from among 0x00, 0xFE, 0xFF, *exp\_res\_N*, *exp\_res\_R* and  $e_1$ . The selection criteria for 0x00, 0xFE, and 0xFF are identical to those in the single path adder. *exp\_res\_N* is selected as the output when the N-path is enabled. When the R-path is enabled, either  $e_1$  or *exp\_res\_R* may be selected, as described previously.



# Chapter 5

## Implementation and Testing

### 5.1 Test Strategy

The single path and dual path adders were both implemented in the Verilog hardware description language. Softfloat is a software implementation of the IEEE floating-point standard written in C [1]. Testfloat is a complementary program that tests whether a floating-point implementation conforms to the IEEE standard [2]. The test harness makes use of the addition test vectors provided with Testfloat as well as C functions in the Testfloat and Softfloat libraries.

A C++ library interfaces directly with the Testfloat and Softfloat libraries, providing wrapper functions to retrieve test vectors and use the Softfloat addition operation. The Softfloat implementation of addition handles denormalized numbers, so a wrapper function masks the inputs and outputs of the Softfloat function call to flush denormalized numbers to zero. This wrapper function also raises the Input Denormal and Output Denormal exceptions. A PLI Interface implements the mapping between the verilog function calls and the wrapper functions. The Verilog test code instantiates the adder under test and calls the appropriate PLI function to get the input operands and results. It compares the output floating-point number and exceptions from the adder module and the Softfloat function call, printing out discrepancies to the terminal.

The single and dual path adders are logically equivalent. They pass all the Test-

float test cases. With the modification to flush denormalized numbers to zero, there is only one difference in the output of the adder modules and the Softfloat implementation. If both inputs are QNaNs, the standard requires that one of them be output, and sometimes our adder implementation does not choose the same NaN to output as Softfloat.

## 5.2 Toolflow

CAD tools from both Cadence and Synopsis were used to synthesize the design to the TSMC's Artisan 0.18 $\mu m$  library. Figure 5-1 illustrates the steps involved.

Cadence Physically Knowledgeable Synthesis (PKS) takes in Verilog RTL and performs synthesis and place and route. It outputs a netlist of the synthesized design as a Verilog file and the placement information as a DEF file. PKS was given constraints on area and time using a script file

A Perl script takes the DEF file containing placement information from PKS and replaces Metal 1 layers with Metal 2 or Metal 3, outputting a new DEF file.

Silicon Ensemble takes in the altered DEF file, and routes all signals, including power and ground. It outputs the layout as a GDSII file and the routing and placement information in a DEF file.

Assura takes in the synthesized Verilog netlist, the GDSII layout of the circuit and leaf cells, a DRC techfile, and a Spice netlist of the standard cells. Assura performs layout vs schematic verification (LVS) and produces a Spice file of the design with extracted parasitics.

Nanosim takes in the design Spice file, a set of test vectors and a Spice model. It simulates the Spice design with the input test vectors and checks the results against the output test vectors. The log file contains the results of the simulation as well as power and timing reports.

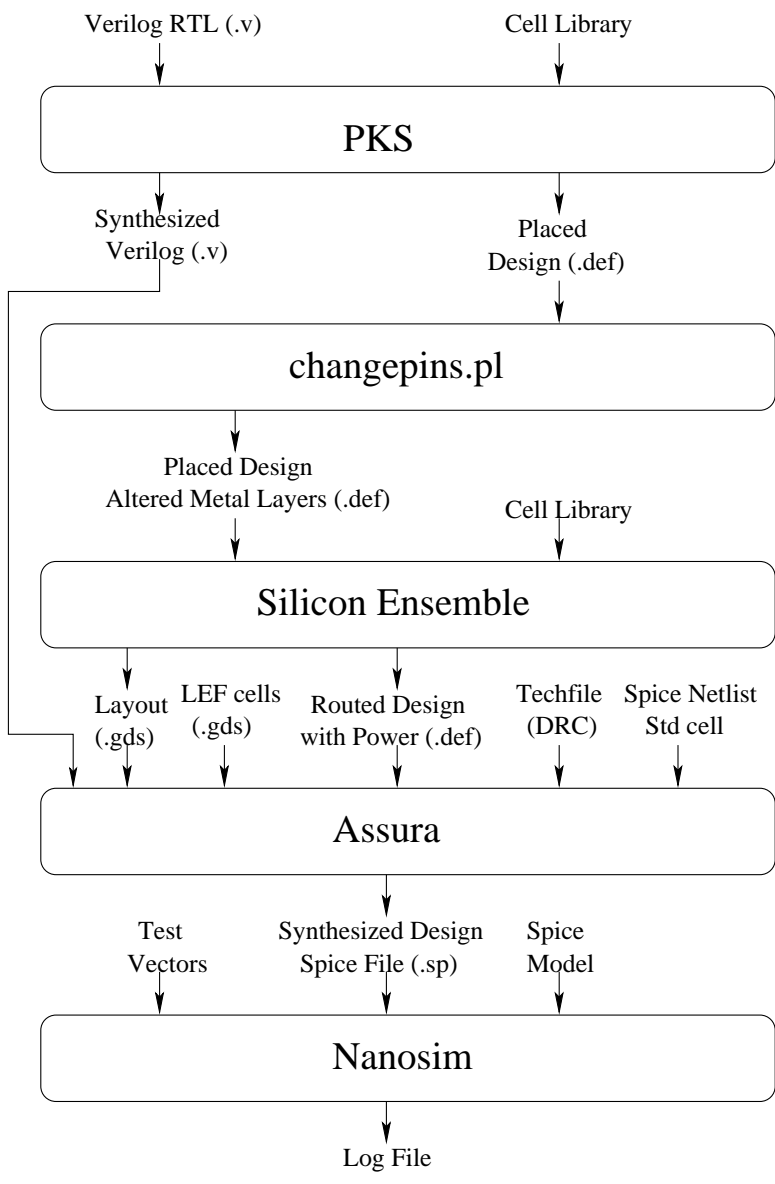


Figure 5-1: Synthesis Tool Flow

### 5.3 Pipelining the Dual Path Adder

The timing goal was a cycle time of 5ns and either 3 or 4 pipeline stages. An un-pipelined Verilog model was run through the synthesis, with timing budgets of 15ns and 20ns. In both cases, the critical path ran through the R-path. The delay along the critical path for the dual path adder is summarized in Table 5.2.

Operation	Output Arrival Time (ns)		Unit Delay (ns)	
	15ns	20ns	15ns	20ns
Input Register	0.21	0.38	0.21	0.38
Significand Subtraction	2.64	2.75	2.43	2.37
Blarger Control Logic	3.04	3.29	0.40	0.54
Mux to Select f1	3.34		0.30	
Mux to select uf2		3.63		0.34
exp_shift	3.51		0.17	
R-Shift	5.39	5.39	1.88	1.76
R-Significand Add	8.85	9.97	3.46	4.58
R-mux	9.56	10.61	0.71	0.64
R-or	9.68	10.73	0.12	0.12
Round Significand	12.35	14.31	2.67	3.58
Ctl Logic	12.85	15.39	0.50	1.08
Significand Mux	13.84	16.29	0.99	0.90

Table 5.2: Dual Path Adder Critical Path Delay

The total cell area when synthesized with a timing budget of 15ns was 29189.16  $\mu^2$ . The cell area when synthesized with a timing budget of 20ns was 26910.58  $\mu^2$ . The slack time when run with a timing budget of 15ns was 1.06 ns (output delay estimated at 0.1ns) , and the total delay when run with a timing budget of 20ns was only 1.29 ns over 15ns. Therefore, a 3 stage pipeline with a 5ns cycle time should be achievable without a polynomial growth in area.

The first set of pipeline registers was placed directly after the swap unit, as by the time the outputs from the swap unit were ready, the control logic would have figured out if the N-path was in use or the R-path. The second set of pipeline registers was placed after the CLZ module in the N-path and after significand addition in the R-path. The adder was then resynthesized.

## 5.4 Pipelining the Single Path Adder

The timing goals for the single path adder were a cycle time of 5ns and either 3 or 4 pipeline stages. An unpipelined verilog model was run through the synthesis, with timing budgets of 15ns and 20ns. The delay along the critical path for the single path adder is summarized in Table 5.4.

Operation	Output Arrival Time (ns)		Unit Delay (ns)	
	15ns	20ns	15ns	20ns
Input Register	0.23	0.31	0.23	0.31
Significand Subtraction	2.47	2.68	2.24	2.37
Blarger Control Logic	2.80	3.23	0.33	0.35
Muxing of $uf_2$	3.26	3.77	0.46	0.54
Sticky Shift	4.75	5.40	1.49	1.43
Significand Addition	8.07	9.23	3.32	3.83
CLZ	10.34	11.71	2.27	2.48
Normalization Shift	10.66	12.47	0.32	0.76
Significand Rounding	13.05	15.17	2.45	2.70
Control Logic	13.70	16.16	1.21	1.01
Significand Mux	14.56	17.20	0.78	1.04

Table 5.4: Single Path Adder Critical Path Delay

If the design is pipelined in 4 stages, at least one stage will have an estimated delay of greater than 5 ns, according to the 20ns synthesis results. However, the rest of the stages would have delays of less than 4 ns. Each additional pipeline stage adds complexity to the logic and approximately 2000  $\mu^2$  per 32-bit register. The total cell area when synthesized with a timing budget of 15ns was 26,248.25  $\mu^2$ . The cell area when synthesized with a timing budget of 20ns was 23,926.75  $\mu^2$ . At 15ns, there is still room to cut time without a polynomial growth in area. Therefore, the area of a 3 stage single path adder ought to be comparable or not much more than the area of a 4 stage single path adder. The dual path adder is also being pipelined in 3 stages, so a 3 stage single path adder will provide a more representative baseline for comparison. The single path adder was therefore pipelined in 3 stages and resynthesized.



# Chapter 6

## Results

Three adder configurations are examined. The first configuration is the single path adder, which is pipelined into 3 stages in order to achieve comparable performance to the dual path adders. The second configuration is a 3 stage dual path adder with both paths operating at all times. The third is a 3 stage dual path adder with inputs to the nonconducting path disabled. All three adders have a cycle time of 5ns and hence run at 200 MHz.

### 6.1 Energy Measurement Methodology

Nanosim reports average and RMS block power for individual blocks during a simulation. The desired measurement is energy dissipated per operation, which is estimated from the average power measurements from Nanosim using the following formula:

$$\frac{Energy}{Operation} = \mu W \times \frac{5ns}{Operation}$$

## 6.2 Energy Dissipation Results

Each adder was tested in Nanosim on a suite of 100 random test vectors that were pulled from testfloat. The results of the simulation and total cell area for the three configurations are displayed in Table 6.1.

Adder Model	Cell Area ( $\mu^2$ )	Average Power ( $\mu\text{W}$ )	Energy/Operation (pJ)
Single Path	31188.33	8884	44
Dual Path (no inhibit)	44696.84	13970	69
Dual Path (inhibit)	47836.96	10401	52

Table 6.1: Area and Energy Dissipation Results

The single path adder uses 63.6% of the power of the dual path no inhibit adder and 85.4% of the power of the dual path inhibit adder. If using a dual path adder, inhibiting inputs leads to a 25.5% reduction in power.

# Chapter 7

## Conclusion

The dual path inhibit adder demonstrated the viability of disabling inputs to unused modules to reduce switching energy. Understandably, this implementation did not come close to the 90% theoretical savings of a triple path adder architecture posited by Pillai et al.[6]. These results depend on the inputs being evenly distributed, which is certainly not the case for our 100 test cases. In addition, the triple path adder bypassed both the R and N paths when the exponent difference is greater than the width of the significand. Under the worst case scenario assumption that the bypass path is never used, it is proposed that the triple path adder would still use 50% of the energy of a different implementation.

The triple path adder implemented rounding and addition in one step, as well as pseudo Leading Zero Anticipation (LZA) logic, neither of which were implemented by the dual path adder. The adder model used as its baseline is assumed to implement full LZA and add significands with a carry-select adder. Clearly, the triple path and dual path adder architectures are too different to expect similar performance improvements. In addition, the triple path adder is a theoretical design that has not been implemented and measured. It is assumed that a module which is not enabled does not dissipate power (neglecting static power dissipation) when calculating power consumption.

Our real world implementation of the dual path inhibit adder is most productively evaluated by comparing it with other adders implemented with the same process.

The dual path inhibit adder uses 74.5% of the energy of an ordinary dual path adder. However, the single path adder offers superior performance to both dual path adders in every respect. This adder is smaller in area, meets the same timing goals and uses less energy than either version of the dual path adder. It is expected that a single path adder design is smaller and slower than a dual path adder design. Even though the N-path and R-path modules of the dual path adder are shorter than the single path adder datapath, when the extra control logic and muxing is taken into account, the total area of active modules in the dual path adder is larger than the total area of the single path adder on most cycles. In addition, the inactive path which is not switching, is still dissipating static power. Therefore it is unsurprising that the single path adder uses less energy than the dual path adders.

It is worth noting that more effort was required for the single path adder to meet timing goals than the dual path adders. Assuming the output port delay to be 0.1ns, the slack time of the unpipelined single path adder when run with a timing budget of 15ns was 0.34 ns. The slack time of the unpipelined dual path adder was 1.06 ns. It is possible that were both adders resynthesized with tighter timing constraints, the single path adder would fail to meet timing at a larger cycle time and/or the area of the single path adder would approach that of the dual path adder. Once the floating-point adder is synthesized to target the custom datapath cells, it will have to meet a 2.5ns cycle time spec. Given that the inhibit enabled dual path adder only uses 17% more energy per operation than the single path adder with the current time spec, it is possible that it may outpace the single path adder when synthesized with the custom cells and a shorter cycle time. This design space would be worth exploring in the future.

Another design space worthy of exploration involves the rearrangement of the dual path adder datapath to allow more hardware to be inhibited. The significant subtract in the first stage of the pipeline is not really necessary, as the inputs can be swapped based solely on the relative sizes of the exponents. In the R-path, the significant size is irrelevant. In the N-path, both  $f\_sum\_1=f_1 - f_2$  and  $f\_sum\_2=f_2 - f_1$  could be calculated in parallel and muxed to produce  $f\_sum$ . The time taken by the extra

mux stage would be offset by the time gained by not computing a 23-bit subtraction. The design would have to be repipelined, as the current first stage would become much faster. A portion of the N and R path logic would be brought into the first stage. If this logic could be inhibited midcycle, perhaps through the use of latches, a larger fraction of the module would not be switching on a given cycle, reducing power dissipation.

If the SCALE chip were to be fabricated tomorrow and the desired speed was 200 MHz, the correct floating-point adder design to use would be the single path adder. However, if timing is to be pushed, both adders should be resynthesized to determine which dissipates less power at higher speeds. Time and interest permitting, the recommended partial redesign of the dual path adder will decrease latency and could decrease power consumption.



# Bibliography

- [1] John Hauser. SoftFloat. <http://www.jhauser.us/arithmatic/TestFloat.html>, 2002.
- [2] John Hauser. TestFloat. <http://www.jhauser.us/arithmatic/SoftFloat.html>, 2002.
- [3] *IEEE Standard for Binary Floating Point Arithmetic*. ANSI/IEEE Std. 754-1985, August 1985.
- [4] Ronny Krashinsky, Chris Batten, and Krste Asanovic. The SCALE Architecture.
- [5] MIPS Technologies Inc, Mountain View, CA. *MIPS32 Architecture For Programmers*, September 2002.
- [6] R.V.K Pillai, D. Al-Khalili, and A.J. Al-Khalili. A Low Power Approach to Floating Point Adder Design. In *Proceedings of ICCD '97.*, Computer Design: VLSI in Computers and Processors, pages 178–185, Austin, TX, October 1997. ICCD.
- [7] Nhon Quach, Naofumi Takagi, and Michael Flynn. On Fast IEEE Rounding. Technical report, Stanford University, January 1991.
- [8] Peter-M. Seidel and Guy Even. How many logic levels does floating-point addition require? In *Computer Design: VLSI in Computers and Processors*, pages 142–149, Austin, TX, October 1998. ICCD.

- [9] Peter-Michael Seidel and Guy Even. On the Design of Fast IEEE Floating-Point Adders. In *Proceedings of the 15th IEEE Symposium on Computer Arithmetic*, pages 184–194, Vail, CO, June 2001. IEEE.
- [10] Johnathan Ying Fang Tong, David Nagle, and Rob A. Rutenbar. Reducing Power by Optimizing the Necessary Precision/Range of Floating-Point Arithmetic. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 8(3):273–286, June 2000.