On the Design of IEEE Compliant Floating-Point Units and Their Quantitative Analysis

Dissertation
zur Erlangung des Grades
Doktor der Ingenieurwissenschaften
(Dr.-Ing.) der Technischen Fakultät
im Fachbereich Informatik
der Universität des Saarlandes
vorgelegt
von

Peter-Michael Seidel

Saarbrücken 1999

Dekan:	Prof. Dr. Wolfgang Paul
Erster Berichterstatter:	Prof. Dr. Wolfgang Paul
Zweiter Berichterstatter:	Priv.-Doz. Dr. Silvia Müller
Tag des Kolloquiums:	14. Januar 2000

Abstract

This thesis addresses the question of which are the important issues in the design of a high-speed floating-point unit (FPU) that is fully compliant with the IEEE floating-point standard 754-1985 [19]. There are a few choices that need to be made when designing an IEEE compliant FPU, among them: the internal representation of floating-point numbers, the rounding algorithms, handling of denormal results, usage of the same rounding hardware for different units (e.g. adder, multiplier, divider), and the implementations of the adder, the multiplier and the divider. These choices influence both the cost and the performance of the FPU. Nevertheless, these issues have not been discussed in the open literature todate. This work begins to fill this gap by designing, analyzing and comparing 18 different IEEE compliant FPU implementations, that consider design options regarding: (a) the internal representation of floating-point numbers; (b) the rounding algorithms; (c) sharing of a rounding unit, the implementation of gradual step rounding or the implementation of dedicated rounding units for each functional unit; (d) the implementation of the floating-point multiplier; and (e) the implementation of the floating-point divider. The presented FPU designs make also use of the following innovations, that were developed in the context of this work: (a) a fast implementation of variable position rounding integrated into a FP multiplier [37]; (b) to the best of our knowledge the fastest integrated FP addition and rounding algorithm published todate [40], (c) the fastest FP multiplication rounding algorithm published todate [11, 12] and (d) the fastest linear reciprocal approximation implementation published todate. [36, 39]; (e) an efficient integration of single and double precision rounding [9]; (f) a Booth encoded adder-tree with an improved cost formula [30].

All the FPUs designed in this work are fully compliant with the IEEE standard for all implemented operations, support both single and double precision, and deal with denormal values and special cases in hardware. Because to design an IEEE compliant FPU is a complex and error-prone task, all the FPU designs are specified in full detail at gate level and the correctness of the FPU designs (in particular the compliance with the IEEE standard) is proven. The proposed FPU implementations are analyed and compared regarding the hardware cost, the cycle time and the performance that they achieve on traces of the SPECfp92 benchmark suite [17] integrated into a pipelined RISC processor from [23]. In this quantitative analysis [38] it is demonstrated that the choice of the rounding architecture in the FPU has a larger impact on the performance of the microprocessor than the choice of the FP multiplication or the FP division implementation. In comparison to this the impact of the rounding architecture choice on the cost is relatively small. The rounding architecture that uses dedicated rounding units provides the best performance with only small additional cost, so that this rounding architecture seems to be the best choice in floating-point implementations. The fast implementation of this rounding architecture is only made possible by the fast variable position rounding implementation for multipliers from [37]. This underlines the importance of this technique.

Kurzzusammenfassung

In dieser Arbeit wird der Frage nachgegangen, welches die wichtigsten Designentscheidungen bei der Implementierung einer schnellen Gleitkommaeinheit (FPU), die dem IEEE Standard 754-1985 [19] genügt, sind. Es gibt verschiedene Entscheidungen, die beim Entwurf einer IEEE konformen FPU getroffen werden müssen, darunter: die internen Darstellungen der Gleitkomma- (FP) Zahlen, die Rundungsalgorithmen, die Art der Behandlung von denormalisierten Ergebnissen, die Mehrfachverwendung von Teilen der Hardware, wie z.B. die Benutzung derselben Rundungshardware für verschiedene Einheiten, und die Implementierungen des FP Addierers, des FP Multiplizierers und des FP Dividierers. Diese Entscheidungen beeinflussen sowohl die Kosten alsauch die Leistung der FPU. Nichtsdestotrotz wurden diese Entscheidungen bislang nicht in der Literatur diskutiert. Die vorliegende Arbeit setzt in dieser Lücke an. Es werden 18 unterschiedliche FPUs vorgestellt, analysiert und verglichen, die Optionen zu den folgenden Entscheidungen betrachten: (a) interne Darstellung der FP Zahlen; (b) Rundungsalgorithmen; (c) Gemeinsame Nutzung einer allgemeinen Rundungseinheit, Aufteilen des Rundens in mehrere Schritte und gemeinsame Realisierung einer Teilmenge dieser Schritte oder vollständige eigene Implementierung des Rundens für jede Funktionseinheit; (d) Implementierung des FP Multiplizierers; (e) Implementierung des FP Dividierers. Die vorgestellten FPU Designs benutzen darüberhinaus folgende Neuerungen, die im Rahmen dieser Arbeit entstanden sind: (a) eine schnelle Rundungsimplementierung für den FP Multiplizierer mit variabler Rundungsposition [37]; (b) nach unserem besten Wissen den bisher schnellsten publizierten Algorithmus zum Addieren und Runden von FP Zahlen [40], (c) den bisher schnellsten publizierten Algorithmus zum Runden bei der FP Multiplikation [11, 12] und (d) die bisher schnellste publizierte Implementierung einer linearen Approximation von Reziproken [36, 39]; (e) eine effiziente Integration des Rundens in single precision und double precision [9]; (f) einen Booth-Multiplizierer mit verringerten Kosten [30].

Alle entworfenen FPUs sind für alle implementierten Operationen vollständig konform zum IEEE FP Standard 754, unterstützen sowohl single alsauch double precision Zahlen, und behandeln selbst denormalisierte Ergebnisse und Spezialfälle in Hardware. Weil der Entwurf von IEEE konformen FPUs eine komplexe und fehleranfällige Aufgabe ist, werden sämtliche entworfenen FPUs detailiert auf Gatterebene spezifiziert und ihre Korrektheit (insbesondere die Konformität zum IEEE FP Standard 754) bewiesen. Die vorgestellten FPU Implementierungen werden bezüglich der Hardwarekosten, der Zykluszeit und der Leistung, die sie integriert in einen gepipelinten RISC Processor aus [23] auf Traces der SPECfp92 Benchmark Suite erbringen, analysiert und verglichen. In dieser quantitativen Analyse (siehe auch [38]) wird demonstriert, daß die Auswahl der Rundungs-Architektur einer FPU einen größeren Einfluß auf die Prozessorleistung hat als die Auswahl der Implementierung der FP Multiplikation oder der FP Division. Im Gegensatz dazu ist der Einfluß der Auswahl einer Rundungs-Architektur der FPU auf die Hardwarekosten vergleichsweise gering. Die Rundungs-Architektur, die vollständige eigene Rundungsimplementierungen für jede Funktionseinheit benutzt, liefert bei weitem die beste Leistung und ist lediglich geringfügig teurer als Varianten mit anderen Rundungs-Architekturen. Demzufolge scheint diese Rundungs-Architektur die beste Wahl in FP Implementierungen zu sein. Die schnelle Implementierung dieser Rundungs-Architektur wurde erst durch die schnelle Rundungsimplementierung für FP Multiplizierer mit variabler Rundungsposition nach [37] ermöglicht. Das unterstreicht die Bedeutung dieser Technik.

Extended Abstract

The importance of floating-point operations is increasing in recent graphic and multimedia applications. Therefore, each modern microprocessor has to contain at least one floating-point unit, that supports and accelerates the floating-point computations. To achieve a well defined behavior during the computations, the floating-point support should be conform with the IEEE floating-point standard 754-1985 [19].

Despite the high demand for floating-point hardware implementations, an answer to the question, how to design a fast IEEE compliant FP unit, rarely can be found in the open literature. Moreover, there are several choices that need to be made when designing an IEEE compliant FPU, among them: the internal representation of floating-point numbers, the rounding algorithms, handling of denormal results, usage of the same rounding hardware for different units (e.g. adder, multiplier, divider), and the implementations of the adder, the multiplier and the divider. These choices influence both the cost and the performance of the FPU. Nevertheless, these issues have not been discussed in the open literature todate. In contrast to this lack of publications about the implementation of fully IEEE compliant FP operations or fully IEEE compliant FPUs, there are many published implementations of specific floating-point operations for the case of normalized operands in a specific precision, e.g. [9, 26, 27, 32, 40, 43, 44], and these implementations are highly optimized for speed. Therefore, it is an important question, how to integrate the implementations of the different FP operations for normalized operands into a floating-point unit that supports more than one precision, denormalized numbers and special value results. This mainly includes the questions of which internal FP representations should be used in a FP unit and how the microarchitecture of a FP unit could be organized.

This work starts to fill these gaps in the open literature and to find answers to these open questions. For this purpose, 18 different implementations of a floating-point unit are designed, quantitatively analyzed and compared in this thesis. All proposed FP designs provide full compliance with the IEEE FP standard 754-1985 for all implemented operations, support both single precision and double precision operands and also consider denormalized numbers, special values, exponent wrapping and floating-point exceptions in hardware. The core of this work is the design and the comparison of three different FPU microarchitectures that consider the following three options:

(I) the use of a shared general rounder for all functional units; A basic specification of such a rounder was first described in [10]. Thereafter, this rounder was implemented by our group, resulting in a version that will be included in [23], where also a rigorous proof of the compliance with the IEEE rounding definition will be found. This rounder was further optimized to be included in this thesis.

(II) a gradual rounding implementation in two steps, a first rounding step within the functional units assuming the case of a normalized double precision result and a second rounding step within a shared gradual rounder that fixes the result for all

other cases; For the integrated rounding in the functional units assuming normalized, double precision operands and results, several algorithms from literature could be used. The implementation of the gradual rounder is based on the theory from [21] about gradual rounding. This rounding technique is integrated in this thesis for full IEEE compliant rounding including the handling of denormalized results, special values, exceptions and exponent wrapping.

(III) the use of separate fully IEEE compliant rounding implementations for each functional unit, each including the handling of denormalized numbers, special cases, exceptions and exponent wrapping. The implementation of this microarchitecture for a full IEEE compliant FPU with dedicated rounding implementations is completely new in this thesis. Especially the integration of a variable position rounding implementation into the multiplier, that is required to deal with denormalized multiplication results, was one of the main problems for the implementation of this microarchitecture and is one of the main innovations of this work [37].

Directly linked to the choice of the FP microarchitecture is the question of the internal floating-point representations. In this work, five different internal FP representations are defined. These are used to specify the interfaces between the functional units in detail. In addition to the consideration of the three different microarchitectures for the FP implementation, the implementations of the FP-multiplication and the FP-division are chosen among 6(2x3) variants:

- For the FP multiplication implementation a Booth encoded adder tree is used either in a full-sized version that is able to compute double precision and single precision multiplications in one iteration or in a half-sized version that computes double precision multiplications in two iterations and single precision multiplications in one iteration.

- For the FP division implementation, we consider three different implementations of the Newton-Raphson iteration with an initial reciprocal approximation with an absolute approximation error bounded by 2^{-8}, 2^{-16}, and 2^{-28}, respectively. For this initial reciprocal approximation a fast implementation of a linear approximation formula using partial compressions was developed [36, 39].

In addition to the different design choices for the internal FP representations, the rounding microarchitecture and the choice of the FP multiplication and the FP division implementation, the presented FPU designs make also use of the following innovations, that were developed in the context of this work:

(a) a fast implementation of variable position rounding for FP multiplication [37];

(b) to the best of our knowledge the fastest integrated FP addition and rounding algorithm published todate [40],

(c) the fastest FP multiplication rounding algorithm published todate [11, 12] and

(d) the fastest linear reciprocal approximation implementation published todate. [36, 39];

(e) an efficient integration of single and double precision rounding for FP multiplication [9];

(f) a Booth encoded adder-tree with an improved cost formula [30].

The proposed FPUs are quantitatively analyzed regarding the hardware cost, the cycle time and the performance. The hardware cost and the cycle time are measured using the formal Hardware model from [22]. The performance of the FP units is analyzed on traces of the SPECfp92 Benchmark Suite integrated into a pipelined RISC-processor from [23].

In this quantitative analysis (see also [38]) it is demonstrated that the choice of the rounding microarchitecture in the FPU has a larger impact on the performance of the microprocessor than the choice of the FP multiplication or the FP division implementation. In comparison to this the impact of the microarchitecture choice on the cost is relatively small. The microarchitecture that uses dedicated rounding units provides the best performance with only small additional cost, so that this rounding architecture seems to be the best choice in floating-point implementations.

Zusammenfassung

Floating-Point Operationen gewinnen in heutigen Grafik- und Multimedia-Anwendungen immer mehr an Bedeutung. Deshalb besitzen aktuelle Mikroprozessoren mindestens eine Floating-Point Einheit, die die Floating-point Berechnungen unterstützt und beschleunigt. Um ein wohldefiniertes Verhalten der Floating-point Berechnungen zu erhalten, sollte die Floating-point Unterstützung konform zum IEEE floating-point Standard 754-1985 [19] sein.

Trotz des großen Bedeutung von Floating-Point Implementierungen in Hardware, gibt es in der offenen Literatur nur spärliche Antworten auf die Frage, wie man eine schnelle IEEE konforme FP Einheit entwirft. Darüberhinaus gibt es verschiedene Entscheidungen, die beim Entwurf einer IEEE konformen FPU getroffen werden müssen, darunter: die Wahl der internen Darstellungen der Gleitkomma- (FP) Zahlen, die Rundungsalgorithmen, die Art der Behandlung von denormalisierten Ergebnissen, die Mehrfachverwendung von Teilen der Hardware, wie z.B. die Benutzung derselben Rundungshardware für verschiedene Einheiten, und die Implementierungen des FP Addierers, des FP Multiplizierers und des FP Dividierers. Diese Entscheidungen beeinflussen sowohl die Kosten alsauch die Leistung der FPU. Nichtsdestotrotz wurden diese Entscheidungen bislang nicht in der Literatur diskutiert.

Im Gegensatz zu diesem Mangel an Publikationen über die Implementierung von IEEE konformen FPUs, gibt es allerdings eine Reihe von publizierten Implementierungen von einzelnen Floating-point Operationen für den Fall von normalisierten Operanden in einer festgelegten Genauigkeit, z.B. [9, 26, 27, 32, 40, 43, 44], und diese Implementierungen sind in Hinblick auf ihre Geschwindigkeit optimiert. Deshalb ist es eine wichtige und interessante Frage, wie diese Implementierungen einzelner FP Operationen für normalisierte Operanden in eine FPU, die mehr als einen FP Typ unterstützt und auch die Behandlung von denormalisierten Zahlen und special values berücksichtigt, integriert werden können. Das beinhaltet hauptsächlich die Fragen, welche internen FP Zahlendarstellungen in einer FP Einheit verwendet werden sollten und wie die Architetur einer FP Einheit zu organisieren ist.

Die vorliegende Arbeit setzt in dieser Lücke an. Zu diesem Zweck werden in dieser Arbeit 18 verschiedene FP Implementierungen entworfen, quantitativ analysiert und verglichen. Alle vorgestellten FPU Entwürfe sind für die FP Operationen, die sie implementieren vollständig konform zu dem IEEE Standard 754-1985, unterstützen sowohl single precision alsauch double precision Operanden und berücksichtigen auch denormalisierte Ergebnisse, special values, Exponenten wrapping und FP exceptions in Hardware. Der Kern dieser Arbeit ist der Entwurf und der Vergleich von drei unterschiedlichen FPU Architekturen, die die folgenden Optionen betrachten:

(I) die Verwendung eines gemeinsamen allgemeinen Runders für alle Funktionseinheiten. Eine grundlegende Spezifikation eines solchen Runders wurde zuerst in [10] beschrieben.

Danach wurde dieser Runder in unserer Gruppe in einer Version implementiert, die in [23] vorgestellt werden wird. Dieser Runder wurde für die vorliegende Arbeit weiter optimiert.

(II) eine Rundungsimplementierung in zwei Schritten (gradual rounding), ein erster Rundungsschritt in den Funktionseinheiten unter der Annahme von normalisierten Ergebnissen in double precision und ein zweiter Rundungsschritt in einem gemeinsamen Gradual Rounder, der das Ergebis für alle anderen Fälle (nicht double precision oder kein normalisiertes Ergebnis) anpaßt. Für das Runden in den Funktionseinheiten unter der Annahme von normalisierten double precision Ergebnissen können unterschiedliche Algorithmen aus der offenen Literatur verwendet werden. Die Implementierung des gradual rounders basiert auf der Theorie aus [21]. Dieses Rundungsprinzip wird in der vorliegenden Arbeit für vollständig IEEE konformes Runden unter Berücksichtigung von denormalisierten Ergebnissen, special values, exceptions und Exponent wrapping integriert.

(III) die Verwendung von eigenen voll IEEE konformen Rundungsimplementierungen für jede Funktionseinheit, die jeweils eigenständig denormalisierte Ergebnisse, special values, exceptions und Exponent Wrapping gemäß dem IEEE Standard berücksichtigen. Die Implementierung dieser Architektur einer IEEE konformen FPU mit eigenständigen Rundungsimplementierungen ist vollständig neu in dieser Arbeit. Besonders die Integration des Variable Position Rundens in den Multiplizierer, das erforderlich wird, um denormalisierte Multiplikationsergebnisse behandeln zu können, ist eines der Hauptprobleme dieser FPU Architektur und damit ist die beschriebene Implementierung eine der wichtigsten Innovationen der vorliegenden Arbeit [37] .

Direkt verbunden mit der Wahl der Architektur der FPU ist die Frage nach den zu verwendenden internen FP Darstellungen. In dieser Arbeit werden fünf verschiedene interne FP Darstellungen definiert. Diese werden dann dazu verwendet um die Schnittstellen zwischen den Funktionseinheiten einfach, aber detailiert zu spezifizieren.

Zusätzlich zur Betrachtung der drei unterschiedlichen FPU Architekturen wählen wir die Implementierungen der FP Multiplikation und der FP Division unter 6(2x3) verschiedenen Varianten aus:

- Für die Implementierung der FP Multiplikation wird entweder ein Booth2 Multiplizierer vollständiger Größe verwendet, der sowohl single alsauch double precision Multiplikationen in einer Iteration berechnen kann oder es wird ein Booth2 Multiplizierer halber Größe verwendet, der single precision Multiplikationen in einer Iteration und double precision Multiplikationen in zwei Iterationen berechnet.

- Für die Implementierung der FP Division betrachten wir drei unterschiedliche Implementierungen der Newton-Raphson Iteration mit einer Startapproximation des Reziproken 1FB mit absolutem Approximationsfehler kleiner als 2^{-8}, 2^{-16} bzw. 2^{-28}. Für diese Approximation des Reziproken wurde eine schnelle Implementierung einer linearen Approximationsformel unter Verwendung einer partiellen Kompression entwickelt [36, 39].

Die vorgestellten FPU Designs benutzen darüberhinaus folgende Neuerungen, die im Rahmen dieser Arbeit entstanden sind:

(a) eine schnelle Rundungsimplementierung für den FP Multiplizierer mit variabler Rundungsposition [37];

(b) nach unserem besten Wissen den bisher schnellsten publizierten Algorithmus zum Addieren und Runden von FP Zahlen [40],

(c) den bisher schnellsten publizierten Algorithmus zum Runden bei der FP Multiplikation [11, 12] und

(d) die bisher schnellste publizierte Implementierung einer linearen Approximation von Reziproken [36, 39],

(e) eine effiziente Integration des Rundens in single precision und double precision [9];

(f) einen Booth-Multiplizierer mit verringerten Kosten [30].

Die vorgestellten FPU Implementierungen werden bezüglich der Hardwarekosten, der Zykluszeit und der Leistung, die sie integriert in einen gepipelinten RISC Processor aus [23] auf Traces der SPECfp92 Benchmark Suite erbringen, analysiert und verglichen. In dieser quantitativen Analyse (siehe auch [38]) wird gezeigt, daß die Auswahl der Rundungs-Architektur einer FPU einen größeren Einfluß auf die Prozessorleistung hat als die Auswahl der Implementierung der FP Multiplikation oder der FP Division. Im Gegensatz dazu ist der Einfluß der Auswahl einer Rundungs-Architektur der FPU auf die Hardwarekosten vergleichsweise gering. Die Rundungs-Architektur, die vollständige eigene Rundungsimplementierungen für jede Funktionseinheit benutzt, liefert bei weitem die beste Leistung und ist lediglich geringfügig teurer als Varianten mit anderen Rundungs-Architekturen. Demzufolge scheint diese Rundungs-Architektur die beste Wahl in FP Implementierungen zu sein.

Contents

Chapter 1

Introduction

The importance of floating-point operations is increasing in recent graphic and multimedia applications. Therefore, each modern microprocessor has to contain at least one floating-point unit, that supports and accelerates the floating-point computations. To achieve a well defined behavior during the computations, the floating-point support should be conform with the IEEE floating-point standard 754-1985 [19]. This IEEE specification could also be achieved by supporting parts of it in software, but for high-performance systems a hardware solution is preferable.

Despite the high demand for floating-point hardware implementations, a full answer to the question, how to design a fast IEEE compliant FP unit, rarely can be found in the open literature. Moreover, there are several choices that need to be made when designing an IEEE compliant FPU, among them: the internal representation of floating-point numbers, the rounding algorithms, handling of denormal results, usage of the same rounding hardware for different units (e.g. adder, multiplier, divider), and the implementations of the adder, the multiplier and the divider. These choices influence both the cost and the performance of the FPU. Nevertheless, these issues have not been discussed in the open literature todate.In contrast to this lack of publications about the implementation of fully IEEE compliant FP operations or fully IEEE compliant FPUs, there are many publications about the implementation of specific floating-point operations for the case of normalized operands in a specific precision, e.g. [9, 26, 27, 32, 40, 43, 44], and these implementations are highly optimized for speed. Therefore, it is an important question, how to integrate the implementations of the different FP operations for normalized operands into a floating-point unit that supports more than one precision, denormalized numbers and special value results. This mainly includes the questions of which internal FP representations should be used in a FP unit and how the microarchitecture of a FP unit could be organized.

We present an answer to this question by developing and comparing three different rounding microarchitectures for a FP unit:

(I) In the first microarchitecture all the rounding computations are concentrated in a shared general rounding unit. This rounding unit considers the rounding for all IEEE results including the exponent wrapping and the FP exceptions for both single and double precision operations. A basic specification of such a rounder was first described in [10]. Thereafter, this rounder was implemented by our group, resulting in a version that will be included in [23], where also a rigorous correctness proof of the compliance with the IEEE rounding definition will be found. This rounder is further optimized in this thesis.

(II) In the second microarchitecture, the rounding for the case of normalized double precision results is computed within each functional unit and this rounded result is fixed for all the remaining cases in a second rounding step implemented by a shared gradual rounding unit. For the integrated rounding in the functional units assuming normalized, double precision operands and results, several algorithms from literature could be used. The implementation of the gradual rounder is based on the theory from [21] about gradual rounding. This rounding technique is applied in this thesis for full IEEE compliant rounding including the handling of denormalized results, special values, exceptions and exponent wrapping.

(III) By the third rounding architecture a completely new architecture for an IEEE compliant FPU is suggested. In this architecture no rounding hardware is shared, but each functional unit contains a dedicated rounding implementation that computes full IEEE rounding considering denormal and special values, exceptions and exponent wrapping. The special problem with the implementation of this microarchitecture is the implementation if the floating-point multiplication. The floating-point multiplier conventionally requires normalized significands in its operands and delivers an almost normalized result. For the fast integration of IEEE rounding into the FP multiplier, the significand has to be rounded in parallel to the mulplication computations. For the case of denormalized results this rounding has to be computed at a variable rounding position, that could be at each position within the significand. The idea, how to integrate such a variable position rounding into the multiplication implementation is the key concept for this microarchitecture. Such an implementation is developed in this work. Because such a multiplication implementation allows to work on normalized FP representations (even for denormalized values) as inputs and outputs, the internal FP representations can eb changed to normalized FP representations for this microarchitecture.

To find out the impact of the microarchitecture choice on the quality of the floating-point implementation, we model the performance and the cost of designs that differ by the use of the different microarchitectures. This would already be possible by a comparison of three FP designs, but to improve the expressiveness of the comparison, and to be able to compare the rounding architectures under several conditions, we additionally vary the FP multiplication and FP division implementation for each FP microarchitecture. For this purpose, we choose between two different FP multiplication and three different FP division implementations.

- For the FP multiplication implementation a Booth encoded adder tree is used either in a full-sized version that is able to compute double precision and single precision multiplications in one iteration or in a half-sized version that computes double precision multiplications in two iterations and single precision multiplications in one iteration. For the Booth encoded adder trees the constructions from [30], where we improved cost formula, are used.

- For the FP division implementation, we consider three different implementations of the Newton-Raphson iteration with an initial reciprocal approximation with an absolute approximation error bounded by 2^{-8}, 2^{-16}, and 2^{-28}, respectively. For this initial reciprocal approximation a fast implementation of a linear approximation formula using partial compressions is used, that we developed in [36, 39].

In combination with the three microarchitectures these options combine to a comparison of 18 different FP implementations.

All the FPUs designed in this work are fully compliant with the IEEE standard for all implemented operations, support both single and double precision, and deal with denormalized values and special cases in hardware. Because to design an IEEE compliant FPU is a complex and error-prone task, all the FPU designs are specified in full detail at gate level and the correctness of the FPU designs (in particular the compliance with the IEEE standard) is proven.

The performance of the designs is measured by a trace-driven run-time simulation of a R3000 like pipelined RISC processor [22, 23] that integrates the proposed floating-point implementations. The simulations are computed on traces of the SPECfp92 Benchmarks suite [17]. The costs of the designs are modeled by counting the gates that are required by the different implementations. Thus, based on the performance and the cost of each FP design, the quality of the FP designs and, in particular, the quality of the rounding microarchitectures can be compared.

This thesis is partitioned into the following chapters. Chapter 2 prepares the definitions of the IEEE FP standard in preparation for the description of the FP implementations. The basic description of the FP standard is similar to the description in [10, 23]. Moreover, in this chapter a general framework for the integrated description of different rounding functions is developed. Rigorous correctness proofs for the partitioning of full IEEE compliant rounding into these rounding functions are given. This chapter also provides computation utilities for the implementation of these rounding function. As one important basic concept, injection-based rounding [9, 40, 11, 12] is introduced. Finally, this chapter prepares the internal FP representations, by that the interfaces between the functional units and the shared rounding hardware are specified. Chapter 3 overviews the requirements on the implementation of a FPU und describes the microarchitectures and the design choices for the proposed FP designs. Chapter 4 describes the implementations of all basic FP operations for all three microarchitectures . In combination with a detailed description of the implementations at gate level, the correctness of the designs and the compliance with the IEEE standard is proven. Finally, in Chapter 5 the proposed FPU implementations are quantitatively analyzed and compared.

Chapter 2

IEEE Floating-Point Standard

The IEEE floating-point Standard 754-1985 [19] specifies floating-point number formats, operations and exception handling in detail. This chapter presents its information in a slightly different form following [10, 23].

2.1 Notation

We denote real values by small-letter names xyz and bit-strings by small capitalized names XYZ. The single bits of a bit-string $\mathrm{XYZ} \in \{0,1\}^n$ can be indexed by $\mathrm{XYZ}[n_2 : n_1] = (\mathrm{XYZ}[n_2], \cdots, \mathrm{XYZ}[n_1])$ with integers $n_2 = n_1 + n - 1$. The operation $< \mathrm{XYZ}[n_2 : n_1] >$ defines the binary value of $\mathrm{XYZ}[n_2 : n_1]$, $< \mathrm{XYZ}[n_2 : n_1] >_2$ defines the value of $\mathrm{XYZ}[n_2 : n_1]$ interpreted as a 2's-complement number, and $< \mathrm{XYZ}[n_2 : n_1] >_{bias_n}$ defines the value of $\mathrm{XYZ}[n_2 : n_1]$ interpreted as a biased binary number, that includes the bias $bias_n = 2^{n-1} - 1$:

$$\begin{aligned} < \mathrm{XYZ}[n_2 : n_1] > &= \sum_{i=n_1}^{n_2} \mathrm{XYZ}[i] \cdot 2^i \\ < \mathrm{XYZ}[n_2 : n_1] >_2 &= -\mathrm{XYZ}[n_2] \cdot 2^{n_2} + \sum_{i=n_1}^{n_2-1} \mathrm{XYZ}[i] \cdot 2^i \\ < \mathrm{XYZ}[n_2 : n_1] >_{bias_n} &= \sum_{i=n_1}^{n_2} \mathrm{XYZ}[i] \cdot 2^i - bias_n. \end{aligned}$$

To avoid negative indizes, we allow the right index of a bit-string to be larger than the left index, like in $\mathrm{XYZ}[n_1 : n_2]$. Then, we define a second version of the operations $<>$ and $<>_2$, that interpret the indizes to be multiplied by (-1). These operations are defined by

$$\begin{aligned} < \mathrm{XYZ}[n_1 : n_2] >_{neg} &= \sum_{i=n_1}^{n_2} \mathrm{XYZ}[i] \cdot 2^{-i} \\ < \mathrm{XYZ}[n_1 : n_2] >_{2neg} &= -\mathrm{XYZ}[n_1] \cdot 2^{-n_1} + \sum_{i=n_1+1}^{n_2} \mathrm{XYZ}[i] \cdot 2^{-i}. \end{aligned}$$

The operation $bin_\lambda^{\lambda+n-1}(x) : I\!R \longrightarrow \{0,1\}^n$ computes the bit-string of the binary representation of x of length n from bit-position with weight 2^λ to bit-position with weight $2^{\lambda+n-1}$. If x has two different binary representations, we choose the binary representation with finite length, so that in $x = \sum_i \mathrm{X}[i] \cdot 2^i$ the $\mathrm{X}[i]$ are unique and $bin_\lambda^{\lambda+n-1}(x)$ can be written by:

$$bin_\lambda^{\lambda+n-1}(x) = \mathrm{X}[\lambda+n-1 : \lambda].$$

For $\mathrm{X} \in \{0,1\}^n$ and $\mathrm{S} \in \{0,1\}$ we define

$$\overline{\mathrm{X}} = (\overline{\mathrm{X}[n-1]}, \ldots, \overline{\mathrm{X}[0]}) \quad \text{and} \quad \mathrm{X} \oplus \mathrm{S} = (\mathrm{X}[n-1] \oplus \mathrm{S}, \ldots, \mathrm{X}[0] \oplus \mathrm{S}).$$

Some crucial properties of two's complement numbers are (see [MP95])

$$\begin{aligned} < 0, \mathrm{X}[n-1:0] >_2 &= < \mathrm{X}[n-1:0] > \\ -< \mathrm{X}[n-1:0] >_2 &= < \overline{\mathrm{X}[n-1:0]} >_2 + 1 \\ < \mathrm{X}[n-1], \mathrm{X}[n-1:0] >_2 &= < \mathrm{X}[n-1:0] >_2 \\ < x[n-1:0] >_2 &\equiv < x[n-2:0] > \bmod 2^{n-1}. \end{aligned}$$

From these properties one immediately derives the basic subtraction algorithm for binary numbers. Let $\mathrm{X}, \mathrm{Y} \in \{0,1\}^n$ and let $<\mathrm{X}> \geq <\mathrm{Y}>$. Because $2^n > <\mathrm{X}> \geq <\mathrm{Y}>$ it suffices to compute the result modulo 2^n. Thus

$$\begin{aligned} < \mathrm{X} > - < \mathrm{Y} > &= < 0, \mathrm{X} >_2 - < 0, \mathrm{Y} >_2 \\ &= < 0, \mathrm{X} >_2 + < 1, \overline{\mathrm{Y}} >_2 + 1 \\ &\equiv < \mathrm{X} > + < \overline{\mathrm{Y}} > + 1 \bmod 2^n. \end{aligned}$$

Lemma 2.1 *Biased number strings* $\mathrm{X}[n-1:0] \neq 1^n$ *can be converted to two's complement number strings by (i) an increment and the invertation of the sign bit. Using* $< \mathrm{Y}[n-1:0] > = < \mathrm{X}[n-1:0] > + 1$*, we have:*

$$< (0, \mathrm{X}[n-1:0]) >_{bias_n} = < (\overline{\mathrm{Y}[n-1]}, \overline{\mathrm{Y}[n-1]}, \mathrm{Y}[n-2:0]) >_2.$$

(ii) In the conversion, the sequence of the sign bit inversion and the increment can also be reversed, so that:

$$< (0, \mathrm{X}[n-1:0]) >_{bias_n} = < (\overline{\mathrm{X}[n-1]}, \overline{\mathrm{X}[n-1]}, \mathrm{X}[n-2:0]) >_2 + 1.$$

Proof: (i):

$$\begin{aligned} < (0, \mathrm{X}[n-1:0]) >_{bias_n} &= < (0, \mathrm{X}[n-1:0]) >_2 - bias_n \\ &= < (0, \mathrm{X}[n-1:0]) >_2 + < (1, 10^{n-2}1) >_2 \\ &= < (0, \mathrm{Y}[n-1:0]) >_2 + < (1, 10^{n-2}0) >_2 \\ &= < (\overline{\mathrm{Y}[n-1]}, \overline{\mathrm{Y}[n-1]}, \mathrm{Y}[n-2:0]) >_2. \end{aligned}$$

(ii):

$$\begin{aligned} < (0, \mathrm{X}[n-1:0]) >_{bias_n} &= < (0, \mathrm{X}[n-1:0]) >_2 + < (1, 10^{n-2}1) >_2 \\ &= < (\overline{\mathrm{X}[n-1]}, \overline{\mathrm{X}[n-1]}, \mathrm{X}[n-2:0]) >_2 + 1. \end{aligned}$$

□

Lemma 2.2 *In the other direction, two's complement number strings* $\mathrm{X}[n-1:0] \neq (1, 0^{n-1})$ *can be converted to biased number strings by an inversion and a decrement. Using* $< \mathrm{Y}[n-1:0] >_2 = < (\overline{\mathrm{X}[n-1]}, \mathrm{X}[n-2:0]) >_2 - 1$*, we have:*

$$< \mathrm{X}[n-1:0] >_2 = < \mathrm{Y}[n-1:0] >_{bias_n}.$$

Proof:

$$\begin{aligned}
< \mathrm{x}[n-1:0] >_2 &= < (\mathrm{x}[n-1], \mathrm{x}[n-1:0]) >_2 + bias_n - bias_n \\
&= < (\mathrm{x}[n-1], \mathrm{x}[n-1:0]) >_2 + < (0, 01^{n-1}) >_2 - bias_n \\
&= < (\mathrm{x}[n-1], \mathrm{x}[n-1:0]) >_2 + < (0, 10^{n-1}) >_2 - 1 - bias_n \\
&= < (\overline{\mathrm{x}[n-1]}, \overline{\mathrm{x}[n-1]}, \mathrm{x}[n-2:0]) >_2 - 1 - bias_n \\
&= < (\overline{\mathrm{x}[n-1]}, \mathrm{x}[n-2:0]) >_2 - 1 - bias_n \\
&= < \mathrm{y}[n-1:0] >_2 - bias_n \\
&= < \mathrm{y}[n-1:0] >_{bias_n}
\end{aligned}$$

□

2.2 Numbers and Operations

2.2.1 Factorings

Every real number x can be factored into a sign factor (determined by a sign-bit s), a scale factor (determined by an exponent e) and a significand f:

$$x = (-1)^s \cdot f \cdot 2^e.$$

The tripel (s, e, f) is called a *factoring* and the operation

$$x = val(s, e, f) = (-1)^s \cdot f \cdot 2^e$$

computes the *value* of this factoring.

Although every factoring (s, e, f) is mapped to exactly one real number x by the operation $val(s, e, f)$, every real number x could be represented by infinitely many different factorings, that correspond to the same value

$$x = val(sign(x), 0, |x|) = val(sign(x), -1, 2 \cdot |x|) = \cdots.$$

Definition 2.1 *For a set of numbers $\mathcal{X}$, we define the set of factorings, $FACT(\mathcal{X})$, that represent numbers of $\mathcal{X}$ by*

$$FACT(\mathcal{X}) = \{(s, e, f) \mid s \in \{0, 1\}, e \in \mathbb{Z}, f \in \mathbb{R} \text{ and } val(s, e, f) \in \mathcal{X}\}$$

To define a unique factoring representation of a real number, *normalized factorings* are introduced:

Definition 2.2 *A* normalized factoring *(s', e', f') is a factoring with $s' \in \{0, 1\}$, $e' \in \mathbb{Z}$, $f' \in [1, 2[$. The condition $f' \in [1, 2[$ defines a* normalized significand *f'.*

Thus, every non-zero real value can be represented by a unique normalized factoring.

Definition 2.3 *For all non-zero factorings (s, e, f) with $f \neq 0$, we define the operation $\eta(s, e, f) = (s', e', f')$ to compute the normalized factoring (s', e', f'), so that $val(s', e', f') = val(s, e, f)$. For factorings of zero with $f = 0$, we define η to compute the identity function: $\eta(s, e, 0) = (s, e, 0)$. As in the normalization operation η the exponent range is not limited, η is called an* unbounded normalization shift*. The result of an unbounded normalization shift, (s', e', f'), is called* an unbounded normalized factoring*. Note, that from the definition of the unbounded normalization shift for zeros it follows, that also all factorings of zero with $f = 0$ are unbounded normalized.*

Lemma 2.3 *(i) For $f \neq 0$ and $k = -\lfloor log(f) \rfloor$, the unbounded normalization shift $\eta(s,e,f)$ can be computed by: $\eta(s,e,f) = (s, e-k, f \cdot 2^k)$.*

(ii) If $2^{-\gamma} \leq f < 2$, k can be interpreted as the number of leading zeros *lz of the binary representation $bin^0_{-\gamma}(f)$, so that $\eta(s,e,f) = (s, e-lz, f \cdot 2^{lz})$.*

Proof: (i) The result of the unbounded normalization shift $\eta(s,e,f)$ has to be the normalized factoring of (s,e,f). Therefore, $(s, e-k, f \cdot 2^k)$ has to fulfill the properties (1) $val(s, e-k, f \cdot 2^k) = val(s,e,f)$ and (2) $f \cdot 2^k \in [1,2[$:
(1) $val(s, e-k, f \cdot 2^k) = (-1)^s \cdot f \cdot 2^k \cdot 2^{e-k} = (-1)^s \cdot f \cdot 2^e = val(s,e,f)$.
(2) From $-log(f) \leq -\lfloor log(f) \rfloor < -log(f) + 1$, it follows, that

$$\begin{aligned} f \cdot 2^{-log(f)} &\leq f \cdot 2^{-\lfloor log(f) \rfloor} < f \cdot 2^{-log(f)+1} \\ f/f &\leq f \cdot 2^{-\lfloor log(f) \rfloor} < 2f/f, \end{aligned}$$

and, therefore, $f \cdot 2^k \in [1,2[$, as required.

(ii) We know from (i), that $f \cdot 2^k \in [1,2[$, and therefore, $f \in [2^{-k}, 2^{-k+1}[$. From the condition $f < 2^{-k+1}$, it follows, that in the binary representation of f, $f[0:\gamma] = bin^0_{-\gamma}(f)$, the bits $f[0:k-1]$ have to be zero. From $f > 2^{-k}$ and $f[0:k-1] = 0^k$, it follows, that $f[k] = 1$. Thus, $f[0:\gamma]$ contains exactly $lz = k$ leading zeros and the lemma follows. □

Definition 2.4 *In contrast to the definition of an unbounded normalization shift, we define a* bounded normalization shift *of (s,e,f) by the operation $\lfloor \eta_\beta \rfloor(s,e,f) = (s'', e'', f'')$:*

$$\lfloor \eta_\beta \rfloor(s,e,f) = \begin{cases} (s',e',f') = \eta(s,e,f) & \text{if } e' \geq \beta \\ (s, \beta, f \cdot 2^{e-\beta}) & \text{otherwise,} \end{cases} \tag{2.1}$$

i.e., the factoring (s'', e'', f'') is normalized only if the normalization operation does not produce an exponent smaller than β. The result of a bounded normalization shift (s'', e'', f'') is called a bounded normalized factoring. *From $val(s,e,f) = val(s, \beta, f^{e-\beta})$ and definition 2.3, it follows that also the bounded normalization shift does not change the value of the factoring and we have $val(\lfloor \eta_\beta \rfloor(s,e,f)) = val(s,e,f)$.*

2.2.2 IEEE Numbers

Floating-point number types form subsets of the Reals. They can be represented by factorings with limited and discretized value ranges for exponents and significands. The IEEE floating-point types are defined by describing the possible choices for the sign, the exponent and the significand of a factoring and by the definition of some special values, so that each IEEE floating-point type (precision) consists of:

- *Normalized numbers* are represented by normalized factorings (s', e', f'), where the exponent e' is an integer in the range $e_{min} \leq e' \leq e_{max}$ and the significand f' belongs to the discrete set $< \mathrm{F}'[0:p-1] >_{neg} \in \{1, 1+2^{-p+1}, ..., 2-2^{-p+1}\}$. The condition $f' \in [1,2[$ defines a *normalized significand f'*.

- *Denormalized numbers* are represented by factorings (s,e,f), where the exponent is $e = e_{min}$ and the significand f belongs to the discrete set $< f[0:p-1] >_{neg} \in \{0, 2^{-p+1}, ..., 1-2^{-p+1}\}$. As $f \in [0,1[$, and thus $f \notin [1,2[$, the significand is called *denormalized*.

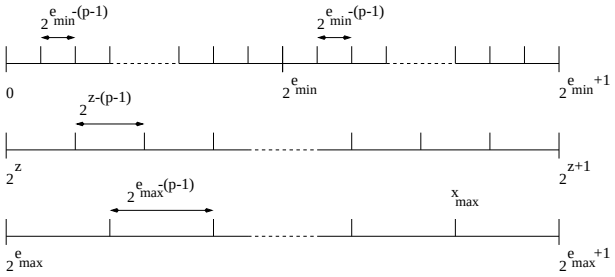

Figure 2.1: Geometry of IEEE floating-point numbers.

- *Special values* are defined by the set consisting of $+\infty$, $-\infty$ and two types of *Not a Number* (NaN): signaling NaN (sNaN) and quiet NaN (qNaN). These values can not be represented by factorings with finite exponents. Therefore, *special* bit strings for the representation of special numbers are used. Nevertheless, we use the symbols $(0, e_\infty, f_\infty)$ for the factoring of $+\infty$ and the symbols $(1, e_\infty, f_\infty)$ for the factoring $-\infty$ corresponding to the special bit strings for the IEEE infinity representations. The NaN representations are not unique. Therefore, if it does not matter which representation is chosen, we use the symbols (s, e_{sNaN}, f_{sNaN}) for sNaN factorings and the symbols (s, e_{qNaN}, f_{qNaN}) for qNaN factorings corresponding to an arbitrary IEEE NaN representation. If we want to refer to a specific NaN representation, we index them with a positive number, like in $(s_1, e_{sNaN1}, f_{sNaN1})$ or $(s_2, e_{qNaN2}, f_{qNaN2})$. We define these factorings of special values to be normalized, so that the normalization shifts compute the identity function on them. Moreover, we extend the definition of the function *val* by $val(s, e_\infty, f_\infty) = (-1)^s \cdot \infty$, $val(s, e_{sNaN}, f_{sNaN}) = sNaN$ and $val(s, e_{qNaN}, f_{qNaN}) = qNaN$.

The union of normalized numbers and denormalized numbers form the *representable numbers* of an IEEE floating-point type. The geometry of the representable numbers is depicted in figure 2.1 on page 8 and shows the following properties:

- For every exponent value e between e_{min} and e_{max} there are two intervals of representable (normalized) numbers: $[2^e, 2^{e+1}[$ and $]-2^{e+1}, -2^e]$. The gaps between consecutive representable numbers in these intervals are $2^{e-(p-1)}$.

- As the exponent value increases by one, the length of the interval $[2^e, 2^{e+1}[$ doubles, and the gaps between the representable (normalized) numbers double as well. Thus, the number of representable numbers per interval is fixed and it equals 2^{p-1}.

- The denormalized numbers are the representable numbers in the interval $]-2^{e_{min}}, 2^{e_{min}}[$. The gaps between consecutive representable numbers in this interval are $2^{e_{min}-(p-1)}$. Thus, the gaps in the interval $[0, 2^{e_{min}}[$ equal the gaps in the interval $[2^{e_{min}}, 2^{e_{min}+1}[$. This property is called in the literature *gradual underflow* since the large gap between zero and $2^{e_{min}}$ is filled with denormalized numbers.

The IEEE definition of normalized and denormalized floating-point numbers includes a definition of their factorings, so that we distinguish between the following sets of factorings of an IEEE floating-point type:

Definition 2.5 *The set of normalized IEEE factorings, $NORfact_{n,p}$, the set of denormalized IEEE factorings, $DENfact_{n,p}$, and the set of special IEEE factorings, $SPEfact$:*

$$\begin{aligned} NORfact_{n,p} &= \{(s,e,f) \mid s \in \{0,1\}, e \in \mathbb{Z} \text{ with } (e_{min} \le e \le e_{max}), \\ &\qquad \text{and } b \in \mathbb{N} \text{ with } (0 \le b < 2^{p-1}) : \quad f = 1 + b \cdot 2^{-(p-1)}\Big\} \\ DENfact_{n,p} &= \Big\{(s,e_{min},f) \mid \mathrm{s} \in \{0,1\}, b \in \mathbb{N} \text{ with } (0 \le b < 2^{p-1}) : f = b \cdot 2^{-(p-1)}\Big\} \\ SPEfact &= \{(0,e_\infty,f_\infty),(1,e_\infty,f_\infty),(s,e_{qNaN},f_{qNaN}),(s,e_{sNaN},f_{sNaN})\}\,. \end{aligned}$$

We define the set of IEEE factorings by

$$IEEEfact_{n,p} = DENfact_{n,p} \cup NORfact_{n,p} \cup SPEfact.$$

Accordingly, the following sets of numbers are defined:

Definition 2.6 *For each IEEE floating-point type, the set of* normalized numbers, $\mathcal{NOR}_{n,p}$, *the set of* denormalized numbers, $\mathcal{DEN}_{n,p}$, *and the set of IEEE* special values, $\mathcal{SPE}$, *are defined by*

$$\begin{aligned} \mathcal{NOR}_{n,p} &= \{x \mid \exists (s,e,f) \in NORfact_{n,p} : x = val(s,e,f)\} \\ \mathcal{DEN}_{n,p} &= \{x \mid \exists (s,e,f) \in DENfact_{n,p} : x = val(s,e,f)\} \\ \mathcal{SPE} &= \{+\infty, -\infty, qNaN, sNaN\}\,. \end{aligned}$$

We define the set of representable numbers *of an IEEE floating-point type, $\mathcal{REP}_{n,p}$, by*

$$\mathcal{REP}_{n,p} = \mathcal{DEN}_{n,p} \cup \mathcal{NOR}_{n,p}.$$

The set of values of an IEEE floating-point type, $\mathcal{FP}_{n,p}$, additionaly includes the special values, so that

$$\begin{aligned} \mathcal{FP}_{n,p} &= \mathcal{DEN}_{n,p} \cup \mathcal{NOR}_{n,p} \cup \mathcal{SPE} \\ &= \mathcal{REP}_{n,p} \cup \mathcal{SPE}. \end{aligned}$$

Lemma 2.4 *The sets of denormalized and normalized IEEE numbers/factorings are disjunct: $\mathcal{NOR}_{n,p} \cap \mathcal{DEN}_{n,p} = \emptyset$ and $NORfact_{n,p} \cap DENfact_{n,p} = \emptyset$. Thus, each IEEE floating-point value $x \in \mathcal{FP}_{n,p}$ has a unique IEEE factoring $(s,e,f) \in IEEEfact_{n,p}$ with $x = val(s,e,f)$.*

Proof: For normalized IEEE factorings $(s_{nor}, e_{nor}, f_{nor}) \in NORfact_{n,p}$, we have $e_{nor} \ge e_{min}$ and $f_{nor} \ge 1$, so that $|x_{nor}| = |val(s_{nor}, e_{nor}, f_{nor})| \ge 2^{e_{min}}$. For denormalized IEEE factorings $(s_{den}, e_{den}, f_{den}) \in DENfact_{n,p}$, we have $e_{den} = e_{min}$ and $f_{nor} < 1$, so that $|x_{den}| = |val(s_{den}, e_{den}, f_{den})| < 2^{e_{min}}$ Thus, all normalized IEEE factorings have a larger absolute value than each of the denormalized IEEE factorings, $|x_{nor}| > |x_{den}|$, so that $\mathcal{NOR}_{n,p} \cap \mathcal{DEN}_{n,p} = \emptyset$ and $NORfact_{n,p} \cap DENfact_{n,p} = \emptyset$. For the second part of the lemma we additionaly have to use, that also each special value has a unique factoring representation. This can easily be seen from the definitions of $\mathcal{SPE}$ and $SPEfact$. □

Lemma 2.5 *From an arbitrary factoring $(s,e,f) \in FACT(\mathcal{FP}_{n,p})$ of an IEEE FP number $x = val(s,e,f) \in \mathcal{FP}_{n,p}$, the bounded normalization shift $\lceil \eta_{e_{min}} \rceil (s,e,f) = (s'',e'',f'')$ computes the corresponding IEEE factoring $(s'',e'',f'') \in IEEEfact_{n,p}$ with $val(s'',e'',f'') = x = val(s,e,f)$.*

precision	p	n	$bias_n$	e_{min}	e_{max}	$\|x\|_{min}$	$\|x\|_{max}$
single	24	8	127	-126	127	$\approx 1.4 \cdot 10^{-45}$	$\approx 3.4 \cdot 10^{38}$
single ext.	≥ 32	≥ 11	—	≤ -1022	≥ 1023	—	—
double	53	11	1023	-1022	1023	$\approx 4.9 \cdot 10^{-322}$	$\approx 1.8 \cdot 10^{310}$
double ext.	≥ 64	≥ 15	—	≤ -16382	≥ 16383	—	—

Table 2.1: IEEE floating-point formats.

Proof: The proof consists of two parts for the cases: (a) $val(s,e,f) \in \mathcal{DEN}_{n,p}$ and (b) $(s,e,f) \in \mathcal{NOR}_{n,p}$.

(a) For $val(s,e,f) \in \mathcal{DEN}_{n,p}$, there is a denormalized IEEE factoring $(a,b,c) \in DENfact_{n,p}$ with $val(s,e,f) = val(a,b,c)$. We know already from the definition of the bounded normalization shift 2.4, that also $val(s'',e'',f'') = val(s,e,f)$. From the definition of denormalized IEEE factorings (see definition 2.5), it follows that $b = e_{min}$. Therefore, for the proof of $(s'',e'',f'') = (a,b,c)$ and $(s'',e'',f'') \in DENfact_{n,p}$, it suffices to show that $e'' = e_{min}$. From $val(s,e,f) \in \mathcal{DEN}_{n,p}$ it follows, that $|val(s,e,f)| < 2^{e_{min}}$. We consider the normalized factoring $(s,e',f') = \eta(s,e,f)$. Because $f' \geq 1$, and $2^{e'} \cdot f' < 2^{e_{min}}$, the exponent $e' < e_{min}$ is smaller than the exponent bound of the bounded normalization shift. Therefore, it follows from the definition of $\lceil \eta_{e_{min}} \rceil$ that $e'' = e_{min}$ and part (a) of the proof is completed.

(b) For $val(s,e,f) \in \mathcal{NOR}_{n,p}$, we have to show that (s'',e'',f'') is normalized. From $val(s,e,f) \in \mathcal{NOR}_{n,p}$, it follows, that $|val(s,e,f)| \geq 2^{e_{min}}$. We consider the normalized factoring $(s,e',f') = \eta(s,e,f)$. Because $f' < 2$, and $2^{e'} \cdot f' \geq 2^{e_{min}}$, the exponent $e' \geq e_{min}$ is larger than or equal to the exponent bound of the bounded normalization shift. Therefore, it follows from the definition of $\lceil \eta_{e_{min}} \rceil$ that (s'',e'',f'') is the normalized factoring $(s'',e'',f'') = \eta(s,e,f)$ and also part (b) of the proof is completed. □

Definition 2.7 *If an unbounded normalization shift is computed on the factorings from $FACT(\mathcal{FP}_{n,p})$, we get a set, that includes the (unbounded) normalized factoring for each IEEE number in $\mathcal{FP}_{n,p}$. In this way we define the set of NF factorings $NFfact_{n,p}$ by:*

$$NFfact_{n,p} = \{(s,e,f) \mid (s,e,f) \in FACT(\mathcal{FP}_{n,p}) \text{ and } (s,e,f) = \eta(s,e,f)\}.$$

In the early days of floating-point design, many different formats with different values for e_{min}, e_{max}, n and p were used. The success of the IEEE floating-point Standard 754-1985 [19] reduced the supported FP types to a few: *single*, *double*, *single extended* and *double extended*. The parameters for these precisions are given in table 2.1. In an IEEE compliant FPU-Design only some of these FP-types have to be implemented. We will focus on the implementation of the *single* and *double* precision types, because these types are most commonly used and the integration of additional types would be straight-forward.

2.2.3 Packed IEEE Floating-Point Format

At the bit level, numbers in the single and double formats are composed of three fields corresponding to sign, biased exponent and fraction (significand without first bit) like depicted in figure 2.2. In the biased exponent representation, a bias of $bias_n = 2^{n-1} - 1$ is used. Because $bias_n = -e_{min} + 1 = e_{max}$ (see table 2.1 for single and double precision), the

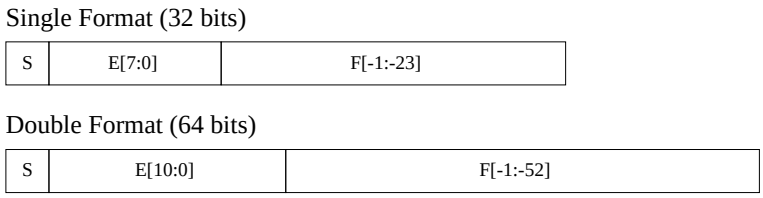

Figure 2.2: Packed IEEE floating-point format.

value of $e+bias_n$ is in the range $1 \leq e+bias_n \leq 2^n-2$. Thus, the bit strings for 0 and 2^n-1 do not occur in the n-bit biased binary representation $\mathrm{E}[n-1:0] = bin_0^{n-1}(e+bias_n)$. Therefore, the exponent strings $\mathrm{E} = 0^n$ and $\mathrm{E} = 1^n$ are used for the representation of denormalized numbers and special values.

The significand f of a representable number can be represented with p bits $\mathrm{F}[0{:}p-1] = bin_{-p+1}^{0}(f)$. But only the fraction $\mathrm{F}[1{:}p-1]$ is included in the number string, and the *hidden bit* $\mathrm{F}[0]$ does not occur explicitely in the number representation. The hidden bit $\mathrm{F}[0]$ equals 1, iff f is normalized, and $\mathrm{F}[0]$ equals 0, iff f is denormalized. Because the exponent representation of e_{min} for denormalized numbers differs from all exponent representations from normalized numbers, the hidden bit $\mathrm{F}[0]$ can be extracted from the exponent representation.

The value of a number x represented by the packed representation $(\mathrm{S}, \mathrm{E}[n-1{:}0], \mathrm{F}[1{:}p-1])$ is defined by

1. If $\mathrm{E}[n-1:0] = 0^n$ (denormalized numbers),
   then $x = (-1)^{\mathrm{S}} \cdot <(0.\mathrm{F}[1:p-1])>_{neg} \cdot 2^{e_{min}}$.

2. If $\mathrm{E}[n-1:0] \neq 0^n$ and $\mathrm{E}[n-1:0] \neq 1^n$ (normalized numbers),
   then $x = (-1)^{\mathrm{S}} \cdot <(1.\mathrm{F}[1:p-1])>_{neg} \cdot 2^{<\mathrm{E}[n-1:0]>_{bias_n}}$.

3. If $\mathrm{E}[n-1:0] = 1^n$ (special values),
   then x is a special value depending on $\mathrm{F}[1:p-1]$:

   - If $\mathrm{F}[1:p-1] = 0^{p-1}$, then x is ∞ and has the sign of $(-1)^{\mathrm{S}}$.
   - If $\mathrm{F}[1:p-1] \neq 0^{p-1}$, then x is NaN regardless of S.

     The standard does not specify how to distinguish between signaling and quiet NaNs. We follow the specification used in [29] and distinguish between signaling and quiet NaNs by the value of $f[1]$: If $f[1] = 1$, then x is a signaling NaN (sNaN), otherwise x is a quiet NaN (qNaN).

2.2.4 Operations

Beside floating-point types, the IEEE FP Standard defines arithmetic operations that have to be implemented in hardware or in software. In this section, we only define exact results of these operations for finite input operands $x = val(s_x, e_x, f_x) \in \mathcal{REP}_{n,p}$ and $y = val(s_y, e_y, f_y) \in \mathcal{REP}_{n,p}$. The computations involving special values will be described later in combination with the exception handling.

- Addition/substraction. We use the bit SOP to distinguish between addition (SOP = 0) and substraction (SOP = 1). The exact value of the addition/substraction result is defined by:

$$exact_{ADD/SUB} = x + (-1)^{\mathrm{SOP}} \cdot y$$

  The computation of the factoring of this value involves several steps. Therefore, we postpone its specification to the description of the addition implementations.

- Multiplication. The exact product of x and y is denoted by:

$$exact_{MULT} = x \cdot y = (-1)^{s_x + s_y} \cdot (f_x \cdot f_y) \cdot 2^{e_x + e_y}.$$

  Thus, $(s_x \otimes s_y, f_x \cdot f_y, e_x + e_y)$ is a factoring of $exact_{MULT}$.

- Division. The exact quotient of x and y is denoted by:

$$exact_{DIV} = x/y = (-1)^{s_x - s_y} \cdot (f_x/f_y) \cdot 2^{e_x - e_y}.$$

  Thus, $(s_x \otimes s_y, f_x/f_y, e_x - e_y)$ is a factoring of $exact_{DIV}$.

- Square-root. For non-negative $x \geq 0$ the exact square-root of x is denoted by:

$$exact_{SQRT} = \sqrt{x} = \sqrt{f_x \cdot 2^{(e_x MOD 2)}} \cdot 2^{e_x DIV 2}$$

  Thus, $(0, \sqrt{f_x \cdot 2^{(e_x MOD 2)}}, e_x DIV 2)$ is a factoring of $exact_{SQRT}$.

- Remainder. For non-zero y the exact remainder $xREMy$ is defined by:

$$exact_{REM} = x - y \cdot n,$$

  where n is the integer nearest the exact value x/y; whenever $|n - x/y| = 0.5$, then n is even.

- Conversion. In conversions, the input operand has already the exact value of the conversion. This value has than to be converted to the destination's format.

$$exact_{CONV} = x.$$

  In this operation we have to consider, that the input operand could also be an integer $< \mathrm{x} >_2$ in two's complement representation. Then,

$$exact_{CONV} = < \mathrm{x} >_2.$$

  A factoring of $exact_{CONV}$ is given by (s_x, e_x, f_x) or $(sign(< \mathrm{x} >_2), 0, |< \mathrm{x} >_2|)$.

Moreover, the computations of the absolute value ($s_x := 0$) and the negative of a floating-point number ($s_x := not(s_x)$) are suggested to be implemented.

The floating-point types are not closed on all of these arithmetic operations. Therefore, the exact result of an operation might not belong to the same floating-point type. To be able to operate on results of operations, nevertheless, it is a basic principle of the IEEE standard to consider the exact result of an operation first and map it to a floating-point number by a selected rounding scheme to get a rounded result in the same floating-point type, finally.

Apart from that, the *test operation* (comparison) delivers a boolean value from two floating-point inputs. There are 26 different comparisons defined by the IEEE standard, which we decode by 5 condition code bits COND[4 : 0]. The bits COND[3 : 0] switch the conditions $\{>, <, =, UNORDERD(?)\}$, and COND[4] negates the boolean result bit. Only 26 of the 32 possible combinations are required by the standard. These are listed in table 2.2.

condition	COND$[4:0]$	$>$	$<$	$=$	?	INV if ?
$=$	00010	F	F	T	F	No
? <>	01101	T	T	F	T	No
$>$	01000	T	F	F	F	Yes
$>=$	01010	T	F	T	F	Yes
$<$	00100	F	T	F	F	Yes
$<=$	00110	F	T	T	F	Yes
?	00001	F	F	F	T	No
$<>$	01100	T	T	F	F	Yes
$<=>$	01110	T	T	T	F	Yes
? $>$	01001	T	F	F	T	No
? $>=$	01011	T	F	T	T	No
? $<$	00101	F	T	F	T	No
? $<=$	00111	F	T	T	T	No
? $=$	00011	F	F	T	T	No
$NOT(>)$	11000	F	T	T	T	Yes
$NOT(>=)$	11010	F	T	F	T	Yes
$NOT(<)$	10100	T	F	T	T	Yes
$NOT(<=)$	10110	T	F	F	T	Yes
$NOT(?)$	10001	T	T	T	F	No
$NOT(<>)$	11100	F	F	T	T	Yes
$NOT(<=>)$	11110	F	F	F	T	Yes
$NOT(? >)$	11001	F	T	T	F	No
$NOT(? >=)$	11011	F	T	F	F	No
$NOT(? <)$	10101	T	F	T	F	No
$NOT(? <=)$	10111	T	F	F	F	No
$NOT(? =)$	10011	T	T	F	F	No

Table 2.2: IEEE test operation (comparison).

2.3 Rounding

2.3.1 IEEE Rounding Definition

IEEE rounding is a mapping from the reals into an IEEE floating-point type. The IEEE standard defines rounding in four rounding modes: round toward 0 (RZ), round to nearest(even) (RNE), round toward $+\infty$ (RI) and round toward $-\infty$ (RMI). Let $\mathcal{REP}^\infty = \mathcal{REP} \cup \{+\infty, -\infty\}$. For the rounding mode $mode \in \{RZ, RNE, RI, RMI\}$, we present the rounding definition of the IEEE standard by the description of the rounding function $r_{mode} : I\!R \longrightarrow \mathcal{REP}^\infty$. For the three directed rounding modes $mode \in \{RZ, RI, RMI\}$ the obvious meaning of IEEE rounding is given by:

$$\begin{aligned} r_{RI}(x) &= min\{y \in \mathcal{REP}^\infty \mid x \leq y\} \\ r_{RMI}(x) &= max\{y \in \mathcal{REP}^\infty \mid x \geq y\} \\ r_{RZ}(x) &= \begin{cases} r_{RMI}(x) & \text{if } x \geq 0 \\ r_{RI}(x) & \text{if } x < 0. \end{cases} \end{aligned}$$

The definition of the rounding function r_{RNE} is a bit more complicated. Let $x^*_{max} = 2^{e_{max}}(2 - 2^{-p})$ and let $y \in \mathcal{REP}$ be the representable number nearest to x if this is unique, otherwise let $y \in \mathcal{REP}$ be the even representable number, that is nearest to x. Then,

$$r_{RNE}(x) = \begin{cases} +\infty & \text{if } x \geq x^*_{max} \\ -\infty & \text{if } x \leq -x^*_{max} \\ y & \text{otherwise.} \end{cases}$$

2.3.2 Rounding Functions

In this section we define rounding for a particular precision λ, so that a real number x is mapped to an integral multiple of $2^{-\lambda}$. For a precision λ, we define four rounding functions, that we index by the names of the four IEEE rounding modes RZ, RNE, RI, and RMI. We will show in the next section how these rounding functions can be used to implement IEEE rounding. For the definition of the rounding functions, we chose the integer t, so that $t \cdot 2^{-\lambda} \leq x < (t+1) \cdot 2^{-\lambda}$ and t^* is the even number of the set $\{t, t+1\}$.

$$rnd_{RI,\lambda}(x) = \begin{cases} t \cdot 2^{-\lambda} & \text{if } x = t \cdot 2^{-\lambda} \\ (t+1) \cdot 2^{-\lambda} & \text{otherwise} \end{cases} \tag{2.2}$$

$$rnd_{RMI,\lambda}(x) = t \cdot 2^{-\lambda} \tag{2.3}$$

$$rnd_{RZ,\lambda}(x) = \begin{cases} t \cdot 2^{-\lambda} & \text{if } x \geq 0 \text{ OR } x = t \cdot 2^{-\lambda} \\ (t+1) \cdot 2^{-\lambda} & \text{otherwise} \end{cases} \tag{2.4}$$

$$rnd_{RNE,\lambda}(x) = \begin{cases} t \cdot 2^{-\lambda} & \text{if } x < (t+0.5) \cdot 2^{\lambda} \\ t^* \cdot 2^{-\lambda} & \text{if } x = (t+0.5) \cdot 2^{\lambda} \\ (t+1) \cdot 2^{-\lambda} & \text{otherwise} \end{cases} \tag{2.5}$$

For the rounding of sign-magnitude representations with $x = (-1)^s \cdot |x|$, the four IEEE rounding modes for the rounding of x can be reduced to the three IEEE rounding modes $\{RZ, RNE, RI\}$ for the rounding of $|x|$ [33]. This is done by reducing the directed rounding modes RZ, RI and RMI to the rounding modes RZ and RI for the rounding on positive arguments based on the sign s of the number. Thus, leaving only the three rounding modes RZ, RNE, and RI that have only to operate on the positive argument $|x|$. In conjunction

mode	RND_MODE[1:0]	$mode \star 0:$	SR_MODE[1:0]	$mode \star 1:$	SR_MODE[1:0]
RZ	00	RZ	00	RZ	00
RNE	01	RNE	01	RNE	01
RI	10	RI	10	RZ	00
RMI	11	RZ	00	RI	10

Table 2.3: rounding mode reduction for sign-magnitude arguments

with table 2.3 for the rounding mode reduction, we define the $\star$-operation:

$$\begin{aligned} \star\text{:} \quad & \{RZ, RNE, RI, RMI\} \times \{0,1\} \longrightarrow \{RZ, RNE, RI\} \\ & (mode, s) \longmapsto mode \star s \end{aligned}$$

that maps the rounding mode $mode$ and the sign s to the corresponding reduced rounding mode $mode \star s$. Based on this definition, the rounding mode reduction can be written as:

$$rnd_{mode,\lambda}(x) = rnd_{mode,\lambda}((-1)^s \cdot |x|) = (-1)^s \cdot rnd_{mode\star s,\lambda}(|x|).$$

If we encode the four IEEE rounding modes by RND_MODE[1:0] and the three reduced rounding modes by SR_MODE[1:0] according to table 2.3, the $\star$-operation can be expressed by the equations:

$$\text{SR\_MODE}[1] = \text{RND\_MODE}[1] \wedge (\text{RND\_MODE}[0] \overline{\otimes} s) \tag{2.6}$$

$$\text{SR\_MODE}[0] = \overline{\text{RND\_MODE}[1]} \wedge \text{RND\_MODE}[0]. \tag{2.7}$$

Furthermore, Quach et al. [33] suggested to implement RNE by round to nearest up (RNU). With an integer t, such that $t \cdot 2^{-\lambda} \leq |x| < (t+1) \cdot 2^{-\lambda}$, the rounding mode RNU is defined by:

$$rnd_{RNU,\lambda}(|x|) = \begin{cases} t \cdot 2^{-\lambda} & \text{if } |x| < (t+0.5) \cdot 2^{-\lambda} \\ (t+1) \cdot 2^{-\lambda} & \text{otherwise.} \end{cases} \tag{2.8}$$

The reason that RNE can be implemented by RNU is that $rnd_{RNU,\lambda}(x) \neq rnd_{RNE,\lambda}(x)$ iff $x = (t+0.5) \cdot 2^{-\lambda}$ and the LSB of the binary encoding of $(t+1) \cdot 2^{-\lambda}$ is 1. Therefore, obtaining $rnd_{RNE,\lambda}(x)$ from $rnd_{RNU,\lambda}(x)$ can be accomplished by "pulling down" the LSB, when $x = (t+0.5) \cdot 2^{-\lambda}$.

2.3.3 IEEE Rounding Functions

In this section a description of IEEE rounding is given, which is more practical than the definition by the IEEE standard. The following lemma shows how the rounding functions for a particular precision λ from the previous section are related to IEEE rounding. After that we will consider the IEEE rounding on factorings.

Lemma 2.6 *For* $2^{e'} \leq |x| < 2^{e'+1}$ *and* $mode \in \{RZ, RNE, RI, RMI\}$, *let* $e'' = max\{e', e_{min}\}$ *and* $xr = rnd_{mode,-e''+p-1}(x)$. *Then,*

$$r_{mode}(x) = \begin{cases} \infty & \text{if } xr \geq 2^{e_{max}+1} \text{ and } mode \in \{RNE, RI\} \\ x_{max} & \text{if } xr \geq 2^{e_{max}+1} \text{ and } mode \in \{RZ, RMI\} \\ -x_{max} & \text{if } xr \leq -2^{e_{max}+1} \text{ and } mode \in \{RZ, RI\} \\ -\infty & \text{if } xr \leq -2^{e_{max}+1} \text{ and } mode \in \{RNE, RMI\} \\ xr & \text{otherwise} \end{cases}$$

Proof: In the definitions of IEEE rounding, in all cases the rounded result $r_{mode}(x)$ is either the nearest number of the destination FP type that is larger than the operand x or the nearest number of the destination FP type that is smaller than or equal to the operand x. In the following, we distinguish between the cases: (a) $|x| < x_{max}$ and (b) $|x| \geq x_{max}$.

(a) For $|x| < x_{max}$, we get $|xr| \leq x_{max} < 2^{e_{max}+1}$, so that we have to show that the IEEE rounding definition from $r_{mode}(x)$ is equivalent to $rnd_{mode,-e''+p-1}(x)$ for all rounding modes. We will first show that the two possible rounding choices of the IEEE rounding definitions are identical to the two possible results from the definition of the function $rnd_{mode,-e''+p-1}(x)$.

(i) For $|x| < 2^{e_{min}}$, we are in the range of denormalized numbers, so that the gap between two consecutive FP numbers is $2^{e_{min}-p+1}$ (see geometry of representable numbers in section 2.2.2) and there is an integer k, such that

$$f1 = k \cdot 2^{e_{min}-p+1} \leq x < (k+1) \cdot 2^{e_{min}-p+1} = f2$$

and $f1, f2 \in \mathcal{FP}_{n,p}$ are the nearest floating-point numbers in $\mathcal{FP}_{n,p}$ larger than and smaller than or equal to x.

The definition of $rnd_{mode,-e''+p-1}(x)$ uses the possible rounding results: $f3 = l \cdot 2^{e''-p+1}$ and $f4 = (l+1) \cdot 2^{e''-p+1}$ with $f3 \leq x < f4$. Since $|x| < 2^{e_{min}}$, we get $e' < e_{min}$ and $e'' = e_{min}$. Thus, the possible rounding choices are the same like in the IEEE rounding definition: $f1 = k \cdot 2^{e_{min}-p+1} = f3$ and $f2 = (k+1) \cdot 2^{e_{min}-p+1} = f4$.

(ii) For $|x| \geq 2^{e_{min}}$, we are in the range of normalized numbers, so that the gap between two consecutive FP numbers is $2^{e'-p+1}$ (see geometry of representable numbers in section 2.2.2). In this case, there is an integer k, so that

$$f1 = k \cdot 2^{e'-p+1} \leq x < (k+1) \cdot 2^{e'-p+1} = f2$$

and $f1, f2 \in \mathcal{FP}_{n,p}$ are the nearest floating-point numbers in $\mathcal{FP}_{n,p}$ larger than and smaller than or equal to x. Because for $|x| \geq 2^{e_{min}}$, we get $e' \geq e_{min}$ and $e'' = e'$, the numbers $f1$ and $f2$ agree with the two possible rounding results of the function $rnd_{mode,-e''+p-1}(x) = rnd_{mode,-e'+p-1}(x)$ also in this case.

Based on the agreement of the two possible rounding choices, one can now easily check, that also the rounding decisions are the same for both the IEEE definition $r_{mode}(x)$ and the rounding functions $rnd_{mode,-e''+p-1}(x)$ for all four rounding modes. This completes the proof for case (a).

(b) For $|x| \geq x_{max}$, the possible rounding choices for the IEEE rounding definition $r_{mode}(x)$ are $+/- x_{max}$ and $+/- \infty$. One can easily check that the specification of the rounding cases in the lemma corresponds to the IEEE definition for $|rnd_{mode,e''-p+1}(x)| \geq 2^{e_{max}+1}$, so that we only have to proof part (b) for $|rnd_{mode,e''-p+1}(x)| < 2^{e_{max}+1}$. For $|x| \geq x_{max}$, the condition $|rnd_{mode,e''-p+1}(x)| < 2^{e_{max}+1}$ can only be fulfilled, if also at least one of the following five conditions is fulfilled:

(i) $|x| = x_{max}$;

(ii) $mode = RZ$;

(iii) $x > 0$ and $mode = RMI$;

(iv) $x < 0$ and $mode = RI$;

(v) $|x| < (2 - 2^{-p}) \cdot 2^{e_{max}}$ and $mode = RNE$.

For the rounding mode RNE, the value $|x| = (2-2^{-p})\cdot 2^{e_{max}}$ (rounding interval midpoint) is not included in case (v), because this value is rounded to $\pm 2^{e_{max}+1}$, which is the 'even' value among the two rounding choices. Since $(2-2^{-p})\cdot 2^{e_{max}} = x^*_{max}$, in all of these five cases, the IEEE rounding definition leads to $|r_{mode}(x)| = x_{max}$. From $|x| \geq x_{max}$, we get $e'' = e' \geq e_{max}$, so that all results of $rnd_{mode,-e''+p-1}(x)$ are integral multiples of $2^{e_{max}-p+1}$. Because the only multiples of $2^{e_{max}-p+1}$, that have a magnitude larger than or equal to x_{max} and smaller than $2^{e_{max}+1}$, are $+/-x_{max}$, it follows that also $|rnd_{mode,e''-p+1}(x)| = x_{max}$ and the proof of the lemma is completed. □

In an FP implementation, the exact result of an operation will be represented by a factoring. In the following, we therefore define IEEE rounding on factorings. We do not have any conditions on the input factoring, but by the requirements for the destination factoring we distinguish between two versions: We would like to get either the IEEE factoring or the NF factoring of the rounded result.

Definition 2.8 *For mode* $\in \{RZ, RNE, RI, RMI\}$*, the rounding function* $iround_{mode} : FACT(\mathbb{R}) \longrightarrow IEEEfact$ *is defined to compute the IEEE factoring of the rounded result:*

$$iround_{mode}(s,e,f) = (sr,er,fr) \in IEEEfact, \text{ with } val(sr,er,fr) = r_{mode}(val(s,e,f))$$

and the rounding function $nround_{mode} : FACT(\mathbb{R}) \longrightarrow NFfact$ *is defined to compute the NF factoring of the rounded result:*

$$nround_{mode}(s,e,f) = (sr,er,fr) \in NFfact, \text{ with } val(sr,er,fr) = r_{mode}(val(s,e,f)).$$

Moreover, we define some functions that will be used for the rounding computations

Definition 2.9 *For mode* $\star\, s \in \{RZ, RNE, RI\}$, aligned significand rounding *is the rounding at the least significant bit position* $p-1$ *of the significand* f*:*

$$a\_sig\_rnd_{mode\star s}(s,e,f) = (s,e,rnd_{mode\star s,p-1}(f)). \tag{2.9}$$

With mode $\star\, s \in \{RZ, RNE, RI\}$*, and* $vp = (p-1) - max\{0, e_{min} - e\}$, normalized significand rounding *is the rounding of the significand* f *at the (variable) position* vp*:*

$$n\_sig\_rnd_{mode\star s}(s,e,f) = (s,e,rnd_{mode\star s,vp}(f)). \tag{2.10}$$

We define the post-normalization shift *function, that normalizes a factoring, iff the significand* f *of the factoring equals* $f = 2$*:*

$$post\_norm(s,e,f) \;=\; \begin{cases} (s,e+1,1) & \text{if } f = 2 \\ (s,e,f) & \text{otherwise.} \end{cases} \tag{2.11}$$

The exponent rounding *maps factorings that represent magnitudes larger than or equal to* $2^{e_{max}+1}$ *to the factoring of* $+/-\infty$ *for the reduced rounding modes RNE or RI and to the factoring of* $+/-x_{max}$ *in the reduced rounding mode RZ while restoring the sign of the factoring:*

$$exp\_rnd_{mode\star s}(s,e,f) = \begin{cases} (s,e_\infty,f_\infty) & \text{if } |val(s,e,f)| \geq 2^{e_{max}+1} \text{ AND } val(s,e,f) \notin \mathcal{SPE} \\ & \quad \text{AND } (mode \star s) \in \{RNE, RI\} \\ (s,e_{max},f_{max}) & \text{if } |val(s,e,f)| \geq 2^{e_{max}+1} \text{ AND} \\ & \quad val(s,e,f) \notin \mathcal{SPE} \text{ AND } (mode \star s) = RZ \\ (s,e,f) & \text{otherwise.} \end{cases} \tag{2.12}$$

In this definition, we distinguish between two different rounding functions for the significand: the aligned significand rounding and the normalized significand rounding. These two rounding functions differ by the choice of the rounding position for the significand. The aligned significand rounding assumes, that the significand is aligned, in such a way, that the significand rounding position is always at significand position $p-1$. This is the case for IEEE factorings. The situation is different for NF factorings. Because they are normalized even for denormalized values, the least significant bit position of the significand, which is the significand rounding position, could vary within a wide range. The variable rounding position vp of the normalized significand rounding takes care of this rounding position shift. In this way, normalized significand rounding is suitable for the significand rounding of NF factorings. The following two lemmas will show, how the computation of IEEE rounding on factorings can be based on the functions from definition 2.9 and proove the above argumentation in detail. Lemma 2.7 will consider the IEEE factoring and lemma 2.8 will consider the NF factoring of the rounded result.

Lemma 2.7 *For mode* $\in \{RZ, RNE, RI, RMI\}$*, the IEEE factoring* $iround_{mode}(s,e,f) : FACT(\mathbb{R}) \longrightarrow IEEEfact$*, with* $val(iround_{mode}(s,e,f) = r_{mode}(val(s,e,f))$*, can be computed by the sequence of a bounded normalization shift, aligned significand rounding, a post-normalization shift and exponent rounding:*

$$iround_{mode}(s,e,f) = exp\_rnd_{mode\star s}(post\_norm(a\_sig\_rnd_{mode\star s}(\lfloor \eta_{e_{min}} \rfloor(s,e,f)))).$$

Proof: Let $(s_1,e_1,f_1) = \lfloor \eta_{e_{min}} \rfloor(s,e,f)$, and $(s_2,e_2,f_2) = a\_sig\_rnd_{mode\star s}(s_1,e_1,f_1)$, and $(s_3,e_3,f_3) = post\_norm(s_2,e_2,f_2)$ and $(s_{ir},e_{ir},f_{ir}) = exp\_rnd_{mode\star s}(s_3,e_3,f_3)$.

We devide the proof into two steps. We will first show in part (a) of the proof, that the factoring (s_{ir},e_{ir},f_{ir}) has the value of the rounded result: $val(s_{ir},e_{ir},f_{ir}) = r_{mode}(val(s,e,f))$. In part (b), it will then be shown, that the factoring (s_{ir},e_{ir},f_{ir}) is an IEEE factoring, namely that $(s_{ir},e_{ir},f_{ir}) \in IEEEfact_{n,p}$.

(a) From the definitions of the bounded normalization shift and the post-normalization shift it follows directly, that these two shift operations do not change the value of the factoring, namely that $val(s_1,e_1,f_1) = val(s,e,f)$ and $val(s_3,e_3,f_3) = val(s_2,e_2,f_2)$. Thus, we have to show, that the combination of the aligned significand rounding and the exponent rounding implements IEEE rounding.

From the definition of the bounded normalization shift it also follows, that $e_1 = max\{e_{min}, e'\} = e''$, where e' is the exponent of the corresponding unbounded normalized factoring. Thus, we can write

$$\begin{aligned}
val(s_3,e_3,f_3) &= val(s_2,e_2,f_2) \\
&= val(a_sig_rnd_{mode\star s}(s_1,e_1,f_1)) \\
&= val(s_1,e_1,rnd_{mode\star s,p-1}(f_1)) \\
&= val(s_1,0,2^{e_1}\cdot rnd_{mode\star s,p-1}(f_1)) \\
&= val(s_1,0,rnd_{mode\star s,-e_1+p-1}(2^{e_1}\cdot f_1)) \\
&= val(0,0,rnd_{mode,-e_1+p-1}((-1)^{s_1}\cdot 2^{e_1}\cdot f_1)) \\
&= rnd_{mode,-e_1+p-1}(val(s_1,e_1,f_1)) \\
&= rnd_{mode,-e''+p-1}(val(s,e,f)).
\end{aligned}$$

Let $xr = rnd_{mode,-e''+p-1}(val(s,e,f))$. Because $val(s_3,e_3,f_3) \notin \mathcal{SPE}$ and $s = s_1 = s_2 = s_3 = s_{ir}$, we get for the value of the rounded result:

$$
\begin{aligned}
val(s_{ir}, e_{ir}, f_{ir}) &= val(exp\_rnd_{mode\star s}(s_3, e_3, f_3)) \\
&= \begin{cases} (-1)^s \cdot \infty & \text{if } |xr| \geq 2^{e_{max}+1} \text{ AND } (mode \star s) \in \{RNE, RI\} \\ (-1)^s \cdot x_{max} & \text{if } |xr| \geq 2^{e_{max}+1} \text{ AND } (mode \star s) = RZ \\ xr & \text{otherwise.} \end{cases} \\
&= r_{mode}(val(s, e, f))
\end{aligned}
$$

The last of these equations follows from lemma 2.6 and from table 2.3 for the combination of signs and reduced rounding modes. In this way step (a) of the proof is completed.

(b) We have to show, that $(s_{ir}, e_{ir}, f_{ir}) \in IEEEfact_{n,p}$. For the factorings of $+/-x_{max}$ and $+/-\infty$ in the exponent rounding definition this is obvious, so that we focus on the case of representable rounding results with $(s_{ir}, e_{ir}, f_{ir}) = (s_3, e_3, f_3)$ in the following. From part (a) we already know that $val(s_{ir}, e_{ir}, f_{ir}) \in \mathcal{FP}_{n,p}$. Because of this and because IEEE factoring representations are unique, it suffices to show that the following two conditions are fulfilled: (COND1) $(|val(s_{ir}, e_{ir}, f_{ir})| \geq 2^{e_{min}}) \Longrightarrow (f_{ir} \in [1, 2[)$; and
(COND2) $(|val(s_{ir}, e_{ir}, f_{ir})| < 2^{e_{min}}) \Longrightarrow (e_{ir} = e_{min})$.

For the remaining part of the proof we distinguish between: (i) $|val(s, e, f)| \geq 2^{e_{min}}$; and (ii) $|val(s, e, f)| < 2^{e_{min}}$. For both of these cases we have to show (COND1) and (COND2):

(i) Because $2^{e_{min}}$ is a representable number, it follows from $|val(s, e, f)| \geq 2^{e_{min}}$, that also the absolute value of the rounded result is larger than or equal to $2^{e_{min}}$. Hence, the condition (COND2) is always fullfilled for case (i).

From $|val(s, e, f)| \geq 2^{e_{min}}$, it follows, that the result of the bounded normalization shift is normalized, so that $f_1 \in [1, 2[$. After significand rounding we get a significand in the range $f_2 \in [1, 2]$, so that the post-normalization shift always outputs a normalized rounded significand $f_3 = f_{ir} \in [1, 2[$, and thus, also condition (COND1) is fulfilled.

(ii) From $|val(s, e, f)| < 2^{e_{min}}$ it follows, that the result of the bounded normalization shift is denormalized with $f_1 \in [0, 1[$ and $e_1 = e_{min}$. For $f_1 \in [0, 1[$ we get a rounded significand in the range $f_2 \in [0, 1]$, so the exponent rounding does no change and we get the exponent of the rounded result $e_{ir} = e_3 = e_{min}$. In this way condition (COND2) is fullfilled. From $val(s_{ir}, e_{ir}, f_{ir}) \geq 2^{e_{min}}$, $f_{ir} \in [0, 1]$ and $e_{ir} = e_{min}$, it follows that $f_{ir} = 1$ is normalized, so that also (COND1) is fulfilled.

□

Lemma 2.8 *For $mode \in \{RZ, RNE, RI, RMI\}$ the NF factoring $nround_{mode}(s, e, f) : FACT(\mathbb{R}) \longrightarrow NFfact$, which has the value $val(nround_{mode}(s, e, f)) = r_{mode}(val(s, e, f))$, can be computed by the sequence of an unbounded normalization shift, normalized significand rounding, another unbounded normalization shift and exponent rounding:*

$$nround_{mode}(s, e, f) = exp\_rnd_{mode\star s}(\eta(n\_sig\_rnd_{mode\star s}(\eta(s, e, f)))).$$

Proof: Let $(s_{1n}, e_{1n}, f_{1n}) = \eta(s, e, f)$, and let $(s_{2n}, e_{2n}, f_{2n}) = n\_sig\_rnd_{mode\star s}(s_{1n}, e_{1n}, f_{1n})$. In addition to this we use the notation from the previous lemma.

We devide the proof into the following two steps: We will first show in part (a) that the factoring $(s_{nr}, e_{nr}, f_{nr}) = exp\_rnd_{mode\star s}(\eta(n\_sig\_rnd_{mode\star s}(\eta(s, e, f))))$ has the value of the rounded result: $val(s_{nr}, e_{nr}, f_{nr}) = r_{mode}(val(s, e, f))$. In part (b) of the proof,

it will then be shown that the factoring (s_{nr}, e_{nr}, f_{nr}) is a NF factoring, namely that $(s_{nr}, e_{nr}, f_{nr}) \in NFfact_{n,p}$.

(a) The normalization shifts do not change the value of a factoring and the value of the exponent rounding only depends on the value of its input factoring. Hence, for the proof of $val(s_{nr}, e_{nr}, f_{nr}) = val(s_{ir}, e_{ir}, f_{ir}) = r_{mode}(val(s, e, f))$, it suffices to show that

$$val(s_{2n}, e_{2n}, f_{2n}) = n\_sig\_round(\eta(s, e, f)) = a\_sig\_round(\lceil \eta_{e_{min}} \rceil(s, e, f) = val(s_2, e_2, f_2).$$

For the proof of this equation, we distinguish between: (i) $|val(s, e, f)| \geq 2^{e_{min}}$; and (ii) $|val(s, e, f)| < 2^{e_{min}}$.

(i) For $|val(s, e, f)| \geq 2^{e_{min}}$,the output of the bounded normalization shift is normalized, so that $(s_{1n}, e_{1n}, f_{1n}) = (s_1, e_1, f_1)$ and $e_{1n} = e' > e_{min}$. Hence, in the definition of normalized significand rounding, the variable rounding position becomes $vp = (p-1) - max\{0, e_{min} - e\} = p - 1$ and agrees with the rounding position $p - 1$ of the aligned significand rounding.Thus, also the output factorings of both significand rounding functions are the same: $val(s_{2n}, e_{2n}, f_{2n}) = val(s_2, e_2, f_2)$.

(ii) Because for $|val(s, e, f)| = 0$, none of the 2 steps in both computations change the factoring, we only deal with non-zero numbers in the following. Since $|val(s, e, f)| < 2^{e_{min}}$, we get for the exponent of the unbounded normalized factoring $e_{1n} = e' < e_{min}$. For the same reason, the output of the bounded normalization shift is denormalized with $e_1 = e_{min}$, so that the overall rounding position of aligned significand rounding becomes $-e_{min} + p - 1$. In the case of normalized significand rounding, the variable significand rounding position is $vp = (p-1) - max\{0, e_{min} - e_{1n}\} = (p-1) - e_{min} - e_{1n}$. In the combination with the exponent factor $2^{e_{1n}}$, we get the overall rounding position $-e_{min} + p - 1$ also in this case.

(b) We have to show, that (s_{nr}, e_{nr}, f_{nr}) is a NF factoring. Hence, (s_{nr}, e_{nr}, f_{nr}) has to be normalized for all non-zero numbers. After the second unbounded normalization shift, we get a normalized significand f_{nr} in the range $f_{nr} \in [1, 2[$ for all non-zero representable numbers. Because there is no condition on the NF factoring of a zero and the factoring representations of $+/- \infty$ and $+/- x_{max}$ are defined in the exponent rounding output to be normalized, (s_{nr}, e_{nr}, f_{nr}) is a NF factoring in any case. □

We distinguish between rounding in single precision and double precision by the choice of the corresponding values of: p, n, e_{min}, e_{max}, f_{max}, e_∞ and f_∞.

For the definition of exceptions, and some correctness proofs, it is helpful to have a rounding function $\breve{r}$ with an unbounded exponent range. For a factoring (s, e, f) with $f \neq 0$, the new rounding $\breve{r}$ is defined by:

$$\breve{r}_{mode}(s, e, f) = post\_norm(a\_sig\_rnd_{mode*s}(\eta(s, e, f))). \tag{2.13}$$

2.4 Special Cases

The IEEE standard defines six exceptions, that can occur, when a floating-point operation is executed: *overflow*, *underflow*, *inexact*, *invalid*, *division by zero* and *unimplemented FP operation*. The occurrences of these exceptions are signaled by the six IEEE flags OVF, UNF, INX, INV, DVZ, and UFO. Exept for the combinations of INX with OVF or UNF, at most one FP exception can occur during an operation.

The *trap handler enable*-bits: OVF_EN, UNF_EN, INX_EN, INV_EN, and DVZ_EN are set by the user. For an unimplemented FP operation, the corresponding trap handler is always enabled. If a trap handler is enabled, i.e., the trap handler enable bit is active, the occurrence of the corresponding exception starts the execution of an exception trap routine. With a disabled trap handler, a result is returned immedately even for the occurance of the corresponding exception. If INX_EN and OVF_EN or UNF_EN are enabled and both exceptions occur during the same operation, the OVF or UNF-trap has precedence over the execution of the INX-trap.

After describing the IEEE flags and exception handling in detail, we will overview the results of operations on special values, that have a strong relationship to exceptions. Finally, we will give a general summary on the computations for each IEEE operation.

2.4.1 IEEE Flags

We consider an arithmetic floating-point operation $op \in$ { ADD/SUB, MULT, DIV, SQRT, CONV} operating on finite operands. This operation op delivers the exact result $exact_{op}$, that can be represented by the factoring (s_{ex}, e_{ex}, f_{ex}). We denote the value of the rounded result of the operation by $bro = val(round_{mode}((s_{ex}, e_{ex}, f_{ex}))$, and the value of the result that is rounded with an unbounded exponent range by: $uro = val(\breve{r}_{mode}(s_{ex}, e_{ex}, f_{ex}))$.

Overflow The overflow flag signals, that the magnitude of the unbounded rounded result is bigger than the magnitude of the largest representable number:

$$|uro| > |x_{max}| = (2 - 2^{-p+1}) \cdot 2^{e_{max}}.$$

Underflow The conditions for an underflow differ depending on the value of UNF_EN. They are based on the definitions of *tininess* and *loss-of-accuracy*:

- There are two possible definitions for tininess given by the standard: A result is *tiny-before-rounding*, if $0 \neq |exact_{op}| < 2^{e_{min}}$, and *tiny-after-rounding*, if $0 \neq |uro| < 2^{e_{min}}$.
- Similarly, the standard provides two loss-of-accuracy definitions: *Loss-of-accuracy-a* occurs, if $exact_{op} \neq 0$ AND $uro \neq bro$, and *loss-of-accuracy-b* occurs, if $bro \neq exact_{op}$.

For both tininess and loss-of-accuracy, the implementor may choose one of the two definitions provided by the standard, but these choices have to be the same for all operations and precisions. Based on these conditions, the *underflow* exception is defined by:

- If UNF_EN = 0, then an underflow occurs if tininess and loss-of-accuracy occurs.
- If UNF_EN = 1, then an underflow occurs if tininess occurs.

Definition 2.10 *We define the boolean function $TINY(s, e, f)$, that delivers the boolean value corresponding to the tininess condition $(0 \neq |val(s, e, f)| < 2^{e_{min}})$.*

Lemma 2.9 *For $0 \neq f$ and the normalized factoring $(s, e', f') = \eta(s, e, f)$, the number $x = val(s, e, f)$ is tiny, signaled by $(TINY(s, e, f) = 1)$, iff $(e' < e_{min})$.*

Proof: Because (s, e', f') is normalized, $1 \leq f' < 2$ and $2^{e'} \leq |val(s, e, f)| < 2^{e'+1}$. Thus, the tininess condition can be written as $e' + 1 \leq e_{min}$. This is equivalent to $e' < e_{min}$. □

Inexact An inexact exception occurs if $bro \neq exact_{op}$. This is exactly the *loss-of-accuracy-b* condition and includes the case of an overflow.

Division by Zero The DVZ flag signals, that the second operand of a division equals $+0$ or -0 and the first operand is a finite non-zero number.

Invalid The INV flag is signaled:

1. for any operation, where at least one operand is a signaling NaN,
2. for effective subtractions of two infinities,
3. for the multiplication of 0 and infinity regardless of the signs,
4. for divisions of $0/0$ or ∞/∞ regardless of the signs,
5. for remainders, where the first operand is infinite or the second is a zero,
6. for the square root of an operand less than zero,
7. for comparisons with a condition code that demands 'invalid if unordered'(see table 2.2) and the operands are unordered.

Unimplemented FP operation The UFO flag is signaled for any FP operation that is not implemented in hardware.

2.4.2 Exceptions

If an exception is recognized during the computation of an operation, the corresponding IEEE flag(s) are set to 1. The IEEE flags are sticky, i.e., if they have been set once, they stay active till they are cleared by the user. The further computation depends on the value of the corresponding trap handler enable bit:

Disabled Trap Handler In most cases the delivered result has to be the correctly rounded result *bro*, but for a division by zero, a correctly signed infinity, and for an invalid exception, a qNaN has to be delivered. Moreover, for operations on special values and zeros, the results are summarized in the next paragraph. With the computation of the result, the execution of the operation is finished.

Enabled Trap Handler The operation starts the corresponding trap routine, that is responsible for the further computations. The operands for the trap routine are specified by the standard and differ from the above results depending on the exception:

Each trap should get the operation type of the operation that caused the exception, the information, which exception occured and the destination's format. In the case of a trapped invalid or a trapped division by zero, the operand values have to be accessible to the trap routine. In a trapped inexact exception the correctly rounded result is given to the trap routine. In trapped overflows and trapped underflows *exponent wrapping* has to be computed, before the result is fed to the trap routine:

- **Trapped Overflow.** If a trapped overflow occurs, then the wrapped exponent $e - \alpha$ is used and a factoring of $r_{mode}(exact_{op} \cdot 2^{-\alpha})$ is delivered to the trap routine, where $\alpha = 3 \cdot 2^{n-2}$. We will consider the corresponding IEEE factoring $iround_{mode}(s, e - \alpha, f)$ or the corresponding NF factoring $nround_{mode}(s, e - \alpha, f))$
  The magnitude of all exact overflow results is larger than $lbound_{ovf} = 2^{e_{max}} = 2^{2^{n-1}-1}$. An upper bound on the magnitude of exact results is found looking at the case, that the largest representable number, that is smaller than $2 \cdot 2^{e_{max}} = 2^{2^{n-1}}$, is divided by the representable number with the smallest magnitude $2^{e_{min}-p+1} = 2^{-2^{n-1}+2-p+1}$:

  $$2^{2^{n-1}} / 2^{-2^{n-1}+2-p+1} = 2^{2^{n-1}+2^{n-1}-2+p-1} = 2^{2^n+p-3}$$

  Thus, the magnitude of all exact results of the standard's operations on representable numbers is smaller than $ubound_{ovf} = 2^{2^n+p-3}$. The exponent wrapping by $-\alpha$ reduces the lower bound on the magnitude of exact overflow results to

  $$lbound_{ovf} \cdot 2^{-\alpha} = 2^{2^{n-1}-1-3\cdot 2^{n-2}} = 2^{-2^{n-2}-1} > 2^{-2^{n-1}+2} = 2^{e_{min}}$$

  and the upper bound on the magnitude of exact overflow results to

  $$ubound_{ovf} \cdot 2^{-\alpha} = 2^{2^n+p-3-3\cdot 2^{n-2}} = 2^{2^{n-2}+p-3} < 2^{2^{n-1}-1} = 2^{e_{max}}.$$

  Therefore, after exponent wrapping all overflow results have values of normalized numbers.

- **Trapped Underflow.** If a trapped underflow occurs, then the wrapped exponent $e + \alpha$ is used and a factoring of $r_{mode}(exact_{op} \cdot 2^{\alpha})$ is delivered to the trap routine, where $\alpha = 3 \cdot 2^{n-2}$. We will consider the corresponding IEEE factoring $iround_{mode}(s, e + \alpha, f)$ or the corresponding NF factoring $nround_{mode}(s, e + \alpha, f))$
  The magnitude of all exact underflow results is smaller than $ubound_{unf} = 2^{e_{min}} = 2^{-2^{n-1}+2}$. A lower bound on the magnitude of exact results is found looking at the case, that the representable number with the smallest magnitude $2^{e_{min}-p+1} = 2^{-2^{n-1}-p+3}$ is multiplied by itself:

  $$2^{-2^{n-1}-p+3} \cdot 2^{-2^{n-1}-p+3} = 2^{-2^n-2p+6}.$$

  Thus, the magnitude of all exact results of the standard's operations on representable numbers is larger than or equal to $lbound_{unf} = 2^{-2^n-2p+6}$.
  The exponent wrapping by α increases the lower bound on the magnitude of exact underflow results to

  $$lbound_{unf} \cdot 2^{\alpha} = 2^{-2^n-2p+6+3\cdot 2^{n-2}} = 2^{-2^{n-2}-2p+6} > 2^{-2^{n-1}+2} = 2^{e_{min}}$$

ADD	$+0$	-0	$+y$	$-y$	$+\infty$	$-\infty$	qNaN2	sNaN
$+0$	$+0$	$+0$(–0 RMI)	$+y$	$-y$	$+\infty$	$-\infty$	qNaN2	qNaN
-0	$+0$(–0 RMI)	-0	$+y$	$-y$	$+\infty$	$-\infty$	qNaN2	qNaN
$+x$	$+x$	$+x$	*n.s.*	*n.s.*	$+\infty$	$-\infty$	qNaN2	qNaN
$-x$	$-x$	$-x$	*n.s.*	*n.s.*	$+\infty$	$-\infty$	qNaN2	qNaN
$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	qNaN	qNaN2	qNaN
$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	qNaN	$-\infty$	qNaN2	qNaN
qNaN1	qNaN1	qNaN1	qNaN1	qNaN1	qNaN1	qNaN1	qNaN1/2	qNaN
sNaN	qNaN	qNaN	qNaN	qNaN	qNaN	qNaN	qNaN	qNaN

Table 2.4: Results of additions on special values.

ADD	$+/-0$	$+y$	$-y$	$+\infty$	$-\infty$	qNaN2	sNaN
$+/-0$	*no*	*no*	*no*	*no*	*no*	*no*	INV
$+x$	*no*	OVF/UNF/INX/*no*	UNF/INX/*no*	*no*	*no*	*no*	INV
$-x$	*no*	UNF/INX/*no*	OVF/UNF/INX/*no*	*no*	*no*	*no*	INV
$+\infty$	*no*	*no*	*no*	*no*	INV	*no*	INV
$-\infty$	*no*	*no*	*no*	INV	*no*	*no*	INV
qNaN1	*no*	*no*	*no*	*no*	*no*	*no*	INV
sNaN	INV	INV	INV	INV	INV	INV	INV

Table 2.5: Exceptions of additions.

and the upper bound on the magnitude of exact underflow results to

$$ubound_{unf} \cdot 2^{\alpha} = 2^{-2^{n-1}+2+3\cdot 2^{n-2}} = 2^{2^{n-2}+2} < 2^{2^{n-1}-1} = 2^{e_{max}}.$$

Therefore, after exponent wrapping also all underflow results have values of normalized numbers.

Corollary 2.10 *After exponent wrapping all results of operations on representable numbers have values of normalized numbers.*

2.4.3 Operations on Special Values

In this section, we summarize the results of additions (see table 2.4, for subtractions the second operand has to be multiplied by -1), multiplications (see table 2.6), divisions (see table 2.8), and square roots (see table 2.10) on special values and zeros and list the possible exceptions (see table 2.5,2.7,2.9, and 2.10).

In the tables, different possibilities of one entry are separated by '/', 'n.s.' means that the corresponding entry can not be specified in general, 'no' means, that no exception occurs, and '(-0 RMI)' means, that the result is -0 if the rounding mode is RMI. Because the representation of qNaNs is not unique, we enumerate such operands by qNaN1 and qNaN2.

MULT	$+0$	-0	$+y$	$-y$	$+\infty$	$-\infty$	qNaN2	sNaN
$+0$	$+0$	-0	$+0$	-0	qNaN	qNaN	qNaN2	qNaN
-0	-0	$+0$	-0	$+0$	qNaN	qNaN	qNaN2	qNaN
$+x$	$+0$	-0	*n.s.*	*n.s.*	$+\infty$	$-\infty$	qNaN2	qNaN
$-x$	-0	$+0$	*n.s.*	*n.s.*	$-\infty$	$+\infty$	qNaN2	qNaN
$+\infty$	qNaN	qNaN	$+\infty$	$-\infty$	$+\infty$	$-\infty$	qNaN2	qNaN
$-\infty$	qNaN	qNaN	$-\infty$	$+\infty$	$-\infty$	$+\infty$	qNaN2	qNaN
qNaN1	qNaN1	qNaN1	qNaN1	qNaN1	qNaN1	qNaN1	qNaN1/2	qNaN
sNaN	qNaN	qNaN	qNaN	qNaN	qNaN	qNaN	qNaN	qNaN

Table 2.6: Results of multiplications on special values.

MULT	$+/-0$	$+/-y$	$+/-\infty$	qNaN2	sNaN
$+/-0$	*no*	*no*	INV	*no*	INV
$+/-x$	*no*	OVF/UNF/INX/*no*	*no*	*no*	INV
$+/-\infty$	INV	*no*	*no*	*no*	INV
qNaN1	*no*	*no*	*no*	*no*	INV
sNaN	INV	INV	INV	INV	INV

Table 2.7: Multiplication exceptions.

DIV	$+0$	-0	$+y$	$-y$	$+\infty$	$-\infty$	qNaN2	sNaN
$+0$	qNaN	qNaN	$+0$	-0	$+0$	-0	qNaN2	qNaN
-0	qNaN	qNaN	-0	$+0$	-0	$+0$	qNaN2	qNaN
$+x$	$+\infty$	$-\infty$	*n.s.*	*n.s.*	$+0$	-0	qNaN2	qNaN
$-x$	$-\infty$	$+\infty$	*n.s.*	*n.s.*	-0	$+0$	qNaN2	qNaN
$+\infty$	$+\infty$	$-\infty$	$+\infty$	$-\infty$	qNaN	qNaN	qNaN2	qNaN
$-\infty$	$-\infty$	$+\infty$	$-\infty$	$+\infty$	qNaN	qNaN	qNaN2	qNaN
qNaN1	qNaN1	qNaN1	qNaN1	qNaN1	qNaN1	qNaN1	qNaN1/2	qNaN
sNaN	qNaN	qNaN	qNaN	qNaN	qNaN	qNaN	qNaN	qNaN

Table 2.8: results of division on special values.

DIV	$+/-0$	$+/-y$	$+/-\infty$	qNaN	sNaN
$+/-0$	INV	*no*	*no*	*no*	INV
$+/-x$	DVZ	OVF/UNF/INX/*no*	*no*	*no*	INV
$+/-\infty$	*no*	*no*	INV	*no*	INV
qNaN	*no*	*no*	*no*	*no*	INV
sNaN	INV	INV	INV	INV	INV

Table 2.9: Division exceptions.

SQRT	$+0$	-0	$+y$	$-y$	$+\infty$	$-\infty$	qNaN1	sNaN
result	$+0$	-0	*n.s.*	qNaN	$+\infty$	qNaN	qNaN1	qNaN
exception	*no*	*no*	INX/*no*	*INV*	*no*	*INV*	*no*	*INV*

Table 2.10: Results and exceptions of squareroots on special values.

2.4.4 Summary of IEEE Computations

In the previous sections, various aspects of the computations for IEEE operations were described separately. In this section all aspects of the computations will be summarized for each IEEE operation.

In the previous section about the computation on special value and zero operands x and y, we saw, that for these cases the result can have only one of a few possible values, namely, $+/-0$, $+/-\infty$, $qNaN$, x, y. If none of these special cases occurs, the IEEE rounded result with or without exponent wrapping should be output.

Assume, that we have a factoring (s_{rc}, e_{rc}, f_{rc}), that represents the exact result $exact_{op}$ for $op \in \{ADD/SUB, MULT, DIV, SQRT, CONV\}$ and for non-zero representable operands. We define five special condition flags SCQNAN, SCINF, SCX, SCY, and SCZERO that correspond to the occurance of the special cases results: $qNaN$, $+/-\infty$, $x = val(sa, ea, fa)$, $y = val(sb, eb, fb)$, and $+/-0$. Moreover, the exponent wrapping constant is defined by:

$$wec = \begin{cases} -\alpha & \text{if OVF AND OVF\_EN} \quad \text{(wrapped overflow)} \\ +\alpha & \text{if UNF AND UNF\_EN} \quad \text{(wrapped underflow)} \\ 0 & \text{otherwise,} \end{cases} \tag{2.14}$$

Based on these definitions, the IEEE factoring of the final result of an IEEE operation can be selected by:

$$(s_{ifnl}, e_{ifnl}, f_{ifnl}) = \begin{cases} (0, e_{qNaN}, f_{qNaN}) & \text{if SCQNAN} \\ (s_{inf}, e_{\infty}, f_{\infty}) & \text{if SCINF} \\ (sa, ea, fa) & \text{if SCX} \\ (sb, eb, fb) & \text{if SCY} \\ (s_0, e_0, 0) & \text{if SCZERO} \\ iround(s_{rc}, e_{rc} + wec, f_{rc}) & \text{otherwise} \end{cases} \tag{2.15}$$

The corresponding NF factoring of the rounded IEEE operation result is given by (see lemma 2.8):

$$(s_{nfnl}, e_{nfnl}, f_{nfnl}) = \begin{cases} (0, e_{qNaN}, f_{qNaN}) & \text{if SCQNAN} \\ (s_{inf}, e_{\infty}, f_{\infty}) & \text{if SCINF} \\ (sa, ea, fa) & \text{if SCX} \\ (sb, eb, fb) & \text{if SCY} \\ (s_0, e_0, 0) & \text{if SCZERO} \\ nround(s_{rc}, e_{rc} + wec, f_{rc}) & \text{otherwise} \end{cases} \tag{2.16}$$

Definition 2.11 *We extend the definition of the function iround on factorings of special values* $(s_{sp}, e_{sp}, f_{sp}) \in SPEfact$ *by the identity* $iround(s_{sp}, e_{sp}, f_{sp}) = (s_{sp}, e_{sp}, f_{sp})$.

Also this extension is included in the computation sequence for iround from lemma 2.7. The reason for this is, that we define the factorings of special values to be exact and normalized and with an exponent of $e_{max} + 1$. *Thus, the first three steps of the bounded normalization shift, the significand rounding and the post-normalization shift do not change the factorings of special values. Also in the last step the factoring is not changed, because the definition of the exponent rounding in equation 2.12 already includes this case.*

Note, that the exponent wrapping constant is 0 for operations on special values, because no overflow or underflow can occur for them. Thus, with the extension of the definition

of the function *iround* and the definition of the exact result factoring:

$$(s_{ex}, e_{ex}, f_{ex}) \quad = \quad \begin{cases} (0, e_{qNaN}, f_{qNaN}) & \text{if SCQNAN} \\ (s_{inf}, e_{\infty}, f_{\infty}) & \text{if SCINF} \\ (sa, ea, fa) & \text{if SCX} \\ (sb, eb, fb) & \text{if SCY} \\ (s_0, e_0, 0) & \text{if SCZERO} \\ (s_{rc}, e_{rc}, f_{rc}) & \text{otherwise,} \end{cases} \tag{2.17}$$

for all cases the IEEE factoring of the final result $(s_{ifnl}, e_{ifnl}, f_{ifnl})$ can be described by:

$$(s_{ifnl}, e_{ifnl}, f_{ifnl}) \quad = \quad iround(s_{ex}, e_{ex} + wec, f_{ex}) \tag{2.18}$$

With the same extension of the function *nround* and a similar argumentation for the computation sequence for *nround* from lemma 2.8, for all cases the corresponding NF factoring of the final result $(s_{nfnl}, e_{nfnl}, f_{nfnl})$ is computed by

$$(s_{nfnl}, e_{nfnl}, f_{nfnl}) \quad = \quad nround(s_{ex}, e_{ex} + wec, f_{ex}) \tag{2.19}$$

The equations for the special condition flags and the sign s_{sc} can be easily extracted for each IEEE operation from the tables on the special value results in the previous section. With the factorings of the input operands (sa, ea, fa) and (sb, eb, fb), and the following conditions on these factorings

$$\begin{array}{lcl@{\qquad}lcl} \text{ZEROA} & \Longleftrightarrow & (|val(sa, ea, fa)| = 0) & \text{ZEROB} & \Longleftrightarrow & (|val(sb, eb, fb)| = 0) \\ \text{INFA} & \Longleftrightarrow & (|val(sa, ea, fa)| = \infty) & \text{INFB} & \Longleftrightarrow & (|val(sb, eb, fb)| = \infty) \\ \text{QNANA} & \Longleftrightarrow & (val(sa, ea, fa) = qNaN) & \text{QNANB} & \Longleftrightarrow & (val(sb, eb, fb) = qNaN) \\ \text{SNANA} & \Longleftrightarrow & (val(sa, ea, fa) = sNaN) & \text{SNANB} & \Longleftrightarrow & (val(sb, eb, fb) = sNaN) \\ \text{ZERO}_{rc} & \Longleftrightarrow & (f_{rc} = 0), & & & \end{array}$$

we get the following equations:

- addition/subtraction:

$$\begin{aligned} \text{SCQNAN} &= \text{SNANA} \vee \text{SNANB} \vee (\text{INFA} \wedge \text{INFB} \wedge (\text{SA} \otimes \text{SB})) && (2.20) \\ \text{SCX} &= (\text{QNANA} \wedge \overline{\text{SNANB}}) \vee (\text{ZEROB} \wedge \overline{\text{ZEROA}} \wedge \overline{\text{SNANA}}) && (2.21) \\ \text{SCY} &= (\text{QNANB} \wedge \overline{\text{QNANA}} \wedge \overline{\text{SNANA}}) \vee (\text{ZEROA} \wedge \overline{\text{ZEROB}} \wedge \overline{\text{SNANB}}) && (2.22) \\ \text{SCINF} &= \overline{\text{SCQNAN}} \wedge \overline{\text{SCX}} \wedge \overline{\text{SCY}} \wedge (\text{INFA} \vee \text{INFB}) && (2.23) \\ \text{SCZERO} &= (\text{ZEROA} \wedge \text{ZEROB}) \vee \text{ZERO}_{rc} && (2.24) \\ \text{S}_{inf} &= (\text{SA} \wedge \text{INFA}) \vee (\text{SB} \wedge \text{INFB}) && (2.25) \\ \text{S}_0 &= (is\_RMI \wedge (\text{SA} \oplus \text{SB} \oplus \text{SOP})) \vee (\text{SA} \wedge (\text{SB} \oplus \text{SOP})) && (2.26) \end{aligned}$$

- multiplication:

$$\begin{aligned} \text{SCQNAN} &= \text{SNANA} \vee \text{SNANB} \vee (\text{INFA} \wedge \text{ZEROB}) \vee (\text{INFB} \wedge \text{ZEROA}) && (2.27) \\ \text{SCX} &= \text{QNANA} \wedge \overline{\text{SNANB}} && (2.28) \\ \text{SCY} &= \text{QNANB} \wedge \overline{\text{SCX}} \wedge \overline{\text{SNANA}} && (2.29) \\ \text{SCINF} &= \overline{\text{SCQNAN}} \wedge \overline{\text{SCX}} \wedge \overline{\text{SCY}} \wedge (\text{INFA} \vee \text{INFB}) && (2.30) \\ \text{SCZERO} &= (\text{ZEROA} \wedge \text{ZEROB}) \vee \overline{(\text{SCQNAN} \vee \text{SCX} \vee \text{SCY})} && (2.31) \\ \text{S}_{inf} &= \text{SA} \otimes \text{SB} && (2.32) \\ \text{S}_0 &= \text{S}_{inf} && (2.33) \end{aligned}$$

- division:

$$\begin{aligned}
\text{SCQNAN} &= \text{SNANA} \vee \text{SNANB} \vee (\text{ZEROA} \wedge \text{ZEROB}) \vee (\text{INFA} \wedge \text{INFB}) && (2.34)\\
\text{SCX} &= \text{QNANA} \wedge \overline{\text{SNANB}} && (2.35)\\
\text{SCY} &= \text{QNANB} \wedge \overline{\text{SCX}} \wedge \overline{\text{SNANA}} && (2.36)\\
\text{SCINF} &= \overline{\text{SCQNAN}} \wedge \overline{\text{SCX}} \wedge \overline{\text{SCY}} \wedge (\text{INFA} \vee \text{ZEROB}) && (2.37)\\
\text{SCZERO} &= \overline{\text{SCQNAN}} \wedge \overline{\text{SCX}} \wedge \overline{\text{SCY}} \wedge (\text{INFB} \vee \text{ZEROA}) && (2.38)\\
\text{S}_{inf} &= \text{SA} \otimes \text{SB} && (2.39)\\
\text{S}_0 &= \text{S}_{inf} && (2.40)
\end{aligned}$$

- square-root:

$$\begin{aligned}
\text{SCQNAN} &= (\overline{\text{ZEROA}} \wedge \text{SA} \wedge \overline{\text{QNANA}}) \vee \text{SNANA} && (2.41)\\
\text{SCX} &= \text{QNANA} && (2.42)\\
\text{SCY} &= 0 && (2.43)\\
\text{SCINF} &= \text{INFA} \wedge \overline{\text{SA}} && (2.44)\\
\text{SCZERO} &= \text{ZEROA} && (2.45)\\
\text{S}_{inf} &= 0 && (2.46)\\
\text{S}_0 &= \text{SA} && (2.47)
\end{aligned}$$

This completes the specification of the IEEE operations. In the next section we will provide some methodologies, by that the rounding computations can be simplified.

2.5 Rounding Computation Utilities

2.5.1 Representatives

Definition 2.12 *For an integer γ, two real numbers x_1 and x_2 are γ-equivalent, denoted by $x_1 \stackrel{\gamma}{=} x_2$ if there exists an integer q such that $x_1, x_2 \in]q \cdot 2^{-\gamma}, (q+1) \cdot 2^{-\gamma}[$ or $x_1 = x_2 = q \cdot 2^{-\gamma}$.*

Thus, the binary representation of γ-equivalent reals must agree in the first γ positions to the right of the binary point. We choose the γ-representatives of the equivalence classes as follows:

Definition 2.13 *Let x denote a real number and γ an integer. Let q denote the integer satisfying: $q2^{-\gamma} \leq x < (q+1)2^{-\gamma}$. The γ-representative of x, denoted by $rep_\gamma(x)$, is defined by:*

$$rep_\gamma(x) = \begin{cases} q2^{-\gamma} & \text{if } x = q2^{-\gamma} \\ (q+0.5)2^{-\gamma} & \text{if } x \in]q2^{-\gamma}, (q+1)2^{-\gamma}[. \end{cases}$$

The γ-representatives form integral multiples of $2^{-\gamma-1}$. Thus, they can be represented by $\gamma+1$ bits to the right of the binary point. Note, that the least significant bit in this representation indicates whether the corresponding equivalence class is a single point or an open interval. The following lemma describes, how the γ-representative of a binary number can be computed.

Lemma 2.11 *With $f \in [0, 2[$, integers $0 < \gamma < k$, so that f is a multiple of 2^{-k}, and $\text{F}[0:k] = bin^0_{-k}(f)$, we define:*

$$\begin{aligned} sticky\_bit_\gamma(f) &= \text{OR}(\text{F}[\gamma+1:k]) \\ sticky_\gamma(f) &= <\text{F}[0:\gamma]>_{neg} + sticky\_bit_\gamma(f) \cdot 2^{-\gamma-1}. \end{aligned}$$

The γ-representative of f is then given by

$$rep_\gamma(f) = sticky_\gamma(f).$$

Proof: The binary representation $rep_\gamma(f)$ is identical to $\text{F}[0:k]$ up to the position with weight $2^{-\gamma}$. If f is an integral multiple of $2^{-\gamma}$, then $f = rep_\gamma(f) =< \text{F}[0:\gamma] >_{neg}$ and $\text{F}[\gamma+1:k]$ is all zeros, and so is $sticky\_bit_\gamma(f)$. If f is not an integral multiple of $2^{-\gamma}$, then $\text{F}[\gamma+1:k]$ is not all zeros, and therefore, $sticky\_bit_\gamma(f) = 1$, and $rep_\gamma(f) = sticky_\gamma(f)$, as required. □

Lemma 2.12 *For integers γ_1, γ_2, with $0 < \gamma_2 < \gamma_1$, and $f = <\text{F}[0:k]>_{neg}$ (i) one can derive $rep_{\gamma_2}(f)$ from $rep_{\gamma_1}(f) = <\text{REP1}[0:\gamma_1+1]>_{neg}$ by*

$$rep_{\gamma_2}(f) = <(\text{REP1}[0:\gamma_2], \text{OR}(\text{REP1}[\gamma_2+1:\gamma_1+1]))>_{neg},$$

(ii) $\quad f_x \stackrel{\gamma_1}{=} f_y \quad \longrightarrow \quad f_x \stackrel{\gamma_2}{=} f_y.$

Proof: (i) The bits $sticky\_bit_{\gamma_1}(f)$ and $sticky\_bit_{\gamma_2}(f)$ are defined by

$$\begin{aligned} sticky\_bit_{\gamma_1}(f) &= \text{OR}(\text{F}[\gamma_1+1:k]) \\ sticky\_bit_{\gamma_2}(f) &= \text{OR}(\text{F}[\gamma_2+1:k]). \end{aligned}$$

Substitution of the $sticky\_bit_{\gamma_1}(f)$-definition in the $sticky\_bit_{\gamma_2}(f)$-definition, yields

$$sticky\_bit_{\gamma_2}(f) = \text{OR}(\text{F}[\gamma_2 + 1 : \gamma_1], sticky\_bit_{\gamma_1}(f)).$$

Because $\text{REP1}[0 : \gamma_1] = \text{F}[0 : \gamma_1]$ and $\text{REP1}[\gamma_1 + 1] = sticky\_bit_{\gamma_1}(f)$ by the definition of $rep_{\gamma_1}(f)$, we have

$$sticky\_bit_{\gamma_2}(f) = \text{OR}(\text{REP1}[\gamma_2 + 1 : \gamma_1 + 1])$$

and part (i) of the lemma follows.
(ii) We have $rep_{\gamma_1}(f_x) = rep_{\gamma_1}(f_y)$. In the first part was shown, that γ_2-representatives can be computed from γ_1-representatives. Therefore, $rep_{\gamma_2}(f_x) = rep_{\gamma_2}(f_y)$, and part (ii) of the lemma follows. □

For $mode \in \{RZ, RNE, RI\}$ and $f' = rep_\gamma(f)$ one can additionaly show the following equations:

$$f_x \stackrel{\gamma}{=} f_y \quad \text{iff} \quad 2^k \cdot f_x \stackrel{\gamma-k}{=} f_y \cdot 2^k \text{ for an integer } k \tag{2.48}$$

$$rnd_{mode,\gamma-1}(f) \quad = \quad rnd_{mode,\gamma-1}(f') \tag{2.49}$$

$$rnd_{mode,\gamma-1}(f) = f \quad \text{iff} \quad rnd_{mode,\gamma-1}(f') = f' \tag{2.50}$$

$$rnd_{mode,\gamma-1}(f') = f' \quad \text{iff} \quad f' = q \cdot 2^{-(\gamma-1)} \text{ for an integer } q \tag{2.51}$$

$$f' = f \quad \text{iff} \quad rnd_{mode,\gamma-1}(f') = f. \tag{2.52}$$

For the computation of rounded factorings we will use the following properties.

Lemma 2.13 *Let (s_x,e_x,f_x) and (s_y,e_y,f_y) be two factorings and let (s'_x,e'_x,f'_x) and (s'_y,e'_y,f'_y) be the corresponding bounded normalized factorings: $(s'_x, e'_x, f'_x) = \lfloor \eta_{e_{min}} \rfloor (s_x, e_x, f_x)$ and $(s'_y, e'_y, f'_y) = \lfloor \eta_{e_{min}} \rfloor (s_y, e_y, f_y)$. If the values of (s_x, e_x, f_x) and (s_y, e_y, f_y) are $(p - e'_x)$-equivalent:*

$$x = val(s_x, e_x, f_x) = val(s'_x, e'_x, f'_x) \stackrel{p-e'_x}{=} val(s'_y, e'_y, f'_y) = val(s_y, e_y, f_y) = y,$$

then (i) $s'_x = s'_y$, $e'_x = e'_y$, and $f'_x \stackrel{p}{=} f'_y$;(ii) $iround_{mode}(s_x, e_x, f_x) = iround_{mode}(s_y, e_y, f_y)$; and (iii) $nround_{mode}(s_x, e_x, f_x) = nround_{mode}(s_y, e_y, f_y)$.

Proof: The assumption that $x \stackrel{p-e_x}{=} y$ means that either $x = y$ or $x, y \in I$, where $I =]q \cdot 2^{e'_x - p}, (q+1) \cdot 2^{e'_x - p}[$, for some integer q. If $x = y$ then the claim follows from the uniqueness of the (bounded) normalized factoring representations (s'_x, e'_x, f'_x) and (s'_y, e'_y, f'_y). For the second case, since the interval I cannot contain both negative and positive numbers, let us assume that $x > 0$, and hence $y > 0$ as well.

Note also, that the interval I either consists only of denormalized values or normalized values. The reason is that $2^{e_{min}}$ can not belong to the interval I. Since both factorings (s'_x, e'_x, f'_x) and (s'_y, e'_y, f'_y) are bounded normalized, it follows that either $f'_x, f'_y \in [0, 1[$ or $f'_x, f'_y \in [1, 2[$. If $f'_x, f'_y \in [0, 1[$, then it follows that $e'_x = e'_y = e_{min}$. Therefore, $f'_x, f'_y \in]q \cdot 2^{-p}, (q + 1) \cdot 2^{-p}[$, and $f'_x \stackrel{p}{=} f'_y$, as required in part (i).

If $f'_x, f'_y \in [1, 2[$, let us assume by contradiction that $e'_x > e'_y$. This would imply that

$$f'_y \cdot 2^{e'_y} < 2^{1+e'_y} \leq 2^{e'_x} \leq f'_x \cdot 2^{e'_x}.$$

But the interval I cannot contain $2^{e'_x}$, so that we have a contradiction to our assumption. Therefore, $e_x = e_y$, and as before, this implies $f'_x \stackrel{p}{=} f'_y$, as required. Part (ii) follows from

the computation of the rounding function $iround(s,e,f)$ according to lemma 2.7, the definition of aligned significand rounding in equation 2.9 and equation 2.49 with $\gamma = p$.

We use from definition 2.8, that the rounded values are the same for both rounding functions $iround_{mode}(s,e,f)$ and $nround_{mode}(s,e,f)$, so that from part (ii) it follows, that $val(nround_{mode}(s_x,e_x,f_x)) = val(nround_{mode}(s_y,e_y,f_y))$. Part (iii) then follows from the uniqueness of NF factoring representations. □

Similarly, for the computation of $\breve{r}_{mode}(s,e,f)$, we have:

Lemma 2.14 *Let (s_x,e_x,f_x) and (s_y,e_y,f_y) be two factorings and let (s'_x,e'_x,f'_x) and (s'_y,e'_y,f'_y) be the corresponding unbounded normalized factorings: $(s'_x,e'_x,f'_x) = \eta(s_x,e_x,f_x)$ and $(s'_y,e'_y,f'_y) = \eta(s_y,e_y,f_y)$. If the values of (s'_x,e'_x,f'_x) and (s'_y,e'_y,f'_y) are $(p-e'_x)$-equivalent:*

$$val(s_x,e_x,f_x) \overset{p-e'_x}{=} val(s_y,e_y,f_y),$$

then (i) $s'_x = s'_y$, $e'_x = e'_y$, and $f'_x \overset{p}{=} f'_y$; and (ii) $\breve{r}_{mode}(s_x,e_x,f_x) = \breve{r}_{mode}(s_y,e_y,f_y)$.

Proof: The proof is a simplified version of the proof of the previous Lemma, because all factorings are normalized in this case and no distinction between normalized and denormalized factorings is necessary. □

Based on Lemma 2.13 and Lemma 2.14 a rounding circuitry only has to know the p-representative of the significand of the unbounded or bounded normalized factoring and not its precise value to be able to round the factoring correctly.

Usually, no bounded or unbounded normalized factoring, but only an arbitrary factoring is considered as input of the rounding computations. Then, the knowledge of the p-representative of the significand does *not* directly ensure the possibility of correct IEEE rounding like in the cases of Lemma 2.13 and Lemma 2.14. But if a simple additional condition on this p-representative of the significand is fulfilled, it is possible to find the correctly rounded result nevertheless:

Lemma 2.15 *Let (s,e,f) be a an arbitrary factoring and e' the exponent of the corresponding normalized factoring. If a positive integer $p' \geq p$ exists, so that $fr = rep_{p'}(f)$ and the following condition is fulfilled:*

$$fr \;\geq\; 1 \quad OR \quad fr = f,$$

then $(s,e,f) \overset{p-e'}{=} (s,e,fr)$, $iround_{mode}(s,e,f) = iround_{mode}(s,e,fr)$ and $nround_{mode}(s,e,f) = nround_{mode}(s,e,fr)$.

Proof: We separate the conditions (i) $fr = f$ and (ii) $fr \geq 1$:
(i) If $fr = f$, it is obvious that $round_{mode}(s,e,f) = round_{mode}(s,e,fr)$. (ii) By equation 2.48 from $fr \overset{p'}{=} f$, it follows that $val(s,e,fr) \overset{p'-e}{=} val(s,e,f)$. Let (s',e',f') and (s'',e'',fr') be the bounded normalized factoring corresponding to (s,e,f) and (s,e,fr): $(s',e',f') = \lfloor \eta_{e_{min}} \rfloor (s,e,f)$ and $(s'',e'',fr') = \lfloor \eta_{e_{min}} \rfloor (s,e,fr)$. From $fr \geq 1$ it follows that $f \geq 1$, so that with $f' \leq 2$, and $val(s,e,f) = val(s',e',f')$, we have $e' \geq e$. Using $p - e' \leq p' - e$ and lemma 2.12 we get

$$val(s'',e'',fr') = val(s,e,fr) \overset{p-e'}{=} val(s,e,f) = val(s',e',f').$$

The use of lemma 2.13(ii)-(iii) on this equation completes the proof. □

L	R	STICKY	RZ	RNE	RI
d.c.	0	0	ftr	ftr	ftr
d.c.	0	1	ftr	ftr	ftri
0	1	0	ftr	ftr	ftri
1	1	0	ftr	ftri	ftri
d.c.	1	1	ftr	ftri	ftri

Table 2.11: Significand rounding on representatives.

Lemma 2.16 *We consider an integer p, positive values x, x_h and the value x_l with $x = x_h + x_l$, $x_h = k \cdot 2^{-p}$ for an integer k and $|x_l| < 2^{-p}$, and a non-zero positive value q with $q \cdot |x_l| < 2^{-p}$. The value $x' = x_h + q \cdot x_l$ then is p-equivalent to x, so that*

$$rep_p(x) = rep_p(x').$$

Proof: We separate the proof in three cases: (a) ($x_l = 0$); (b) ($x_l > 0$); and (c) ($x_l > 0$). The proof of case (a) follows directly from $x = x_h = x'$. In case (b), from $0 < x_l < 2^{-p}$ it follows, that $x_h < x < x_h + 2^{-p}$, and $rep_p(x) = x_h + 2^{-p-1}$. In the same way from $0 < q \cdot x_l < 2^{-p}$, it follows, that $x_h < x' < x_h + 2^{-p}$ and, thus, $rep_p(x') = x_h + 2^{-p-1} = rep_p(x)$. In case (c), from $-2^{-p} < x_l < 0$ it follows, that $x_h - 2^{-p} < x < x_h$, and $rep_p(x) = x_h - 2^{-p-1}$. In the same way from $-2^{-p} < q \cdot x_l < 0$, it follows, that $x_h - 2^{-p} < x' < x_h$. Thus, $rep_p(x') = x_h - 2^{-p-1} = rep_p(x)$ and the proof of the lemma is completed. □

Finally, we describe some details of significand rounding on representatives.

Definition 2.14 *For the rounding at position $\lambda-1$ of a positive significand $f < 2$ with the λ-representative $frep =<\text{FREP}[0{:}\,\lambda{+}1]>_{neg} = rep_\lambda(f)$, we define the* truncated significand *$ftr =<\text{FREP}[0{:}\,\lambda{-}1]>_{neg}$ and the* incremented significand *$ftri = ftr + 2^{-\lambda+1}$.*

Because $ftr \le f < ftr + 2^{-\lambda+1} = ftri$, the values ftr and $ftri$ are the two possible results of $rnd_{(mode \star s),\lambda-1}(frep)$. Which of them is chosen, depends only on the rounding mode $(mode \star s) \in \{RZ, RNE, RI\}$ that is encoded by SR_MODE[1:0] according to table2.3 and the three least significant bits of the representative, the *L-bit* L = FREP[λ−1], the *round-bit* R = FREP[λ], and the *sticky bit* STICKY = FREP[λ+1]. Table 2.11 lists all different rounding cases according to the rounding definitions for positive arguments from equation 2.4-2.2. In this table an entry 'd.c.' (don't care) means, that the value of this bit does not effect the result. From this table one can easily derive the equation for the condition that the incremented significand $ftri$ has to be chosen as the rounded significand. This condition is called the condition for the *rounding increment*:

$$\begin{aligned} \text{RINC} &= is\_RI(mode) \wedge (\text{R} \vee \text{STICKY}) \vee is\_RNE(mode) \wedge \text{R} \wedge (\text{L} \vee \text{STICKY}) && (2.53) \\ &= \text{SR\_MODE}[1] \wedge (\text{R} \vee \text{STICKY}) \vee \text{SR\_MODE}[0] \wedge \text{R} \wedge (\text{L} \vee \text{STICKY}) && (2.54) \end{aligned}$$

so that with $\lambda = p$ significand rounding can be written by:

$$sig\_rnd_{mode \star s}(s, e, f) = \begin{cases} (s, e, ftri) & \text{if RINC} \\ (s, e, ftr) & \text{otherwise.} \end{cases} \quad (2.55)$$

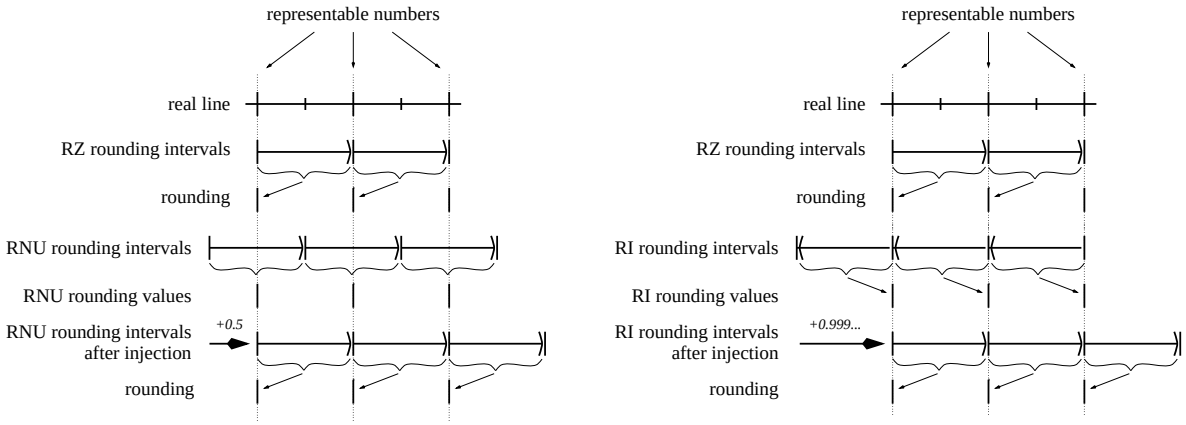

Figure 2.3: Injection mapping

Lemma 2.17 *For a factoring (s, e, f) with a significand $f < 2$, the p-representative $frep =< \text{FREP}[0 : p+1] >_{neg} = rep_p(f)$, the rounding mode $(mode \star s) \in \{RZ, RNE, RI\}$ and the rounded factoring $(s, e, frnd) = sig\_rnd_{mode \star s}(s, e, frep)$, the case that signifcand rounding changes the value of the significand can be recognized by:*

$$(\text{FREP}[p] \ OR \ \text{FREP}[p+1]) \iff (frnd \neq f)$$

We call this condition the significand rounding inexactness.

Proof: The lemma follows from equation 2.51-2.52 with $\gamma = p$ and the use of the property that $frep$ is an integral multiple of 2^{-p+1}, iff ($\text{FREP}[p]$ OR $\text{FREP}[p+1]) = 0$. □

2.5.2 Injection Based Rounding

Rounding by injection reduces the rounding modes RI and RNU to RZ [9, 40, 11, 12]. This reduction is possible for the rounding of operands x, that are integral multiples of 2^{-k}, with an integer k, that is larger than the rounding position λ. The rounding mode reduction is based on adding an injection:

$$inj = \begin{cases} 0 & \text{if RZ} \\ 2^{-\lambda-1} & \text{if RNU} \\ 2^{-\lambda} - 2^{-k} & \text{if RI,} \end{cases}$$

that depends only on the rounding mode.

Lemma 2.18 *With $mode \in \{RZ, RNU, RI\}$, the effect of adding inj can be described by*

$$rnd_{mode,\lambda}(x) = rnd_{RZ,\lambda}(x + inj).$$

Proof: Figure 2.3 depicts this reduction of RNU and RI to RZ. □

2.5.3 Gradual Rounding

In this section we deal with the situation, that rounding of a positive value x is not computed at the proper position λ_2 in a single step like in

$$sires = rnd_{mode,\lambda_2}(x),$$

but that the rounding result has to be computed in multiple steps, where a result of one rounding step is the input of the next rounding step with a smaller rounding precision $0 < \lambda_2 \leq \lambda_1$ like in

$$mures = rnd_{mode,\lambda_2}(rnd_{mode,\lambda_1}(x)).$$

In [21], the principles and problems of such *gradual* rounding are described. If only the rounded result $rores_1 = rnd_{mode,\lambda_1}(x)$ of a rounding step is used in the succeeding rounding decision, information gets lost and the multi-step rounding result $mures$ could differ from the correct single-step rounding result $sires$ (like in figure 2.4). In [21] this situation is called a *step error* and it is proven that such a step error can only occur in rounding mode RNE. To prevent step errors, two tag bits are required for the rounding decision in addition to the rounded result of the previous step:

- TINX is active if the rounded result of the previous step was inexact:

  $$(rnd_{mode,\lambda_1}(x) \neq x).$$

  Corresponding to the inexactness recognition in significand rounding, TINX can be computed from the round-bit R_1 and the sticky-bit STICKY_1 of the previous rounding step:

  $$\mathrm{TINX} \quad = \quad \mathrm{R}_1 \text{ OR } \mathrm{STICKY}_1. \tag{2.56}$$

- TINC is active if the previous rounding decision was a rounding increment (RINC=1):

  $$(bin_{-\lambda_1}^{-\lambda_1}(rnd_{mode,\lambda_1}(x)) \quad \neq \quad bin_{-\lambda_1}^{-\lambda_1}(x)). \tag{2.57}$$

Like in the conventional rounding, the rounded result of the previous rounding step $rores_1$ lies between two rounding possibilities $ftr = t \cdot 2^{-\lambda_2} \leq rores_1 < (t+1) \cdot 2^{-\lambda_2} = ftri$, so that the gradual rounding of $rores_1$ at position λ_2 corresponds to the selection

$$rores_2 \;=\; sires \;=\; \begin{cases} ftri & \text{if GRINC} \\ ftr & \text{otherwise.} \end{cases} \tag{2.58}$$

Using the two tag bits TINX and TINC in the *gradual* rounding decision GRINC enables to simulate single-step rounding by multi-step rounding in all rounding modes $(mode \star s) \in \{RZ, RNE, RI\}$ (encoded by $\mathrm{SR\_MODE}[1:0]$). As a solution [21] suggests to use the following equations to compute GRINC and the two tag bits TINX_2 and TINC_2 of the actual rounding step, where $(\mathrm{L}_2, \mathrm{R}_2, \mathrm{STICKY}_2) = \mathrm{REP}_{\lambda_2+1}(rores_1)[\lambda_2 : \lambda_2+2]$ and TINX_1 and TINC_1 are the corresponding tag bits of the previous rounding step:

$$\begin{aligned} \mathrm{GRINC} \;=\; & (\mathrm{SR\_MODE}[1] \wedge (\mathrm{R}_2 \vee \mathrm{STICKY}_2)) \vee \\ & ((\mathrm{SR\_MODE}[0] \wedge \mathrm{R}_2) \wedge (\mathrm{STICKY}_2 \vee \overline{\mathrm{TINC}_1} \wedge (\mathrm{L}_2 \vee \mathrm{TINX}_1))) \end{aligned} \tag{2.59}$$

$$\begin{aligned} \;=\; & (\mathrm{SR\_MODE}[1] \;\overline{\wedge}\; (\mathrm{R}_2 \vee \mathrm{STICKY}_2)) \;\overline{\wedge}\; \\ & ((\mathrm{SR\_MODE}[0] \wedge \mathrm{R}_2) \;\overline{\wedge}\; (\overline{\mathrm{STICKY}_2} \;\overline{\wedge}\; (\overline{\mathrm{TINC}_1} \;\overline{\wedge}\; (\mathrm{L}_2 \vee \mathrm{TINX}_1)))) \end{aligned} \tag{2.60}$$

$$\mathrm{TINX}_2 \;=\; \mathrm{R}_2 \vee \mathrm{STICKY}_2 \vee \mathrm{TINX}_1 \tag{2.61}$$

$$\mathrm{TINC}_2 \;=\; \mathrm{GRINC} \vee (\overline{\mathrm{STICKY}_2} \wedge \overline{\mathrm{R}_2} \wedge \mathrm{TINC}_1). \tag{2.62}$$

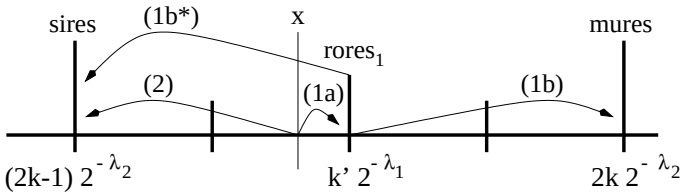

Figure 2.4: Gradual rounding: In rounding mode RNE the value x should be rounded at position λ_2 to the value $sires = (2k-1)\cdot 2^{-\lambda_2}$ like depicted by arrow (2). If we round the value x in a first step at position λ_1, we get the intermediate rounded result $rores_1 = k'\cdot 2^{-\lambda_1}$ like depicted by arrow (1a). The result of a second conventional rounding step of $rores_1$ at position λ_2 (arrow (1b)) is $mures = 2k\cdot 2^{-\lambda_2}$ and differs from the single step rounding result $sires$. A gradual rounding step on $rores_1$ at position λ_2 that uses additional information from the previous rounding step yields the rounded result $sires$ like depicted by arrow (1b*).

Based on the equations for the gradual rounding, we define gradual rounding functions corresponding to the definition of the rounding functions $rnd_{mode,\lambda}$:

Definition 2.15 *For $mode \in \{RZ, RNE, RI\}$ encoded by* SR_MODE$[1:0]$*, the gradual rounding function $grnd_{mode,\lambda} : I\!R \times \{0.1\}^2 \longrightarrow I\!R \times \{0.1\}^2$ is defined by*

$$grnd_{mode,\lambda}(f, \mathrm{TINC}_1, \mathrm{TINX}_1) = \begin{cases} (ftri, \mathrm{TINC}_2, \mathrm{TINX}_2) & \textit{if } \mathrm{GRINC} \\ (ftr, \mathrm{TINC}_2, \mathrm{TINX}_2) & \textit{otherwise.} \end{cases}$$

with the truncated significand ftr and the incremented significand $ftri$ from definition 2.14, and the computation of GRINC, TINC_2 *and* TINX_2 *according to equations 2.60-2.62.*

Lemma 2.19 *For any integers $\lambda_2 \leq \lambda_1$ the rounding function rnd_{mode,λ_2} can be decomposed into two gradual rounding steps, so that for*

$$(frnd, \mathrm{X}[1:0]) = grnd_{mode,\lambda_2}\left(grnd_{mode,\lambda_1}(f, 00)\right),$$

we get $frnd = rnd_{mode,\lambda_2}(f)$.

Proof: This lemma just summarizes the previous descriptions of the properties of gradual rounding using our definition 2.15 of the gradual rounding function $grnd_{mode\star s,\lambda}$. □

Definition 2.16 *For the significand rounding we define two gradual rounding steps by the significand rounding functions $sgrnd1 : FACT(I\!R) \longrightarrow FACT(I\!R) \times \{0,1\}^2$ and $sgrnd2 : FACT(I\!R) \times \{0,1\}^2 \longrightarrow FACT(I\!R)$. With $(frnd_1, \mathrm{TINC}_1, \mathrm{TINX}_1) = grnd_{mode\star s,52}(f, 00)$,*

$$sgrnd1_{mode\star s}(s, e, f) = ((s, e, frnd_1), \mathrm{TINC}_1, \mathrm{TINX}_1)$$

and with $(frnd_3, \mathrm{TINC}_3, \mathrm{TINX}_3) = grnd_{mode\star s,52}(frnd_2, \mathrm{TINC}_2, \mathrm{TINX}_2)$,

$$sgrnd2_{mode\star s}\left((s, e, frnd_2), \mathrm{TINC}_2, \mathrm{TINX}_2\right) = (s, e, frnd_3)$$

Additionally, we extend the definitions of the bounded normalization shift, the post-normalization shift and the function val on outputs of the first gradual rounding step:

$$\begin{aligned} \lceil \eta_\beta \rceil \left((s, e, f), \mathrm{X}[1:0]\right) &= \left(\lceil \eta_\beta \rceil (s, e, f), \mathrm{X}[1:0]\right) \\ post\_norm\left((s, e, f), \mathrm{X}[1:0]\right) &= \left(post\_norm(s, e, f), \mathrm{X}[1:0]\right) \\ val\left((s, e, f), \mathrm{X}[1:0]\right) &= val(s, e, f) \end{aligned}$$

Lemma 2.20 *The rounding function iround can be docomposed into a normalization shift, a first significand gradual rounding step by sig_grnd1, a bounded normalization shift, a second significand gradual rounding step by sig_grnd1, a post-normalization shift and exponent rounding. Thus, with the definition of the normalized gradual result factoring*

$$((s_{GF}, e_{GF}, f_{GF}), \text{TINC}, \text{TINX}) \quad = \quad post\_norm(sgrnd1_{mode\star s}(\eta(s,e,f))), \tag{2.63}$$

the IEEE factoring of the rounded result can be computed by

$$(s_{res}, e_{res}, f_{res}) = \\ exp\_rnd_{mode\star s}(post\_norm(sgrnd2_{mode\star s}(\lceil \eta_{e_{min}} \rceil((s_{GF}, e_{GF}, f_{GF}), \text{TINC}, \text{TINX})))),$$

so that $iround_{mode}(s,e,f) = (s_{res}, e_{res}, f_{res})$.

Proof: We first introduce some notation: We denote the input factoring of the first gradual rounding step by $(s_1, e_1, f_1) = \eta(s,e,f)$ and the output by $((s_2, e_2, f_2)\text{TINC}, \text{TINX}) = sgrnd1_{mode\star s}(s_1, e_1, f_1)$. The input factoring of the second gradual rounding step is denoted by $((s_3, e_3, f_3)\text{TINC}, \text{TINX}) = \lceil \eta_{e_{min}} \rceil((s_{GF}, e_{GF}, f_{GF}), \text{TINC}, \text{TINX})$ and the output is denoted by $(s_4, e_4, f_4) = sgrnd2_{mode\star s}\,((s_3, e_3, f_3)\text{TINC}, \text{TINX})$.

The lemma will be proven in two steps. In the first step (a) we will show that the value of $val(s_{res}, e_{res}, f_{res})$ corresponds to the value of the IEEE rounded result $val(iround(s,e,f))$. In the second step (b) we will show, that also $(s_{res}, e_{res}, f_{res})$ is an IEEE factoring like $iround(s,e,f)$.

In part (a) of the proof, we have only to consider the values during the computation. The value of the input factoring might only be changed by the two gradual rounding steps and by the exponent rounding, so that $x_1 = val(s,e,f) = val(s_1, e_1, f_1)$ and $x_2 = val(s_2, e_2, f_2) = val(s_3, e_3, f_3)$. Because the exponent rounding function is the same like in the computation of $iround(s,e,f)$ according to lemma 2.7, we only have to compare the significand rounding, namely we only have to show that for $e'' = max\{e_{min}, e_1\}$

$$val(s_4, e_4, f_4) = rnd_{mode,-e''+p-1}(x).$$

Like for the rounding function rnd, we can also use for the gradual rounding function $grnd$, that for integers x: $2^x \cdot val(grnd_{mode,\lambda}(f, \text{X}[1\!:\!0])) = val(grnd_{mode,\lambda-x}(2^x \cdot f, \text{X}[1\!:\!0]))$. In this way the rounded values of the gradual rounding steps x_2 and $x_4 = val(s_4, e_4, f_4)$ can be written by:

$$\begin{aligned}
x_4 &= (-1)^{s_4} \cdot 2^{e_4} \cdot f_4 && (2.64)\\
&= (-1)^{s_3} \cdot 2^{e_3} \cdot val(grnd_{mode\star s,p-1}(f_3, \text{TINC}, \text{TINX})) && (2.65)\\
&= (-1)^{s_3} \cdot val(grnd_{mode\star s,-e_3+p-1}(2^{e_3} \cdot f_3, \text{TINC}, \text{TINX})) && (2.66)\\
x_3 &= (-1)^{s_3} \cdot 2^{e_3} \cdot f_3 && (2.67)\\
&= (-1)^{s_2} \cdot 2^{e_2} \cdot f_2 && (2.68)\\
&= (-1)^{s_1} \cdot 2^{e_1} \cdot val(grnd_{mode\star s,52}(f_1, 00)) && (2.69)\\
&= (-1)^{s_1} \cdot val(grnd_{mode\star s,-e_1+52}(2^{e_1} \cdot f_1, 00)) && (2.70)\\
&= (-1)^{s} \cdot val(grnd_{mode\star s,-e_1+52}(abs(x), 00)) && (2.71)
\end{aligned}$$

In the computation steps between the two gradual rounding steps, the exponent could be changed by the post-normalization shift, so that $e_{GF} \in \{e_1, e_1+1\}$ and by the bounded normalization shift, so that $e_3 = max\{e_{min}, e_{GF}\}$ (Note, that f_{GF} is normalized, so that

e_{GF} is already the normalized exponent.) Because both operations only could increase the exponent and because we consider $p \in \{24, 53\}$, we get $-e_3 + p - 1 \leq -e_1 + 52$. For this reason we can use lemma 2.19 on the two gradual rounding steps with $\lambda_1 = -e_1 + 52$ and $\lambda_2 = -e_3 + p - 1$. In this way equation 2.67, 2.68 and 2.71 combine to

$$\begin{aligned} x_4 &= (-1)^s \cdot grnd_{mode\star s,-e_3+p-1}(grnd_{mode\star s,-e_1+52}(abs(x), 00)) && (2.72)\\ &= (-1)^s \cdot rnd_{mode\star s,-e_3+p-1}(abs(x)) && (2.73)\\ &= rnd_{mode,-e_3+p-1}(x). && (2.74) \end{aligned}$$

There are only two cases possible, namely: (i) $e_{GF} = e_1$, and (ii) $e_{GF} = e_1 + 1$:

(i) From $e_{GF} = e_1$, it follows, that $e_3 = max\{e_{min}, e_1\} = e''$, so that we get $x_4 = rnd_{mode,-e''+p-1}(x)$, as required.

(ii) For $e_{GF} = e_1 + 1$, it follows from the definition of the post-normalization shift, that the significand is changed to $f_{GF} = 1$. The rounding does not change the operand $((-1)^s \cdot 2^{e_1+1})$ regardless of whether rounding position $-(e_1+1)+p-1$ or rounding position $-e_1 + p - 1$ is considered. Thus, we get $x_4 = rnd_{mode,-e''+p-1}(x)$ also in this case and part (a) of the proof is completed.

Part (b) can be proven like part (b) of lemma 2.7 starting with the input factoring of the bounded normalization shift (s_{GF}, e_{GF}, f_{GF}), because the computations in the four steps that are computed on (s_{GF}, e_{GF}, f_{GF}) in this lemma correspond to the four steps in the computation of $iround(s, e, f)$ according to lemma 2.7. □

Definition 2.17 *For mode $\in \{RZ, RNE, RI, RMI\}$, we define the two gradual rounding functions $ground1_{mode}$ and $ground2_{mode}$ by*

$$\begin{aligned} &ground1_{mode}(s, e, f) = post\_norm(sgrnd1_{mode\star s}(\eta(s, e, f)))\\ &ground2_{mode}((s_{GF}, e_{GF}, f_{GF}), \text{TINC}, \text{TINX}) =\\ &\quad exp\_rnd_{mode\star s}(post\_norm(sgrnd2_{mode\star s}(\lceil \eta_{e_{min}} \rceil((s_{GF}, e_{GF}, f_{GF}), \text{TINC}, \text{TINX})))). \end{aligned}$$

Corollary 2.21 *With the definition 2.17, lemma 2.20 can be written by:*

$$iround_{mode}(s, e, f) = ground2_{mode}(ground1_{mode}(s, e, f)).$$

Lemma 2.22 *The equation $((s_{GF}, e_{GF}, f_{GF}), \text{TINC}, \text{TINX}) = ground1_{mode}(s, e, f)$ is invariant on the addition of $k \in \mathbb{R}$ to the exponent, namely*

$$((s_{GF}, e_{GF} + k, f_{GF}), \text{TINC}, \text{TINX}) = ground1_{mode}(s, e + k, f),$$

so that it does not matter if k is added to the exponent of the input or the output factoring.

Proof: The computations in each of the three steps according to definition 2.17 of function $ground1$, namely, the unbounded normalization shift, the gradual rounding and the post-normalization shift only depend on the values of the significands and the signs of the factorings and not on the exponent values and so does the sequence of these three steps in function $ground1$. □

Corollary 2.23 *With $((s_{GF}, e_{GF}, f_{GF}), \text{TINC}, \text{TINX})$=$ground1_{mode}(s_{ex}, e_{ex}, f_{ex})$ and corollary 2.21, the IEEE factoring of the final result according to equation 2.18 is given by*

$$iround(s_{ex}, e_{ex} + wec, f_{ex}) = ground2_{mode}((s_{GF}, e_{GF} + wec, f_{GF}), \text{TINC}, \text{TINX}).$$

2.6 Internal Representations

In this section, based on factoring representations, we define floating-point number representations at the bit level. In each presented format the number representations are integrated for the cases of single precision and double precision. The first three formats, namely the *packed format*, the *unpacked format* and the *normalized format* contain the representation of single precision and double precision IEEE values. The last two formats, the *representative format* and the *gradual result format*, represent results of IEEE operations that have not been fully rounded yet, In these cases some further computation steps are required to achieve the corresponding single precision or double precision IEEE FP value and there could be two or more representations in these formats that lead to the same IEEE FP value after rounding.

2.6.1 Packed Format

The number representations in the *packed format* (PF) are based on the packed representations for single precision and double precision defined by the IEEE standard. These packed representations encode the IEEE factoring of a number in a binary form. In the packed format, the IEEE packed representations for single and double precision (see figure 2.2) are integrated into a 64 bit wide representation, where the smaller single precision representations are left aligned and padded with 32 zeros (figure 2.5). We index a bus with this format by $BUS_{PF}[63:0]$. For single precision usage we have:

$$\begin{aligned} \mathrm{S}_{PF} &= BUS_{PF}[63] && (2.75)\\ \mathrm{E}_{PF}[7:0] &= BUS_{PF}[62:55] && (2.76)\\ \mathrm{F}_{PF}[1:23] &= BUS_{PF}[54:32] && (2.77) \end{aligned}$$

and for double precision usage we have:

$$\begin{aligned} \mathrm{S}_{PF} &= BUS_{PF}[63] && (2.78)\\ \mathrm{E}_{PF}[10:0] &= BUS_{PF}[62:52] && (2.79)\\ \mathrm{F}_{PF}[1:52] &= BUS_{PF}[51:0]. && (2.80) \end{aligned}$$

Definition 2.18 *For single and double precision with $(n,p) \in \{(8,24),(11,53)\}$ we define the function $\mathrm{PF} : IEEEfact_{n,p} \longrightarrow \{0,1\}^{64}$, that computes the representation of an IEEE factoring $(s,e,f) \in IEEEfact_{n,p}$ in the packed format. With $e = <\mathrm{E}_{PF}[n:0]>_{bias_n}$ and $f = <\mathrm{F}_{PF}[0:p-1]>_{neg}$ for representable numbers and quiet NaNs, the function PF is defined by*

$$\mathrm{PF}(s,e,f) = \begin{cases} (\mathrm{S}, 1^{n-1}, 0^{64-n}) & \text{if } f = f_\infty \\ (\mathrm{S}, 1^{n-1}, 1, 0^{63-n}) & \text{if } f = f_{sNaN} \\ (\mathrm{S}, 1^{n-1}, 0, \mathrm{F}_{PF}[2:p-1], 0^{64-n-p}) & \text{if } f = f_{qNaN} \\ \big(\mathrm{S}, (\mathrm{E}_{PF}[n-1:0] \wedge \mathrm{F}_{PF}[0]), \mathrm{F}_{PF}[1:p-1], 0^{64-n-p}\big) & \text{otherwise.} \end{cases}$$

In the opposite direction the function $fact_{PF} : \{0,1\}^{64} \longrightarrow IEEEfact_{n,p}$ computes the IEEE factoring that is represented by $BUS_{PF}[63:0]$ in the packed format. With

$$(\mathrm{S}, \mathrm{E}_{PF}[n-1:0], \mathrm{F}_{PF}[1:p-1]) = BUS_{PF}[63:63-n-p-1],$$

the denormalized factoring $(s, e_{den}, f_{den}) = (s, e_{min}, <(0, \mathrm{F}_{PF}[1:p-1])>_{neg})$ and the normalized factoring $(s, e_{nor}, f_{nor}) = (s, <\mathrm{E}_{PF}[n-1:0]>_{bias_n}, <(1, \mathrm{F}_{PF}[1:p-1])>_{neg})$, the

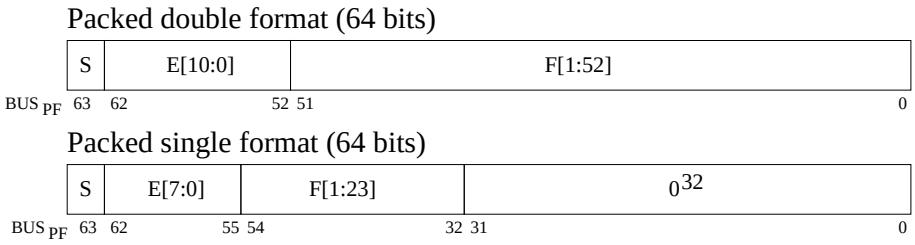

Figure 2.5: Packed format for single and double precision.

function $fact_{PF}$ is defined according to the definitions of the IEEE packed representations from section 2.2.3 by

$$fact_{PF}(BUS_{PF}[63:0]) = \begin{cases} (s, e_{den}, f_{den}) & \textit{if } \text{E}_{PF}[n-1:0] = 0^n \\ (s, e_{nor}, f_{nor}) & \textit{if } \text{E}_{PF}[n-1:0] \neq 1^n \wedge \text{F}_{PF}[1:p-1] \neq 0^{p-1} \\ (s, e_{\infty}, f_{\infty}) & \textit{if } \text{E}_{PF}[n-1:0] = 1^n \wedge \text{F}_{PF}[1:p-1] = 0^{p-1} \\ (s, e_{sNaN}, f_{sNaN}) & \textit{if } \text{E}_{PF}[n-1:0] = 1^n \wedge \text{F}_{PF}[1] = 1 \\ (s, e_{qNaN}, f_{qNaN}) & \textit{otherwise} \end{cases}$$

2.6.2 Unpacked Format

The unpacked format is also a binary encoding for IEEE factorings. But in this case the number representations are unpacked, i.e., information about an IEEE factoring is not provided with the minimum amount of bits, but additional bits are included in the representation to have better access to certain informations about the number:

The hidden bit $\text{F}_{UF}[0]$ is included in the unpacked number representation. For representable numbers this bit is well defined by the exponent representation from the packed format. For special values like $+/-\infty$ and $NaNs$, we define the hidden bit to have the value $\text{F}_{UF}[0] = 1$, so that ∞ and NaN representations always include a normalized significand. Moreover, special values and zeros are indicated by 4 additional bits: ZERO, INF, QNAN and SNAN. At most one of these bits can be active in a number representation.

In the unpacked format the exponent is represented in the two's-complement representation for representable numbers. We define the exponent of special values to have the two's complement representation of $e_{UF} = e_{max} + 1$ in analogy to the packed format. To be able to include the two's complement representation of $e_{max} + 1 = 2^{n-1}$ in the exponent, the exponent representation is extended by one bit to a width of $n+1$-bits. For representable numbers this bit extension is computed by a sign extension. Representations of zero include an arbitrary exponent. In this case the value of the number is indicated by the sign and the additional bit ZERO in a unique way.

For an integrated representation of single and double precision values three bit fields are separated according to sign, exponent and significand (see figure 2.6). For single precision usage the significand is padded with 29 zeros on the right and a single precision exponent needs 3 additional bits on the left, that are computed by sign extension. We index a bus with this format by $BUS_{UF}[69:0]$ and have:

$$\begin{aligned} \text{S}_{UF} &= BUS_{UF}[69] \\ \text{E}_{UF}[11:0] &= BUS_{UF}[68:57] \\ \text{F}_{UF}[0:52] &= BUS_{UF}[56:4] \end{aligned}$$

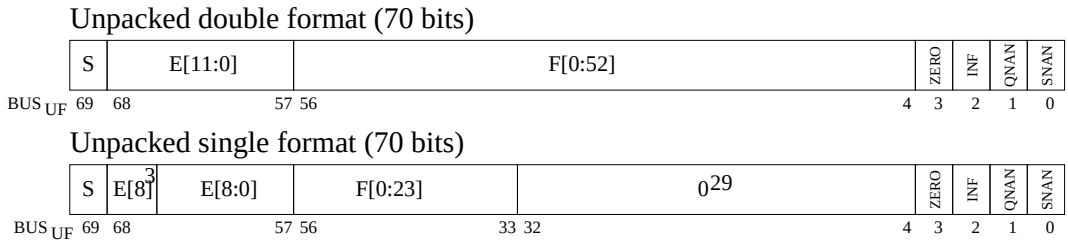

Figure 2.6: Unpacked format for single and double precision.

$$\begin{array}{lcllcl} \text{ZERO}_{UF} & = & BUS_{UF}[3] & \text{INF}_{UF} & = & BUS_{UF}[2] \\ \text{QNAN}_{UF} & = & BUS_{UF}[1] & \text{SNAN}_{UF} & = & BUS_{UF}[0]. \end{array}$$

Definition 2.19 *We define the function* UF : $IEEEfact_{n,p} \longrightarrow \{0,1\}^{70}$*, that computes the representation of an IEEE factoring* $(s,e,f) \in IEEEfact$ *in the unpacked format. With* $e = <\text{E}_{UF}[11{:}0]>_2$ *and* $f = <\text{F}_{UF}[0{:}52]>_{neg}$ *for representable numbers and quiet NaNs, the function* UF *is defined by*

$$\text{UF}(s,e,f) = \begin{cases} (\text{S}, & 0^{12}, & 0,0^{52}, & 1,0,0,0) & \text{if } f = 0 \\ (\text{S}, & 0,1,0^{10}, & 1,0^{52}, & 0,1,0,0) & \text{if } f = f_\infty \\ (\text{S}, & 0,1,0^{10}, & 1,0,\text{F}_{UF}[2{:}52], & 0,0,1,0) & \text{if } f = f_{qNaN} \\ (\text{S}, & 0,1,0^{10}, & 1,1,0^{51}, & 0,0,0,1) & \text{if } f = f_{sNaN} \\ (\text{S}, & \text{E}_{UF}[11{:}0], & \text{F}_{UF}[0{:}52], & 0,0,0,0) & \text{otherwise.} \end{cases}$$

In the opposite direction the function $fact_{UF} : \{0,1\}^{70} \longrightarrow IEEEfact_{n,p}$ *computes the IEEE factoring that is represented by* $BUS_{UF}[69:0]$ *in the unpacked format. With*

$$(\text{S}, \text{E}_{UF}[11:0], \text{F}_{UF}[0{:}52], \text{ZERO}, \text{INF}, \text{QNAN}, \text{SNAN}) = BUS_{UF}[69:0],$$

the function $fact_{UF}$ *is defined by*

$$fact_{UF}(BUS_{UF}[69{:}0]) = \begin{cases} (s, e_0, 0) & \text{if ZERO} \\ (s, e_\infty, f_\infty) & \text{if INF} \\ (s, e_{qNaN}, f_{qNaN}) & \text{if QNAN} \\ (s, e_{sNaN}, f_{sNaN}) & \text{if SNAN} \\ (s, <\text{E}_{UF}[11{:}0]>_2, <\text{F}_{UF}[0{:}52]>_{neg}) & \text{otherwise.} \end{cases}$$

2.6.2.1 Packed Format $\longrightarrow$ Unpacked Format

For the conversion of a number from a packed representation to the corresponding unpacked representation we summarize the conditions on the additional bits that have to be computed:

$$\begin{array}{lcll} \text{F}_{UF}[0] & = & 0 & , \text{ iff } \text{E}_{PF} = 0^n \\ \text{ZERO} & = & 1 & , \text{ iff } \text{E}_{PF} = 0^n \text{ AND } \text{F}_{PF}[1{:}p{-}1] = 0^{p-1} \\ \text{INF} & = & 1 & , \text{ iff } \text{E}_{PF} = 1^n \text{ AND } \text{F}_{PF}[1{:}p{-}1] = 0^{p-1} \\ \text{QNAN} & = & 1 & , \text{ iff } \text{E}_{PF} = 1^n \text{ AND } \text{F}_{PF}[1] = 0 \text{ AND } \text{F}_{PF}[2{:}p{-}1] \neq 0^{p-2} \\ \text{SNAN} & = & 1 & , \text{ iff } \text{E}_{PF} = 1^n \text{ AND } \text{F}_{PF}[1] = 1. \end{array} \tag{2.81}$$

Among the other bits, only the exponent representation changes by the subtraction of the corresponding bias and the non-redundant representation of e_{min}. The following equation

uses a sign extension for single precision and double precision exponents and lemma 2.1(ii) to convert the exponent from biased to two's complement representation for $\mathrm{E}_{PF} \neq 1^n$. Because for $\mathrm{E}_{PF} = 1^n$,

$$bin_0^n(< (\overline{\mathrm{E}_{PF}[n-1]}^2, \mathrm{E}_{PF}[n-2:0)] > +1) = < (0^2, 1^{n-2}) > +1 = 2^{n-1} = e_{max} + 1,$$

also the case $\mathrm{E}_{PF} = 1^n$ is included in the first and the third line.

$$\mathrm{E}_{UF}[11:0] = \begin{cases} bin_0^{11}(< (\overline{\mathrm{E}_{PF}[10]}^2, \mathrm{E}_{PF}[9:0)] > +1) & \text{if double AND } \mathrm{E}_{PF} \neq 0^{11} \\ (11, 0^8, 10) & \text{if double AND } \mathrm{E}_{PF} = 0^{11} \\ bin_0^{11}(< (\overline{\mathrm{E}_{PF}[7]}^5, \mathrm{E}_{PF}[6:0]) > +1) & \text{if single AND } \mathrm{E}_{PF} \neq 0^7 \\ (11111, 0^5, 10) & \text{if single AND } \mathrm{E}_{PF} = 0^7. \end{cases} \quad (2.82)$$

2.6.2.2 Unpacked Format $\longrightarrow$ Packed Format

Also in the conversion direction from unpacked number representations to packed number representations, the sign bit S_{PF} and the fraction $\mathrm{F}_{PF}[1:p-1]$ are copied identically. For the exponent conversion we have to distinguish between two cases of: (a) normalized numbers or special values; and (b) denormalized numbers or zero.

(a) In the case of normalized numbers or special values like $+/-\infty$ or $NaNs$, we have to convert the exponent from the two's complement representation to the biased representation. For normalized numbers, the bit $\mathrm{E}_{UF}[n]$ is only a sign extension: $\mathrm{E}_{UF}[n] = \mathrm{E}_{UF}[n-1]$, so that the conversion is computed with the help of lemma 2.2:

$$\mathrm{E}'_{PF}[n-1:0] = bin_0^{n-1}(< (\overline{\mathrm{E}_{UF}[n-1]}, \mathrm{E}_{UF}[n-2:0]) > -1).$$

For $+/-\infty$ or $NaNs$, we have $\mathrm{E}_{UF} = (01, 0^{n-1})$ and the above formula yields $\mathrm{E}'_{PF}[n-1:0] = 1^n$, as required for packed ∞- or NaN-representations. Thus, this formula can be used for normalized numbers, $+/-\infty$ or $NaNs$. Because

$$bin_0^7(< (\overline{\mathrm{E}_{UF}[7]}, \mathrm{E}_{UF}[6:0]) > -1) = bin_0^7(< (\overline{\mathrm{E}_{UF}[10]}, \mathrm{E}_{UF}[9:8], \overline{\mathrm{E}_{UF}[7]}, \mathrm{E}_{UF}[6:0]) > -1),$$

the formula can be integrated for single and double precision by

$$\mathrm{E}'_{PF}[n-1:0] = bin_0^{n-1}(< (\overline{\mathrm{E}_{UF}[10]}, \mathrm{E}_{UF}[9:8], \mathrm{E}_{UF}[7] \oplus \mathrm{DBL}, \mathrm{E}_{UF}[6:0]) > -1).$$

(b) In the case of a zero or a denormalized number, the exponent representation 0^n is required in the packed format.

Because $\mathrm{F}_{UF}[0] = 0$, iff the number is a zero or a denormalized number, we can distinguish between the two cases by the value of $\mathrm{F}_{UF}[0]$. Thus, the exponent conversion can be summarized by:

$$\mathrm{E}_{PF} = \begin{cases} bin_0^{n-1}(< (\overline{\mathrm{E}_{UF}[10]}, \mathrm{E}_{UF}[9:8], \mathrm{E}_{UF}[7] \oplus \mathrm{DBL}, \mathrm{E}_{UF}[6:0]) > -1) & \text{if } \mathrm{F}_{UF}[0] \\ 0^n & \text{otherwise.} \end{cases} \quad (2.83)$$

$$= bin_0^{n-1}(< (\overline{\mathrm{E}_{UF}[10]}, \mathrm{E}_{UF}[9:8], \mathrm{E}_{UF}[7] \oplus \mathrm{DBL}, \mathrm{E}_{UF}[6:0]) > -1) \wedge \mathrm{F}_{UF}[0] \quad (2.84)$$

2.6.3 Normalized Format

The only difference between the unpacked and the *normalized format* (NF) is, that in the case of the normalized format, the NF factoring of an IEEE value is encoded, and not the IEEE factoring like in the unpacked format. Therefore, the representation in the normalized format only differs from the unpacked format representation for denormalized numbers, which are also represented with a normalized significand in the normalized format. The only numbers which still contain a leading zero significand bit $\text{F}_{NF}[0]$ in the normalized format are $+/-0$.

Figure 2.7 indexes a bus with the normalized format by $BUS_{NF}[69:0]$ and separates bit fields similar to the unpacked representation:

$$\begin{aligned} \text{S}_{NF} &= BUS_{NF}[69] \\ \text{E}_{NF}[11:0] &= BUS_{NF}[68:57] \\ \text{F}_{NF}[0:52] &= BUS_{NF}[56:4] \end{aligned}$$

$$\begin{aligned} \text{ZERO}_{NF} &= BUS_{NF}[3] & \text{INF}_{NF} &= BUS_{NF}[2] \\ \text{QNAN}_{NF} &= BUS_{NF}[1] & \text{SNAN}_{NF} &= BUS_{NF}[0]. \end{aligned}$$

Definition 2.20 *We define the function* $\text{NF} : NFfact_{n,p} \longrightarrow \{0,1\}^{70}$*, that computes the representation of an NF factoring* $(s,e,f) \in NFfact_{n,p}$ *in the normalized format. With* $e = <\text{E}_{NF}[11{:}0]>_2$ *and* $f = <\text{F}_{NF}[0{:}52]>_{neg}$ *for representable numbers and quiet NaNs, the function* NF *is defined by*

$$\text{NF}(s,e,f) = \begin{cases} (\text{S}, & 0^{12}, & 0,0^{52}, & 1,0,0,0) & \text{if } f = 0 \\ (\text{S}, & 0,1,0^{10}, & 1,0^{52}, & 0,1,0,0) & \text{if } f = f_\infty \\ (\text{S}, & 0,1,0^{10}, & 1,0,\text{F}_{NF}[2{:}52], & 0,0,1,0) & \text{if } f = f_{qNaN} \\ (\text{S}, & 0,1,0^{10}, & 1,1,0^{51}, & 0,0,0,1) & \text{if } f = f_{sNaN} \\ (\text{S}, & \text{E}_{NF}[11{:}0], & \text{F}_{NF}[0{:}52], & 0,0,0,0) & \text{otherwise.} \end{cases}$$

In the opposite direction the function $fact_{NF} : \{0,1\}^{70} \longrightarrow NFfact_{n,p}$ *computes the NF factoring that is represented by* $BUS_{NF}[69:0]$ *in the normalized format. With*

$$(\text{S}, \text{E}_{NF}[11:0], \text{F}_{NF}[0{:}52], \text{ZERO}, \text{INF}, \text{QNAN}, \text{SNAN}) = BUS_{NF}[69:0],$$

the function $fact_{NF}$ *is defined by*

$$fact_{NF}(BUS_{NF}[69{:}0]) = \begin{cases} (s, e_0, 0) & \text{if ZERO} \\ (s, e_\infty, f_\infty) & \text{if INF} \\ (s, e_{qNaN}, f_{qNaN}) & \text{if QNAN} \\ (s, e_{sNaN}, f_{sNaN}) & \text{if SNAN} \\ (s, <\text{E}_{NF}[11{:}0]>_2, <\text{F}_{NF}[0{:}52]>_{neg}) & \text{otherwise.} \end{cases}$$

2.6.3.1 Unpacked Format $\longrightarrow$ Normalized Format

To convert numbers from the unpacked representation to the normalized representation, an unbounded normalization shift η for non-zero denormalized numbers according to definition 2.3 has to be computed. Because we can recognize non-zero denormalized numbers by the condition ($\overline{\text{F}[0]}$ AND $\overline{\text{ZERO}}$), the conversion could be described by

$$(\text{S}_{NF}, <\text{E}_{NF}>_2, <\text{F}_{NF}>_{neg}) = \begin{cases} \eta(\text{S}_{UF}, <\text{E}_{UF}>_2, <\text{F}_{UF}>_{neg}) & \text{if } (\overline{\text{F}[0]} \text{ AND } \overline{\text{ZERO}}) \\ (\text{S}_{UF}, <\text{E}_{UF}>_2, <\text{F}_{UF}>_{neg}) & \text{otherwise.} \end{cases} \quad (2.85)$$

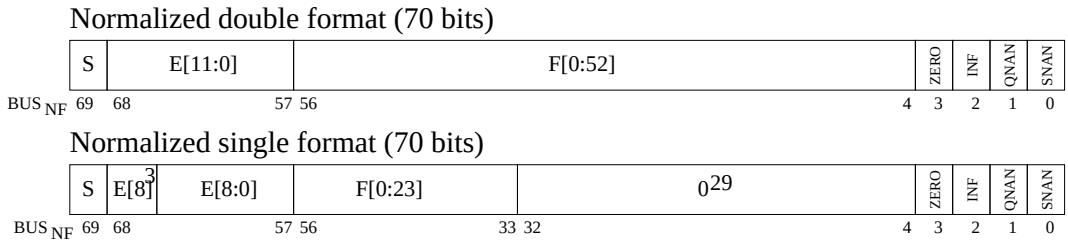

Figure 2.7: Normalized format for single and double precision.

To determine the shift amount for the unbounded normalization shift, the amount of *leading zeros* lz in the significand $\mathrm{F}_{UF}[0:52]$ has to be detected, following lemma 2.3(ii). For the normalization, the significand is left-shifted and the exponent is decremented by the amount lz. In this way, the normalization shift might decrease the exponent by a maximum of 53, because the widest significand contains 53 bits, that could all be zero. Therefore, by the exponent adjustment the range of the exponent is enlarged to $[1024 : -1075]$ for double precision and a 12-bit two's-complement exponent representation is sufficient in the normalized representation.

If $(\overline{\mathrm{F}[0]}\mathrm{AND}\overline{\mathrm{ZERO}}) = 0$, the number is either a zero and the significand is $\mathrm{F}_{UF}[0:52] = 0^{53}$, or the number is a normalized number, $+/-\infty$ or a NaN, so that $\mathrm{F}_{UF}[0] = 1$, and therefore, the shift amount is zero: $lz = 0$. In both cases the significand representation is not changed by an normalization shift by lz positions. Thus, the normalization shift can also be computed for $(\overline{\mathrm{F}[0]}$ AND $\overline{\mathrm{ZERO}}) = 0$, so that for all cases

$$\mathrm{F}_{NF}[0:52] = (\mathrm{F}_{UF}[lz:52], 0^{lz}). \tag{2.86}$$

For the exponent adjustment the shift amount lz is substracted. Because the exponent representation is not valid for zeros, this subtraction can also be computed for all cases.

$$< \mathrm{E}_{NF}[11:0] >_2 = < (\mathrm{E}_{UF}[10], \mathrm{E}_{UF}[10:0]) >_2 - lz. \tag{2.87}$$

Because $lz = 0$ for infinities and NaNs, in the normalized representation we get the exponent $e_{NF} = emax + 1 = e_{UF}$ for them like in the unpacked representation. Also the sign bit and additional bits stay the same in both representations.

As denormalized significands are shifted, these significands do not end with weight 2^{-p+1}, but the least significant bit of the significand is changed to the the significand position with weight $2^{-p+lz+1}$. Because significand rounding is done at this least significant bit position of the significand, the significand rounding position changes to the position with weight $2^{-p+lz+1}$ for denormalized numbers. in the normalized format. In combination with the changed exponent $e-lz$ this results in the rounding position $\lambda = e-lz-p+lz+1 = e-p+1$, which agrees with the previous IEEE rounding description and with the rounding procedure according to lemma 2.8.

2.6.3.2 Normalized Format $\longrightarrow$ Unpacked Format

In this conversion direction from normalized to unpacked number representations, the representations of denormalized IEEE values have to be changed. For these numbers, the factoring representations have to be denormalized, so that the exponent is adjusted to e_{min}. Because the exponent of denormals in the normalized representation is smaller than

e_{min}, the shift distance lz can be computed by

$$lz = \begin{cases} e_{min} - < \mathrm{E}_{NF}[11:0] >_2 & \text{if } (e_{min} - < \mathrm{E}_{NF}[11:0] >_2) \geq 0 \\ 0 & \text{otherwise.} \end{cases} \quad (2.88)$$

The conversion then changes the exponent and the significand representations by

$$\mathrm{F}_{UF}[0:52] = (0^{lz}, \mathrm{F}_{NF}[0:52-lz]), \quad (2.89)$$

$$< \mathrm{E}_{UF}[11:0] >_2 = \begin{cases} < \mathrm{E}_{NF}[11:0] >_2 & \text{if } \mathrm{F}_{UF}[0] \\ e_{min} & \text{otherwise.} \end{cases} \quad (2.90)$$

The sign bit and additional bits stay the same in both representations.

2.6.4 Representative Format

The *representative format* (RF) is a representation for results of IEEE operations on IEEE values in preparation for IEEE rounding. For a detailed description of the representative format, we first define the set of values $\mathcal{IRES}_{n,p}$ and the set of factorings $RFfact_{n,p}$ on that the representative format is based and show some properties of these sets.

Definition 2.21 *The set of result values $\mathcal{IRES}_{n,p}$ is defined by*

$$\mathcal{IRES}_{n,p} := \{0\} \cup \{x \in I\!R \mid 2^{-2^n-2p+6} < abs(x) < 2^{2^n+p-3}\} \cup \mathcal{SPE}.$$

and the set of RF factorings $RFfact_{n,p}$ is defined by

$$RFfact_{n,p} = \{(s,e,f) \mid \exists x \in \mathcal{IRES}_{n,p} : val(s,e,f) = rep_{p-e}(x) \ AND \ (f < 4 \ AND \ (f \geq 1 \ OR \ (f \ is \ multiple \ of \ 2^{-p+1}))\}$$

If $x \stackrel{p-e_{RF}}{=} val(s_{RF}, e_{RF}, f_{RF})$ for a value $x \in \mathcal{IRES}_{n,p}$ and a factoring $(s_{RF}, e_{RF}, f_{RF}) \in RFfact_{n,p}$, then (s_{RF}, e_{RF}, f_{RF}) is called a RF factoring representation *of x.*

Note, that because special values have an exact and normalized significand representation and we defined the exponent of special factorings to have properties like $e_{sp} = e_{max}+1$, we get $SPEfact \subset RFfact_{n,p}$, and special value factorings are RF factorings of the corresponding special values.

Lemma 2.24 *This lemma consists of four parts:*

(a) Each exact result of an IEEE operation on IEEE values has a value from $\mathcal{IRES}_{n,p}$.

(b) Each value $x \in \mathcal{IRES}_{n,p}$, has at least one RF factoring representation $(s_{RF}, e_{RF}, f_{RF}) \in RFfact_{n,p}$ with $val(s_{RF}, e_{RF}, f_{RF}) = rep_{p-e_{RF}}(x)$.

(c) Each exact result x of an IEEE operation on IEEE values in single precision or double precision has at least one RF factoring representation $(s_{RF}, e_{RF}, f_{RF}) \in RFfact_{11,53}$.

(d) If $(s_{RF}, e_{RF}, f_{RF}) \in RFfact_{11,53}$ is a RF factoring of the exact result $x \in \mathcal{IRES}_{11,53}$ then for mode $\in \{RZ, RNE, RI, RMI\}$, IEEE rounding of x in single and double precision can also be computed on the factoring (s_{RF}, e_{RF}, f_{RF}) by

$$r_{mode}(x) = val(iround_{mode}(s_{RF}, e_{RF}, f_{RF})).$$

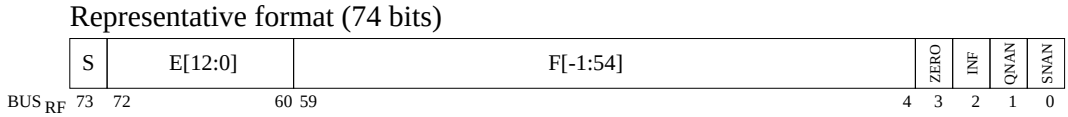

Figure 2.8: Representative format for single and double precision.

Proof: We proof the four parts of the lemma separately:

(a) Obviously all possible zero or special value results of IEEE operations are included in $\mathcal{IRES}_{n,p}$. All non-zero representable results have a magnitude that is larger than 2^{-2^n-2p+6} and smaller than 2^{2^n+p-3} (see section 2.4.2). All real numbers with these properties are inluded in $\mathcal{IRES}_{n,p}$, so that there can not be an IEEE result, that is not included in $\mathcal{IRES}_{n,p}$.

(b) If (s,e,f) is an arbitrary factoring of $x \in \mathcal{IRES}_{n,p}$, and (s',e',f') is the corresponding normalized factoring $(s',e',f') = \eta(s,e,f)$, then obviously, $(s',e',rep_{53}(f')) \in RFfact_{n,p}$ and $(s',e',rep_{53}(f'))$ is a RF factoring of x. Thus, indeed each x has at least one RF factoring in $RFfact_{n,p}$.

(c) Part (c) follows from part (a) and part (b) using that $\mathcal{IRES}_{8,24} \subset \mathcal{IRES}_{11,53}$.

(d) If $(s_{RF},e_{RF},f_{RF}) \in RFfact_{11,53}$ is a RF factoring of the exact result $x \in \mathcal{IRES}_{11,53}$, then $val(s_{RF},e_{RF},f_{RF}) = rep_{53-e_{RF}}(x)$. There is a unique factoring (s_{RF},e_{RF},f) with $x = val(s_{RF},e_{RF},f)$. For the significand f of this factoring we get $f_{RF} = rep_{53}(f)$. Because for both, single and double precision, $p \leq 53$, it follows from lemma 2.15 with $p' = 53$, that $iround_{mode}(s_{RF},e_{RF},f_{RF}) = iround_{mode}(s_{RF},e_{RF},f)$ and by definition 2.8 of $iround_{mode}$ we get $val(iround_{mode}(s_{RF},e_{RF},f_{RF})) = rnd_{mode}(x)$, as required. □

Corollary 2.25 *The previous lemma has shown that each exact result x of an IEEE operation on IEEE values in both single and double precision has at least one RF factoring representation $(s_{RF},e_{RF},f_{RF}) \in RFfact_{11,53}$ and that IEEE rounding of x can be computed by rounding of (s_{RF},e_{RF},f_{RF}).*

The representative format is an encoding of RF factorings $(s_{RF},e_{RF},f_{RF}) \in RFfact_{11,53}$. From the conditions on RF factorings in definition 2.21 it follows, that the exponent e_{RF} can be represented by a 13-bit 2's complement representation $e_{RF} =< \text{E}_{RF}[12:0] >_2$ and the representation of the significand f_{RF} requires 56-bits: $f_{RF} =< \text{F}_{RF}[-1:54] >_{neg}$.

Like in the unpacked and normalized format, the special values and zeros are indicated by the 4 additional bits: ZERO, INF, QNAN, and SNAN. For these cases, the sign S_{RF} and the significand representation $\text{F}_{RF}[1:52]$ correspond to the IEEE representation and the bits $\text{F}_{RF}[-1]$, $\text{F}_{RF}[53:54]$ are defined to be zero. Additionaly, for zeros, $\text{F}_{RF}[0{:}52] = 0^{53}$ and the exponent representation is not valid. For $+/-\infty$ and NaNs, we define $\text{F}_{RF}[0] = 1$ and $e_{RF} = e_{max} + 1$ like in the unpacked and the normalized format.

Figure 2.8 depicts a representation in the representative format $BUS_{RF}[73:0]$ with bit fields:

$$\begin{aligned} \text{S}_{RF} &= BUS_{RF}[73] \\ \text{E}_{RF}[12:0] &= BUS_{RF}[72:60] \\ \text{F}_{RF}[-1:54] &= BUS_{RF}[59:4] \end{aligned}$$

$$\begin{aligned} \text{ZERO}_{RF} &= BUS_{RF}[3] & \text{INF}_{RF} &= BUS_{RF}[2] \\ \text{QNAN}_{RF} &= BUS_{RF}[1] & \text{SNAN}_{RF} &= BUS_{RF}[0]. \end{aligned}$$

Definition 2.22 *We define the function* RF : $RFfact_{n,p} \longrightarrow \{0,1\}^{74}$*, that computes the representation of an RF factoring* $(s_{RF}, e_{RFF}, f_{RF}) \in RFfact_{n,p}$ *in the representative format. With* $e_{RF} = <\mathrm{E}_{RF}[12{:}0]>_2$ *and* $f_{RF} = <\mathrm{F}_{RF}[-1{:}54]>_{neg}$ *for representable numbers and quiet NaNs, the function* RF *is defined by*

$$\mathrm{RF}(s,e,f) = \begin{cases} (\mathrm{S}, & 0^{13}, & 00, 0^{54}, & 1,0,0,0) & \text{if } f = 0 \\ (\mathrm{S}, & 00,1,0^{10}, & 01, 0^{54}, & 0,1,0,0) & \text{if } f = f_\infty \\ (\mathrm{S}, & 00,1,0^{10}, & 01,0,\mathrm{F}_{RF}[2{:}52],00, & 0,0,1,0) & \text{if } f = f_{qNaN} \\ (\mathrm{S}, & 00,1,0^{10}, & 01,1,0^{53}, & 0,0,0,1) & \text{if } f = f_{sNaN} \\ (\mathrm{S}, & \mathrm{E}_{RF}[12{:}0], & \mathrm{F}_{RF}[-1{:}54], & 0,0,0,0) & \text{otherwise.} \end{cases}$$

In the opposite direction the function $fact_{RF} : \{0,1\}^{74} \longrightarrow RFfact_{n,p}$ *computes the RF factoring that is represented by* $BUS_{RF}[73:0]$ *in the representative format. With*

$$(\mathrm{S}, \mathrm{E}_{RF}[12:0], \mathrm{F}_{RF}[-1{:}54], \mathrm{ZERO}, \mathrm{INF}, \mathrm{QNAN}, \mathrm{SNAN}) = BUS_{RF}[73:0],$$

the function $fact_{RF}$ *is defined by*

$$fact_{RF}(BUS_{RF}[73{:}0]) = \begin{cases} (s, e_0, 0) & \text{if ZERO} \\ (s, e_\infty, f_\infty) & \text{if INF} \\ (s, e_{qNaN}, f_{qNaN}) & \text{if QNAN} \\ (s, e_{sNaN}, f_{sNaN}) & \text{if SNAN} \\ (s, <\mathrm{E}_{RF}[12{:}0]>_2, <\mathrm{F}_{RF}[-1{:}54]>_{neg}) & \text{otherwise.} \end{cases}$$

Note, that not all possible bit combinations from $\{0,1\}^{74}$ are valid representations in the representative format. A representation $BUS_{RF}[73:0]$ could be invalid for two reasons:

1. $BUS_{RF}[73:0]$ would be the RF representation of a number that is not in $\mathcal{IRES}_{11,53}$.
2. the significand conditions from definition 2.21 for RF factorings are not fullfilled.

We are only interested in the second case, because we will only deal with numbers from $\mathcal{IRES}_{11,53}$. Therefore, we formulate the conditions that have to be fullfilled for the significand of RF representations at bit level in the following:

Corollary 2.26 *The conditions on the significand* $f_{RF} =< \mathrm{F}_{RF}[-1:54] >_{neg}$ *of a RF factoring, namely* ($f_{RF} < 4$ *AND* ($f_{RF} \geq 1$ *OR* (f_{RF} *is multiple of* 2^{-52})) *are fullfilled, iff* ($\mathrm{F}_{RF}[-1] \vee \mathrm{F}_{RF}[0] \vee \overline{\mathrm{F}_{RF}[54]}$). *This means, that either one of the most significant two bits* $\mathrm{F}_{RF}[-1:0] = BUS_{RF}[59:58]$ *has to be one or the least significant bit* $\mathrm{F}_{RF}[54] = BUS_{RF}[4]$ *has to be zero.*

2.6.5 Gradual Result Format

Also in the *gradual result format*, the results of IEEE operations on IEEE values should be represented. But in contrast to the representative format, in the gradual result format already part of the rounding has been computed on the exact IEEE results. For a detailed description of the gradual result format, first we introduce the set of factorings, on that the gradual result format is based.

Definition 2.23 *The set of GF factorings* $GFfact$ *is defined by:*

$$\begin{aligned} GFfact \;=\; \{((s_{GF}, e_{GF}, f_{GF}), \mathrm{TINX}, \mathrm{TINC}) \;\mid\; & \exists (s,e,f) \in FACT(\mathcal{IRES}_{11,53}) : \\ & ((s_{GF}, e_{GF}, f_{GF}), \mathrm{TINX}, \mathrm{TINC}) = post\_norm(sgrnd1(\eta(s,e,f)))\} \end{aligned}$$

By this definition, GF factorings are the result of a gradual rounding step according to the intermediate result from lemma 2.20. Thus, IEEE rounding can also be computed on these factorings:

Corollary 2.27 *If* $((s_{GF}, e_{GF}, f_{GF}), \text{TINX}, \text{TINC}) \in GFfact$ *is the GF factoring of the value* $x \in \mathcal{IRES}_{11,53}$*, then for mode* $\in \{RZ, RNE, RI, RMI\}$ *IEEE rounding of* x *in single and double precision can be computed on* $((s_{GF}, e_{GF}, f_{GF}), \text{TINC}, \text{TINX})$ *according to lemma 2.20 by*

$$iround_{mode}(s, e, f) = exp\_rnd_{mode \star s_{GF}}(post\_norm(sgrnd2_{mode \star s_{GF}}(\lceil \eta_{e_{min}} \rceil((s_{GF}, e_{GF}, f_{GF}), \text{TINC}, \text{TINX}))))$$

The gradual result format is an encoding of GF factorings $((s_{GF}, e_{GF}, f_{GF}), \text{TINC}, \text{TINX}) \in GFfact$. Because the range of the represented numbers is not changed significantly by the previous gradual rounding step, also in this case the exponent e_{GF} is represented by 13 bits: $e_{GF} = <\text{E}_{GF}[12:0]>_2$. The significand, which was rounded at position 52 and which was post-normalized, is represented by $f_{GF} = <\text{F}_{GF}[0:52]>_{neg}$.

Like in the unpacked, normalized and representative format the special values and zeros are indicated by 4 additional bits. For these cases, the sign S_{GF} and the significand representation $\text{F}_{GF}[1:52]$ correspond to the packed IEEE representation. For zeros, $\text{F}_{GF}[0] = 0$ and the exponent representation is not valid. For $+/-\infty$ and NaNs, we define $\text{F}_{GF}[0] = 1$ and $e_{GF} = e_{max}+1$ like in the unpacked, the normalized and the representative format. Thus, in the gradual result format the significand is normalized for all non-zero numbers, i.e., if the additional bit $\text{ZERO}_{GF} = 0$, then $\text{F}_{GF}[0]$ must be 1.

Moreover, the two rounding tags TINC_{GF} and TINX_{GF} from the previous gradual rounding step are included in the number representations of the gradual result format. For special values, which have an exact representation, TINC_{GF} and TINX_{GF} have to be zero, so that no rounding will be computed for them.

Figure 2.9 depicts a bus in the gradual result format indexed by $BUS_{GF}[72:0]$ with bit fields:

$$\begin{aligned} \text{S}_{GF} &= BUS_{GF}[72] \\ \text{E}_{GF}[12:0] &= BUS_{GF}[71:59] \\ \text{F}_{GF}[0:52] &= BUS_{GF}[58:6] \end{aligned}$$

$$\begin{aligned} \text{TINC}_{GF} &= BUS_{GF}[5] & \text{TINX}_{GF} &= BUS_{RF}[4] \\ \text{ZERO}_{GF} &= BUS_{GF}[3] & \text{INF}_{GF} &= BUS_{RF}[2] \\ \text{QNAN}_{GF} &= BUS_{GF}[1] & \text{SNAN}_{GF} &= BUS_{GF}[0]. \end{aligned}$$

Definition 2.24 *We define the function* $\text{GF} : GFfact \longrightarrow \{0,1\}^{73}$*, that computes the representation of an GF factoring* $((s_{GF}, e_{GF}, f_{GF})\text{TINC}, \text{TINX}) \in GFfact_{n,p}$ *in the gradual result format. With* $e_{GF} = <\text{E}_{GF}[12:0]>_2$ *and* $f_{GF} = <\text{F}_{GF}[0:52]>_{neg}$ *for representable numbers and quiet NaNs, the function* GF *is defined by*

$$\text{GF}(s,e,f) = \begin{cases} (\text{S}, & 0^{13}, & 0,0^{52}, & 00, & 1,0,0,0) & \text{if } f = 0 \\ (\text{S}, & 00,1,0^{10}, & 1,0^{52}, & 00, & 0,1,0,0) & \text{if } f = f_{\infty} \\ (\text{S}, & 00,1,0^{10}, & 1,0,\text{F}_{GF}[2:52], & 00, & 0,0,1,0) & \text{if } f = f_{qNaN} \\ (\text{S}, & 00,1,0^{10}, & 1,1,0^{51}, & 00, & 0,0,0,1) & \text{if } f = f_{sNaN} \\ (\text{S}, & \text{E}_{GF}[12:0], & \text{F}_{GF}[0:52], & \text{TINC, TINX}, & 0,0,0,0) & \text{otherwise.} \end{cases}$$

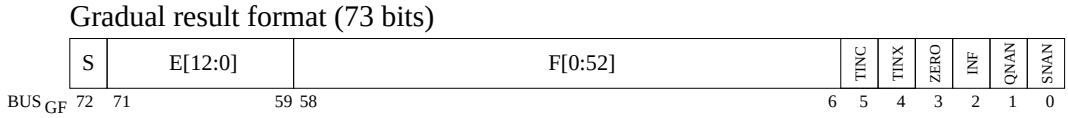

Figure 2.9: Gradual result format for single and double precision.

In the opposite direction the function $fact_{GF} : \{0,1\}^{73} \longrightarrow GFfact_{n,p}$ *computes the GF factoring that is represented by* $BUS_{GF}[72:0]$ *in the representative format. With*

$$(\text{S}, \text{E}_{GF}[12:0], \text{F}_{GF}[0{:}52], \text{TINC}, \text{TINX}, \text{ZERO}, \text{INF}, \text{QNAN}, \text{SNAN}) = BUS_{GF}[72:0],$$

the function $fact_{GF}$ *is defined by*

$$fact_{GF}(BUS_{GF}[72{:}0]) = \begin{cases} ((s, e_0, 0), 00) & \textit{if } \text{ZERO} \\ ((s, e_\infty, f_\infty), 00) & \textit{if } \text{INF} \\ ((s, e_{qNaN}, f_{qNaN}), 00) & \textit{if } \text{QNAN} \\ ((s, e_{sNaN}, f_{sNaN}), 00) & \textit{if } \text{SNAN} \\ ((s, <\text{E}_{GF}[12{:}0]>_2, <\text{F}_{GF}[0{:}52]>_{neg}), \text{TINC}, \text{TINX}) & \textit{otherwise.} \end{cases}$$

Chapter 3

FP Microarchitectures

In the previous section the definitions and the requirements of the IEEE FP standard 754 were presented. From this section we know that an IEEE compliant FP implementation has to implement FP additions/subtractions, FP multiplications, FP divisions, FP square-roots, FP comparisons and FP conversions in hardware or in software.

The FP operations have different importance. One measure of the importance of these FP operations could be the frequency of their usage in an average workload of current microprocessors. As such a measure the frequency of the operations in traces of the SPEC92fp benchmark suite [17] is depicted in figure 3.1. Obviously, the FP addition/subtraction and the FP multiplication are the most frequent arithmetic FP operations in these Benchmark traces. This result agrees with the analysis from [26]. To accelerate the FP performance of a microprocessor, it would make sense to spend the most effort on accelerating the implementation of the most frequent FP operations. It can be seen in table 3.1 from the latencies of the FP operations in commercial microprocessors that indeed the FP addition and the FP multiplication, which are used frequently, are implemented much faster than the FP division, which is only rarely used. Because the IEEE FP standard even allows to implement parts of the FP computations in software, some very infrequent operations like the FP square-root or the FP division-rest even have no hardware realization in most commercial microprocessors. Although the cheap and slow implementation of operations, that are infrequently used, might be cost-effective, one should also think about the effect, that perhaps some operations are infrequently used and tried to be avoided only because current microprocessors provide such a poor performance for these operations. This question could only be answered by the use of benchmarks and compilers, that use as less as possible about the hardware implementation details.

We base our choice of which arithmetic FP operations are implemented in hardware on the processor model, into which our FP designs will be integrated later. We will use a pipelined RISC-processor from [23], that implements the R3000 instruction set. For this reason the performance of the FP designs will also be determined on R3000 traces of the SPEC92fp Benchmarks. The R3000 instruction set includes the FP addition/subtraction, FP multiplication, FP division, FP test, FP conversion, FP absolute value and FP negative value. Therefore, we propose IEEE compliant FP designs that support exactly these arithmetic operations in hardware.

We present three basic microarchitectures of our floating-point designs in the following. The main differences of these microarchitectures is the amout of rounding hardware that is shared between the functional units. If the functional units share some rounding hardware at all, the FP microarchitecture is mainly determined by the specification of

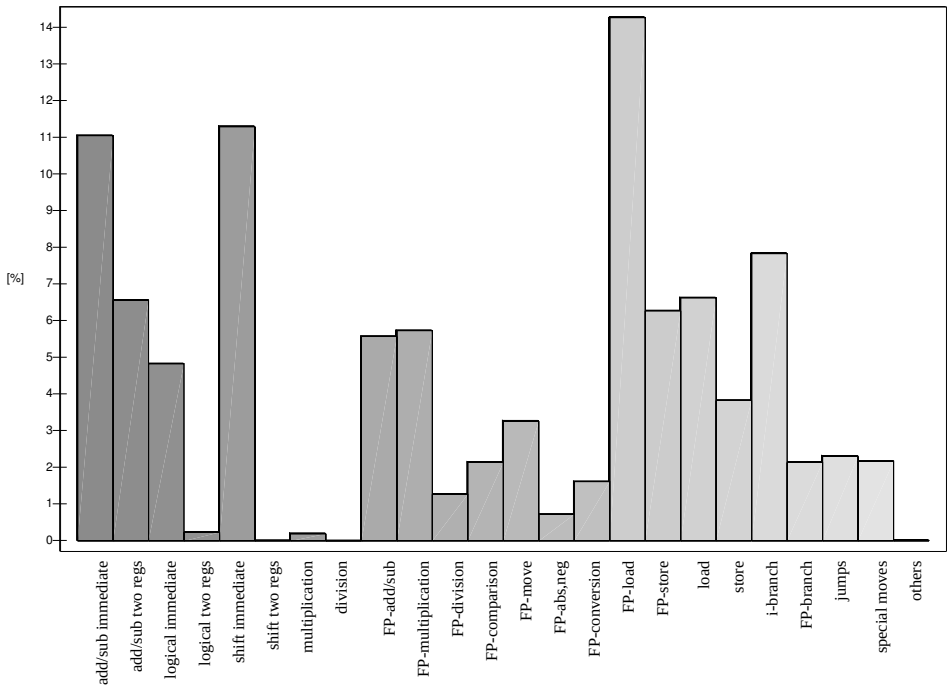

Figure 3.1: Operation frequencies in the traces of the SPECfp 92 benchmarks.

the intermediate FP representation at the interfaces between the functional units and the shared rounding hardware. Our three rounding microarchitectures are based on the intermediate FP representations, that were defined in the previous section. The lemmas 2.7, 2.20 and 2.8 and corollary 2.21 about the different possible partitionings of IEEE rounding computations already suggest the possible partitionings of the rounding implementations.

(I) In the first microarchitecture all the rounding computations are concentrated in a shared general rounding unit. This rounding unit considers the rounding for all IEEE results including the exponent wrapping and the FP exceptions for both single and double precision operations. A basic specification of such a rounder was first described in [10]. Thereafter, this rounder was implemented by our group, resulting in a version that will be included in [23], where also a rigorous correctness proof of the compliance with the IEEE rounding definition will be found. This rounder is further optimized in this thesis. The interface between the functional units and the shared general rounder is the RF factoring representation from definition 2.21. We require, that the functional units compute a RF factoring representation (s_{RF}, e_{RF}, f_{RF}) of the exact result $exact_{op}$. The shared general rounder then has to compute IEEE rounding on the RF factoring representation $iround(s_{RF}, e_{RF}, f_{RF})$. Lemma 2.7 guarantuees, that the IEEE rounding of the RF factoring (s_{RF}, e_{RF}, f_{RF}) agrees with IEEE rounding of $exact_{op}$ including the cases of denormalized and special values results, exceptions and exponent wrapping. In this microarchitecture the integrated packed FP representation is used in the memory and in the registerfile.

(II) In the second microarchitecture, the rounding for the case of normalized double precision results is computed within each functional unit and this rounded result is fixed for all the remaining cases in a second rounding step implemented by a shared gradual rounding unit. For the integrated rounding in the functional units

processor	latency					
	ALU	FP add	FP mult	FP div single	FP div double	FP sqrt
ULTRA-Sparc 1	1	3	3	12	22	12-22
ULTRA-Sparc 3	1	4	4	12	17	12-24
Pentium Pro	1	3	5	17	36	-
PowerPC	1	5	5	17	21	-
Alpha 21064	1	4	4	34	63	-
Alpha 21164	1	4	4	19	31	-
Alpha 21264	1	4	4	12	15	-
R10000	1	2	2	19	33	-
PA-8000	1	3	3	31	31	-

Table 3.1: Latencies of floating-point operations in commercial microprocessors.

assuming normalized, double precision operands and results, several algorithms from literature could be used. The implementation of the gradual rounder is based on the theory from [21] about gradual rounding. This rounding technique is applied in this thesis for full IEEE compliant rounding including the handling of denormalized results, special values, exceptions and exponent wrapping. The interface between the functional units and the gradual rounder is specified by the gradual result format. We require, that if an exact or an RF factoring of the exact result $exact_{op}$ is given by (s_{ex}, e_{ex}, f_{ex}), the functional units have to compute the GF factoring (see definition 2.23) $((s_{GF}, e_{GF}, f_{GF}), \text{TINC}_{GF}, \text{TINX}_{GF}) = ground1(s_{ex}, e_{ex}, f_{ex})$. This computation already includes a first gradual rounding step by the gradual rounding function $ground1$, which assumes a normalized double precision result. The second gradual rounding step is then computed in the shared gradual rounding unit by the function $ground2((s_{GF}, e_{GF}, f_{GF}), \text{TINC}_{GF}, \text{TINX}_{GF})$. Corollary 2.21 and lemma 2.20 guarantuees, that the sequence of the rounding by the gradual rounding functions $ground1$ and $ground2$ on the factoring (s_{ex}, e_{ex}, f_{ex}) simulates IEEE rounding of the factoring (s_{ex}, e_{ex}, f_{ex}) including the cases of denormalized and special values results, exceptions and exponent wrapping. Also in this microarchitecture the integrated packed FP representation is used in the memory and in the registerfile.

(III) By the third rounding architecture a completely new architecture for an IEEE compliant FPU is suggested. In this architecture no rounding hardware is shared, but each functional unit contains a dedicated rounding implementation that computes full IEEE rounding considering denormal and special values, exceptions and exponent wrapping. The special problem with the implementation of this microarchitecture is the implementation if the floating-point multiplication. The floating-point multiplier conventionally requires normalized significands in its operands and delivers an almost normalized result. For the fast integration of IEEE rounding into the FP multiplier, the significand has to be rounded in parallel to the mulplication computations. For the case of denormalized results this rounding has to be computed at a variable rounding position, that could be at each position within the significand. The idea, how to integrate such a variable position rounding into the multiplication implementation is the key concept for this microarchitecture. Such a multiplication implementation including variable position rounding will be presented later (see also

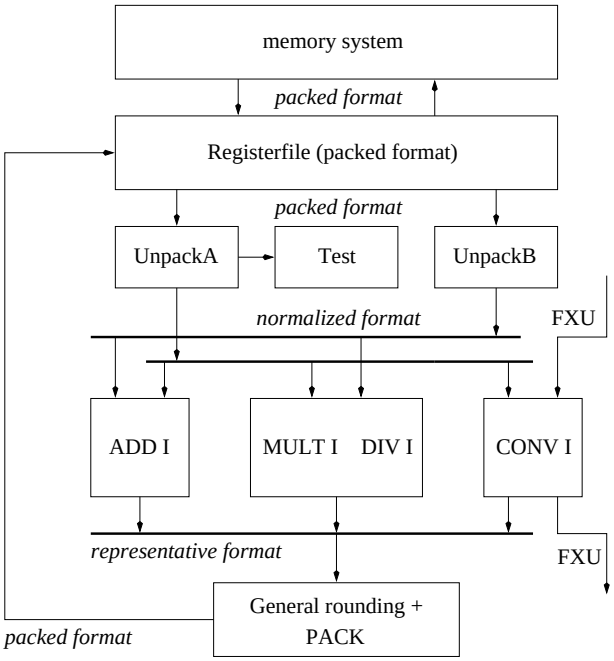

Figure 3.2: FP unit microarchitecture using a shared general rounding unit

[37]). Because such a multiplication implementation allows to work on normalized FP representations (even for denormalized values) as inputs and outputs, the internal FP representations can be changed to normalized NF factoring representations for this microarchitecture. Thus, the registerfile contains the operands in the NF factoring representation and the functional units have to compute the NF factoring of the IEEE rounded result. This is specified by the rounding function *nround*, so that if an exact or an RF factoring of the exact result $exact_{op}$ is given by (s_{ex}, e_{ex}, f_{ex}), the functional units have to compute the NF factoring $nround(s_{ex}, e_{ex}, f_{ex})$. Definition 2.8 and lemma 2.8 guarantuee that this function computes IEEE rounding of the exact result $exact_{op}$ including the cases of denormalized and special values results, exceptions and exponent wrapping. The computation of this rounding function according to lemma 2.8 contains the computation of the normalized significand rounding function $n\_sig\_rnd_{mode*s}$, where the significand has to be rounded at the variable rounding position $vp = vp = (p-1) - max\{0, e_{min} - e_{ex}\}$ (see definition 2.9) that depends on the exponent e_{ex}, so that the rounding position could vary in a wide range as mentioned above.

In the following the main structures and implementation details for the three microarchitectures are described:

Rounding architecture I using general rounding Figure 3.2 depicts the basic structure of this FP rounding architecture. The operands are stored in the registerfile in the packed representation. The unpacking units convert them to a representation in the normalized format. The unpacking is computed in two steps, a conversion from the packed to the unpacked format, followed by a conversion from the unpacked to the normalized format. The normalized operands are necessary for multiplications and divisions. But as in our designs the whole normalization easily fits into one clock cycle, there is no overhead

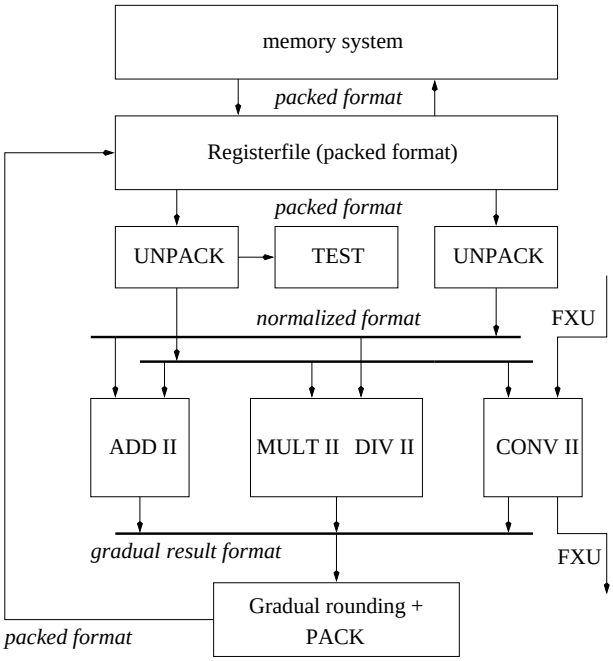

Figure 3.3: FP microarchitecture using a shared gradual rounding unit

in also providing the normalized representation instead of the unpacked representation for additions.

After unpacking, a representative representation (RF factoring) of the exact result of the operation has to be computed. This can be achieved with simple standard algorithms for the addition, multiplication and subtraction, including the computation of a sticky bit like described in the representative computation according to lemma 2.11. The representative representation of the result is fed into the shared general rounding unit, that delivers and feeds back the packed representation of the rounded result into the registerfile. This general rounder is also able to deal with denormalized results, special value results, exponent wrapping and exceptions. The computations, that have to be computed in the general rounder are quite complex. This circuit has to deal with leading zero detections to find out the range of the number, with a normalization shift for results with values of normalized umbers, with a denormalization shift for results with values of denormalized numbers, with significand rounding, a post-normalization shift, exponent rounding and exponent wrapping.

Rounding architecture II using gradual rounding In contrast to the general rounding, in the gradual rounding architecture II (see figure 3.3), a part of the rounding is shifted to the functional units, so that the functional units output normalized results (GF factorings) in the gradual result format. This result is then rounded following the IEEE specifications in a second step in the gradual rounding unit [21], that delivers a packed representation of the rounded result to the registerfile. Because the input to the gradual rounding unit is already normalized, gradual rounding is simpler than general rounding. In particular the leading zero prediction and the normalization shift can be saved from the general rounding implementation

In the functional units the gradual rounding (computation of the rounding function $ground1$) for normalized double precision numbers has to be computed. There are several

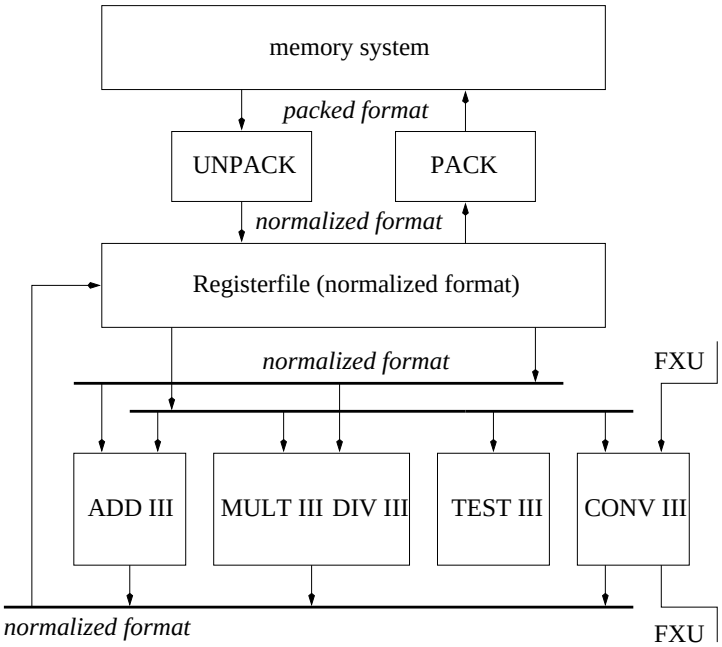

Figure 3.4: FP microarchitecture using a variable position rounding

algorithms from literature [9, 27, 31, 32, 33, 34, 36, 40, 44, 45] for each arithmetic operation that could be used for this situation. In the context of this thesis we introduced injection-based rounding and developed based on this technique new algorithms for FP addition rounding (see also [40]) and FP multiplication rounding (see also [9, 11]), that are to the best of our knowledge the fastest FP addition rounding and FP multiplication rounding algorithms published todate. For the division impledmentation the Newton-Raphson iteration (see also [26, 28, 23])is used. For the initial reciprocal approximation we designed a new fast implementation of a linear approximation formula (see also [36, 39]).

Rounding architecture III using variable position rounding In contrast to both previous architectures, in this variable position rounding implementation (see figure 3.4), we do not operate on the packed format in the registerfile, but on the normalized format in the registerfile [37]. In this way, the normalization conversions inside the unpack and pack units are moved towards the communication between the registerfile and the memory system. Fortunately, in our designs these conversions can be integrated in the load and store operations without increasing their latency.

Among the arithmetic operations, the multiplication becomes most difficult, because for denormalized results, variable position rounding becomes necessary. An algorithm for fast variable position rounding in multipliers was developed in the context of this work and is presented in [37]. In this way, the multiplication implementation including full IEEE compliant rounding also for denormalized results is the core for the use of rounding architecture III with the normalized internal FP representation (NF factoring).

In the FP addition, the variable position rounding does not cause problems, because using our FP adder implementation from [40] all denormal results are exact and do not need to be rounded. Some selections for the computations on special values and the exception handling has to be included in the implementation, so that the implementation from [40] has to be modified for this rounding architecture III to implement full IEEE

compliant rounding.

Further options Apart from the three different rounding architectures described above, we vary the FP multiplication implementation to contain either a full-sized or a half-sized adder tree that both use Booth2 recoding [30, 3, 1]. We have developed an improved cost formula for these adder trees was in [30]. Moreover we consider three different implementations of the division based on the Newton-Raphson iteration [26, 23] with different starting accuracies of the initial reciprocal approximation. These multiplicative division implementations are integrated into the implementations of the multiplier. For the initial reciprocal approximation we use the fast implementation of a linear approximation formula that we already presented in [36, 39] with an absolute approximation error bounded by 2^{-8}, 2^{-16} and 2^{-28}, respectively.

Chapter 4

Basic FP Operations

4.1 Internal Format Conversions

4.1.1 Unpacking I-III (packed $\longrightarrow$ normalized format)

This section describes the unpacking, i.e., the conversion from a packed single or double precision FP representation to the corresponding FP representation in the normalized format. The choice whether we have a single or double precision input is signaled by the bit DBL. In addition to this bit the packed FP input is given by $BUS_{PF}[63:0]$ (section2.6.1). The FP output in the normalized format is denoted by $BUS_{NF}[69:0]$ (section2.6.3).

First, we deal with the problem to extract the bits belonging to sign, exponent and significand from the packed representation integrating the cases for single and double precision. Regarding the sign this is an easy task, namely $\mathrm{S}_{PF} = BUS_{PF}[63]$. The exponent and significand extractions are implemented in the *exponent-extract* circuit and in the *significand-extract* circuit in figure 4.1, that can be realized by a row of muxes each, described by (see equation2.75-equation2.80):

$$\mathrm{E}_{PF}[10:0] = \begin{cases} BUS_{PF}[62:52] & \text{if DBL} \\ (000, BUS_{PF}[62:55]) & \text{otherwise,} \end{cases}$$

$$\mathrm{F}_{PF}[1:52] = \begin{cases} BUS_{PF}[51:0] & \text{if DBL} \\ BUS_{PF}[54:3] & \text{otherwise.} \end{cases}$$

The conversion from the packed to the normalized format can be constructed in two steps: (i) a conversion from the packed to the unpacked format (see section 2.6.2, p.40) followed by (ii) a conversion from the unpacked to the normalized format (see section 2.6.3, p.42):

(i) The unpacked format differs from the packed format by 5 additional bits and a different exponent representation:

- The conditions for the 5 additional bits: F[0], SNAN, QNAN, INF, and ZERO, can be easily read off from equation2.81. To implement these conditions three zero testers are necessary according to:

$$\begin{aligned} \mathrm{FZERO} &= is\_zero(\mathrm{F}_{PF}[1:52]) \\ \mathrm{EZERO} &= is\_zero(\mathrm{E}_{PF}[10:0]) \\ \mathrm{EONE} &= \begin{cases} is\_zero(\overline{\mathrm{E}_{PF}[10:8]}, \overline{\mathrm{E}_{PF}[7:0]}) & \text{if DBL} \\ is\_zero(\mathrm{E}_{PF}[10:8], \overline{\mathrm{E}_{PF}[7:0]}) & \text{otherwise} \end{cases} \\ &= is\_zero(\mathrm{E}_{PF}[10:8] \otimes \mathrm{DBL}, \overline{\mathrm{E}_{PF}[7:0]}) \end{aligned}$$

Based on FZERO, EZERO, and EONE, the additional bits can be computed by:

$$\begin{array}{rclcrcl} \mathrm{F}_{UF}[0] & = & \overline{\mathrm{EZERO}} & & \mathrm{ZERO} & = & \mathrm{FZERO} \wedge \mathrm{EZERO} \\ \mathrm{INF} & = & \mathrm{FZERO} \wedge \mathrm{EONE} & & \mathrm{SNAN} & = & \mathrm{F}[0] \wedge \mathrm{EONE} \\ \mathrm{QNAN} & = & \overline{\mathrm{FZERO}} \wedge \overline{\mathrm{F}[0]} \wedge \mathrm{EONE}. & & & & \end{array}$$

This completes the description of the *additional bits* circuit in figure 4.1.

- The exponent representation $e_{PF}[10:0]$ has to be converted from packed to two's complement representation, where the single and double precision case have to be integrated. One can easily check, that the following equation describes the conversion for non-zero packed exponent representations (EZERO = 0) given by equation2.82 except a missing increment:

$$\mathrm{E}_1[11:0] \quad = \quad \begin{cases} (\overline{\mathrm{E}_{PF}[10]}^2, \mathrm{E}_{PF}[9:0]) & \text{if DBL} \\ (\overline{\mathrm{E}_{PF}[7]}^5, \mathrm{E}_{PF}[6:0]) & \text{otherwise.} \end{cases} \tag{4.1}$$

This missing increment is postponed to the *exponent adjustment* circuit. Note, that only the most significant 5 bits are differing, so that the selection can be implemented by 5 muxes. In the next step, we integrate the case EZERO = 1:

$$\mathrm{E}_2[11:0] \quad = \quad \begin{cases} (1^2, \overline{\mathrm{DBL}}^3, 0^6, 1) & \text{if EZERO} \\ \mathrm{E}_1[11:0] & \text{otherwise.} \end{cases} \tag{4.2}$$

Note, that the value of $< \mathrm{E}_2[11:0] >_2 = < \mathrm{E}_{UF}[11:0] >_2 - 1$ is also defined to be too small by one in the case of EZERO = 1, so that the postponed exponent increment will correct the exponent value for all cases. In this way equation 4.1 and equation 4.2 specify the implementation of the *exponent conversion* circuit in figure 4.1.

(ii) To convert from the unpacked format to the normalized format, we have to implement the exponent and significand conversion (unbounded normalization shift for denormalized numbers) according to equations 2.86 and 2.87.

To simplify the exponent adjustment, we compute $lzi = lz - 1 = lzero(\mathrm{F}[0:52]) - 1$ in the *shift amount* circuit. This is done in two paths and a final selection depending on the value of F[0]:

$$< lzi[11:0] >_2 = lz - 1 = \begin{cases} < (0^6, \mathrm{LZERO}(\mathrm{F}[1:52], 0^{12})[5:0]) >_2 & \text{if } \mathrm{F}[0] = 0 \\ < 1^{12} >_2 = -1 & \text{if } \mathrm{F}[0] = 1 \end{cases} \tag{4.3}$$

To compute $\mathrm{LZERO}(\mathrm{F}[1:52], 0^{12})[5:0]$, we use circuit *lzero* from [23] with t=64 and get the amount of leading zeros $lz - 1 = < \mathrm{LZI}[5:0] >$ for the case $\mathrm{F}[0] = 0$ represented by $\mathrm{LZI}[5:0]$. The most significant 0^6 in equation 4.3 are the sign extension to the 12-bit two's complement representation, because for $\mathrm{F}[0] = 0$, $lzi = lz - 1 \geq 0$ is non-negative. The normalization shift is then computed by:

- A left-shift of $\mathrm{F}_{UF}[0:52]$ by lz positions: $\mathrm{F}_{NF}[0:52] = (\mathrm{F}_{UF}[lz:52], 0^{lz})$. Because we do not know lz, but only $lzi = lz - 1$, we use a cyclic left-shifte, that computes the following function CLS on an 64-bit input $\mathrm{INPUT}[0:63]$ and a shift amount $sfta$ given in a 6-bit binary representation:

$$\mathrm{CLS}(\mathrm{INPUT}[0:63], sfta) = (\mathrm{INPUT}[sfta:63], \mathrm{INPUT}[0:sfta-1])$$

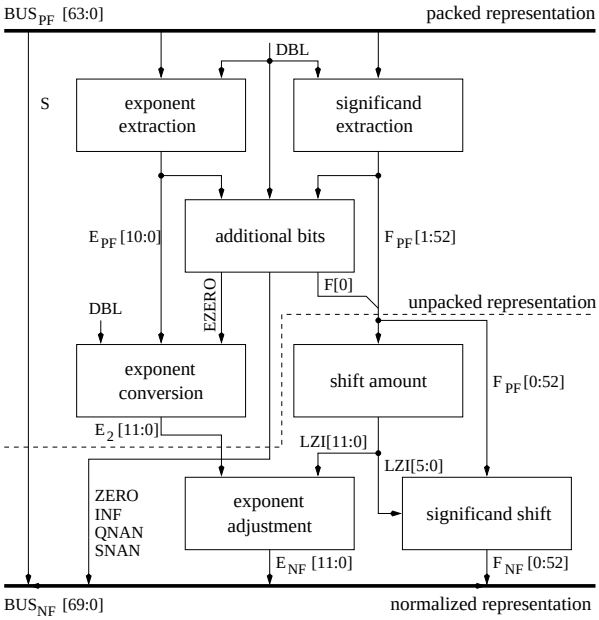

Figure 4.1: Unpack unit

With this circuit we compute in the *significand shift* circuit in figure 4.1

$$\begin{aligned} \text{F\_CLS}[0:63] &= \text{CLS}((\text{F}_{UF}[0:52], 0^{11}), < lzi[5:0] >) \\ &= (\text{F}_{UF}[< lzi[5:0] >: 63], \text{F}_{UF}[0 :< lzi[5:0] > -1]) \\ &= \begin{cases} (0, \text{F}_{UF}[lz:52], 0^{lz+10}) & \text{if } lzi[5:0] \neq 1^6 \\ (0, \text{F}_{UF}[0:52], 0^{10}) & \text{if } lzi[5:0] = 1^6 \end{cases}, \end{aligned}$$

so that we get, as required,

$$\text{F}_{NF}[0:52] = (\text{F}_{UF}[lz:52], 0^{lz}) = \text{F\_CLS}[1:53].$$

- An exponent adjustment $e_2 - lz$. The *exponent adjustment* circuit in figure 4.1 implements this subtraction including the postponed exponent increment $e_2 - lz + 1$ from the *exponent conversion* circuit:

$$\begin{aligned} < \text{E}_{NF}[11:0] >_2 &= < \text{E}_2[11:0] >_2 - lz + 1 \\ &= < \text{E}_2[11:0] >_2 - (lz - 1) \\ &= < \text{E}_2[11:0] >_2 - < \text{LZI}[11:0] >_2 \\ &= < \text{E}_2[11:0] >_2 + < \overline{\text{LZI}[11:0]} >_2 + 1 \end{aligned}$$

A 12-bit wide conditional sum adder is used to compute this sum. The additional 1 is fed into the carry-in input of the adder.

This completes the description of the unpack unit (figure 4.1) that outputs the normalized FP representation

$$BUS_{NF}[69:0] = (\text{S}_{PF}, \text{E}_{NF}[11:0], \text{F}_{NF}[0:52], \text{ZERO}, \text{INF}, \text{QNAN}, \text{SNAN}).$$

4.1.2 General Rounding I (representative $\longrightarrow$ packed format)

This section describes a general dual mode rounding unit that is able to round and to compress a FP number from the representative format $BUS_{RF}[73:0]$ (section 2.6.4) to the single precision or the double precision packed FP representation $BUS_{PF}[63:0]$ (section 2.6.1). The mode, whether the destination is single or double precision, is selected by the bit DBL. The additional inputs of the rounding mode by RMODE[1:0] and the trap handlers UNF_EN and OVF_EN select different IEEE rounding options. The IEEE rounding with these options has to be computed on the input factoring

$$(s_{RF}, e_{RF}, f_{RF}) = fact_{RF}(BUS_{RF}[73:0]).$$

Because the packed representation of the rounded result is based on the IEEE factoring of the rounded result, the packed representation of the rounded result can be specified according to definition 2.8 by

$$BUS_{PF}[63:0] = \mathrm{PF}(iround(s_{RF}, e_{RF} + wec, f_{RF})).$$

In addition to the rounding computations, the occurance of an overflow, underflow and inexact exception should be signaled by OVF, UNF, and INX, respectively.

We first consider to compute the IEEE factoring of the rounded result $(s_{PF}, e_{PF}, f_{PF}) = iround(s_{RF}, e_{RF}+wec, f_{RF})$. A conversion from the IEEE factoring (s_{PF}, e_{PF}, f_{PF}) to the packed representation $BUS_{PF}[63:0]$ (packing) by the function PF then yields the required result representation.

According to lemma 2.7, namely,

$$iround_{mode}(s, e, f) = exp\_rnd_{mode\star s}(post\_norm(sig\_rnd_{mode\star s}(\lfloor \eta_{e_{min}} \rfloor(s, e, f)))),$$

the rounding function $iround_{mode}(s_{RF}, e_{RF}+wec, f_{RF})$ is computed in four steps:

1. a factoring $(s_1, e_1+wec, rep_{53}(f_1))$ corresponding to the bounded normalization shift

$$(s_1, e_1+wec, f_1) = \lfloor \eta_{e_{min}} \rfloor(s_{RF}, e_{RF} + wec, f_{RF}),$$

2. significand rounding $(s_2, e_2+wec, f_2) = a\_sig\_rnd_{mode\star s_1}(s_1, e_1+wec, rep_{53}(f_1))$ (Note, that for the significand rounding definition 2.9, equation 2.49 and lemma 2.12, we have also $(s_2, e_2+wec, f_2) = a\_sig\_rnd_{mode\star s_1}(s_1, e_1+wec, f_1)$ for single and double precision),

3. a post-normalization shift $(s_3, e_3+wec, f_3) = post\_norm(s_2, e_2+wec, f_2), and$

4. exponent rounding $(s_{PF}, e_{PF}, f_{PF}) = exp\_rnd_{mode\star s_3}(s_3, e_3+wec, f_3)$.

We treat the implementation of these 4 steps separately in the next paragraphs (the structure of the whole implementation is depicted in figure 4.5):

Normalization shift & representative computation (1.) The bounded normalization shift is defined by equation 2.1. Using the definition of the function $TINY$ it can be described by:

$$\begin{aligned}(s_1, e_1+wec, f_1) &= \lfloor \eta_{e_{min}} \rfloor(s_{RF}, e_{RF} + wec, f_{RF}) \\ &= \begin{cases} \eta(s_{RF}, e_{RF}+wec, f_{RF}) & \text{if } \overline{TINY(s_{RF}, e_{RF}+wec, f_{RF})} \\ (s_{RF}, e_{min}, f_{RF} \cdot 2^{e_{RF}+wec-e_{min}}) & \text{otherwise.} \end{cases}\end{aligned}$$

Because from $\overline{TINY(s_{RF}, e_{RF}-\alpha, f_{RF})}$ it follows that $\overline{TINY(s_{RF}, e_{RF}, f_{RF})}$, all overflow cases are already contained in the condition $\overline{TINY(s_{RF}, e_{RF}, f_{RF})}$. Because after exponent wrapping all representable results have values of normalized numbers according to corollary 2.10, these results can not be tiny and $wec = 0$ for the second line. Moreover, $\overline{TINY(s_{RF}, e_{RF}+\alpha, f_{RF})}$ follows from UNF_EN for underflows, so that with $\text{TINY} = TINY(s_{RF}, e_{RF}, f_{RF}) \Longleftrightarrow (e' < e_{min})$ the above equation for the bounded normalization shift can be reduced to:

$$(s_1, e_1+wec, f_1) \quad = \quad \begin{cases} (s_{RF}, e'+wec, f') = \eta(s_{RF}, e_{RF}+wec, f_{RF}) & \text{if } \overline{\text{TINY}} \text{ OR UNF\_EN} \\ (s_{RF}, e_{min}, f_{RF} \cdot 2^{e_{RF}-e_{min}}) & \text{otherwise.} \end{cases}$$

To simplify the implementation we postpone the wrapping exponent correction after the normalization shift computations and consider:

$$(s_1, e_1, f_1) \quad = \quad \begin{cases} (s_{RF}, e', f') = \eta(s_{RF}, e_{RF}, f_{RF}) & \text{if } \overline{\text{TINY}} \text{ OR UNF\_EN} \\ (s_{RF}, e_{min}, f_{RF} \cdot 2^{e_{RF}-e_{min}}) & \text{otherwise.} \end{cases}$$

Lemma 4.1 *With the bit-strings*

$$\begin{aligned} \text{SFTA}[5:0] &= \begin{cases} \text{LZII}[5:0] = \text{LZERO}((0, \text{F}_{RF}[-1:54], 0^7) & \textit{if } \overline{\text{TINY}} \textit{ OR UNF\_EN} \\ \text{BIN}_0^5(e_{RF} - e_{min} + 2) & \textit{otherwise.} \end{cases} \\ \text{F}'''[0:63] &= \text{CLS}((0, \text{F}_{RF}[-1:54], 0^7), sfta) \\ \text{SFTMASK}[0:63] &= \begin{cases} 1^{64} & \textit{if } e_{RF} - e_{min} + 2 \geq 0 \textit{ OR UNF\_EN} \\ 0^{64} & \textit{if } e_{RF} - e_{min} + 2 < -64 \\ \text{HDEC}(sfta)[63:0] & \textit{otherwise,} \end{cases} \end{aligned}$$

the factoring $(s_1, e_1, rep_{53}(f_1))$ *can be computed from* (s_{RF}, e_{RF}, f_{RF}) *by:*

$$\begin{aligned} s_1 &= s_{RF} \\ e_1 &= \begin{cases} e' = e_{RF} + 2 - lzii & \textit{if } \overline{\text{TINY}} \textit{ OR UNF\_EN} \\ e_{min} & \textit{otherwise.} \end{cases} \\ \text{REP}_{53}(f_1)[0:53] &= \text{F}'''[0:53] \textit{ AND } \text{SFTMASK}[0:53] \\ \text{REP}_{53}(f_1)[54] &= OR(\text{F}'''[0:53] \textit{ AND } \overline{\text{SFTMASK}[0:53]}, \text{F}'''[54:63]). \end{aligned}$$

Proof: We separate case (a) ($\overline{\text{TINY}}$ OR UNF_EN) and (b) ($\overline{\text{TINY}}$ NOR UNF_EN):

(a) First, for ($\overline{\text{TINY}}$ OR UNF_EN), we deal with the unbounded normalization shift

$$(s_1, e_1, f_1) = (s_{RF}, e', f') = \eta(s_{RF}, e_{RF}, f_{RF}) = \eta(s_{RF}, e_{RF}+2, f_{RF}/4).$$

If we only consider non-zero significands $f_0 = f_{RF}/4$, that have the binary representation $\text{F}_0[0:56] = (0, \text{F}_{RF}[-1:54])$, then these significands have values in the range $[2^{-55}, 2[$, so that lemma 2.3(ii) can be used: Thus, with $lzii = lzero(\text{F}_0[0:56]) = lzero((0, \text{F}_{RF}[-1:54])) \geq 1$, the unbounded normalization shift $\eta(s_{RF}, e_{RF}, f_{RF})$ can be computed by a left-shift of $(0, \text{F}_{RF}[-1:54])$ by $lzii = sfta$ positions (Note, that this computation is also valid for the case of zero significands, because they are not changed by the normalization shift regardless of the shift amount.)

$$\begin{aligned} \text{F}_1[0:63] &= (\text{F}_0[lzii:56], 0^{lzii+7}) = (\text{F}_{RF}[lzii-2:54], 0^{lzii+7}) \\ &= \text{CLS}((0, \text{F}_{RF}[-1:54], 0^7), sfta) \\ &= \text{F}'''[0:63] \end{aligned}$$

and the exponent adjustment (Note, that also this exponent adjustment is valid for zeros, because their factoring representation may contain an arbitrary exponent.)

$$e_1 = e' = e_{RF} + 2 - lzii.$$

With lemma 2.11, the 53-representative of f_1, $rep_{53}(f_1)$, is computed by

$$\text{REP}_{53}(f_1)[0:54] = (\text{F}_1[0:53], OR(\text{F}_1[54:63])).$$

Because from $\overline{\text{TINY}}$ it follows, that $(e_{RF} + 2 - e_{min} \geq 0)$, we get for case (a) $\text{SFTMASK}[0:53] = 1^{54}$, so that

$$\begin{aligned} \text{REP}_{53}(f_1)[0:53] &= \text{F}'''[0:53] \text{ AND } \text{SFTMASK}[0:53] \\ \text{REP}_{53}(f_1)[54] &= \text{OR}(\text{F}'''[0:53] \text{ AND } \overline{\text{SFTMASK}[0:53]}, \text{F}'''[54:63]), \end{aligned}$$

as required.

(b) For ($\overline{\text{TINY}}$ NOR UNF_EN), the resulting exponent after the unbounded normalization shift is e_{min}, and all we have to compute is the 53-representative $rep_{53}(f'')$ of the significand $f'' =< \text{F}'' >_{neg} = f_{RF} \cdot 2^{e_{RF}-e_{min}} = f_0 \cdot 2^{e_{RF}-e_{min}+2}$. The multiplication of $f_0 =< \text{F}_0[0:56] >_{neg}$ by $2^{e_{RF}-e_{min}+2}$ is a left-shift of $\text{F}_0[0:56]$ by $sft\_den =< \text{SFT\_DEN}[12:0] >_2 = e_{RF} - e_{min} + 2$ positions, where a positive shift amount $sft\_den > 0$ corresponds to an effective left-shift and a negative shift amount $sft\_den < 0$ corresponds to an effective right-shift by $|sft\_den|$ positions. In the computation of $rep_{53}(f'')$, we differ between 3 cases depending on the range of sft_den:

i. $sft\_den \geq 0$: Because we deal with denormalized numbers, we have $0 \leq sft\_den < lzii < 56$, so that sft_den can be represented with 6 bits $sft\_den =< \text{SFT\_DEN}[5:0] >= sfta$ and

$$\begin{aligned} \text{F}''[0:63] &= (\text{F}_0[sfta:56], 0^{sfta+7}) = (\text{F}_{RF}[sfta-2:54], 0^{sfta+7}) \\ &= \text{CLS}((0, \text{F}_{RF}[-1:54], 0^7), sfta) \\ &= \text{F}'''[0:63]. \end{aligned}$$

Thus, the 53-representative of $f'' = f''' = f_1$ is computed by (see lemma 2.11):

$$\text{REP}_{53}(f_1)[0:54] = (\text{F}'''[0:53], OR(\text{F}'''[54:63])).$$

Also in this case $\text{SFTMASK}[0:53] = 1^{54}$, so that we have

$$\begin{aligned} \text{REP}_{53}(f_1)[0:53] &= \text{F}'''[0:53] \text{ AND } \text{SFTMASK}[0:53] \\ \text{REP}_{53}(f_1)[54] &= \text{OR}(\text{F}'''[0:53] \text{ AND } \overline{\text{SFTMASK}[0:53]}, \text{F}'''[54:63]), \end{aligned}$$

as required for case (i).

ii. $0 > sft\_den \geq -53$: Because sft_den is negative, the computation of $f'' =< \text{F}''[0:56+|sft\_den|] >_{neg}$ requires a right-shift of $(0, \text{F}_{RF}[-1:54])$ by $|sft\_den|$ positions:

$$\text{F}''[0:56+|sft\_den|] = (0^{|sft\_den|+1}, \text{F}_{RF}[-1:54]).$$

Because sft_den is in the range $[-1:-64]$, the two's complement representation $\text{SFT\_DEN}[12:0]$ can be split into:

$$\begin{aligned} sft\_den &= < (1111111000000) >_2 + < \text{SFT\_DEN}[5:0] > \\ &= -64 + < \text{SFT\_DEN}[5:0] >, \end{aligned}$$

so that $sfta =< \text{SFT\_DEN}[5:0] >= 64 - |sft\_den| \geq 0$. Using a 64-bit cyclic left-shifter with the shift amount $sfta =< \text{SFT\_DEN}[5:0] >$ on $\text{F}_0[0:63] = (0, \text{F}_{RF}[-1:54], 0^7)$, we get

$$\begin{aligned} \text{F}'''[0:63] &= \text{CLS}((0, \text{F}_{RF}[-1:54], 0^7), sfta) \\ &= ((\text{F}_0[64 - |sft\_den| : 63], \text{F}_0[0 : 64 - |sft\_den| - 1])) \\ &= (\text{F}_0[64 - |sft\_den| : 63], 0, \text{F}_{RF}[-1 : 62 - |sft\_den|]) \end{aligned}$$

Thus, $\text{F}'''[0:53]$ could differ from $\text{F}''[0:53]$ only in the $|sft\_den|$ most significant bits, that have to be cleared. The mask

$$\text{SFTMASK}[0:63] = \text{HDEC}(sfta)[63:0] = (0^{|sft\_den|}, 1^{sfta})$$

has exactly zeros in these positions of the significand, so that

$$\text{REP}_{53}(f_1)[0:53] = \text{F}''[0:53] = \text{F}'''[0:53] \text{ AND } \text{SFTMASK}[0:53].$$

The sticky bit is computed from all the remaining bit positions, that are selected by the inverted mask $\overline{\text{SFTMASK}[0:53]}$ and significand positions $[54:63]$, so that

$$\text{REP}_{53}(f'')[54] = OR(\text{F}'''[0:53] \text{ AND } \overline{\text{SFTMASK}[0:53]}, \text{F}'''[54:63]),$$

as required for case (ii).

iii. $-53 > sft\_den$: In this case for the computation of F'', the significand $\text{F}_0[0:56] = (0, \text{F}_{RF}[-1:54])$ is right-shifted by more than 53 positions, so that $\text{F}''[0:53] = 0^{54}$ and no significand bit of $\text{F}_{RF}[-1:54]$ contributes to $\text{REP}_{53}(f_1)[0:53]$. Only the sticky bit in the representative $\text{REP}_{53}(f_1)[54] = OR(\text{F}_{RF}[-1:54])$ is influenced. If $-53 > sft\_den \geq -64$, we have $sfta = < \text{SFT\_DEN}[5:0] > = 64 - |sft\_den|$ like in case (ii), so that $sfta \leq 10$ and $\text{SFTMASK}[0:53] = \text{HDEC}(sfta)[63:10] = 0^{54}$. But also if $sft\_den = e_{RF} - e_{min} + 2 < -64$, we have $\text{SFTMASK}[0:53] = 0^{54}$ by definition. Thus,

$$\begin{aligned} \text{REP}_{53}(f_1)[0:53] &= \text{F}''[0:53] = 0^{54} = \text{F}'''[0:53] \text{ AND } \text{SFTMASK}[0:53] \\ \text{REP}_{53}(f'')[54] &= OR(\text{F}_{RF}[-1:54]) \\ &= OR(\text{F}'''[0:63]) \\ &= OR((\text{F}'''[0:53] \text{ AND } \overline{\text{SFTMASK}[0:53]}) \text{ OR } \text{F}'''[54:63]), \end{aligned}$$

as required for case (iii). This completes the proof of the lemma.

□

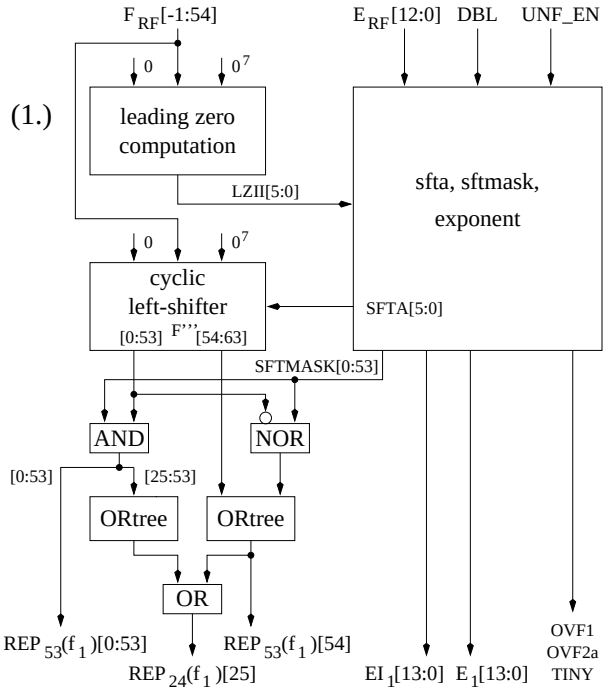

Figure 4.2: Normalization shift implementation in the General rounding unit

The implementation of the normalization shift and the 53-representative computation corresponding to lemma 4.1 is depicted in figure 4.2. Additionaly, this figure includes the computation of the sticky bit of the 24-representative $\text{REP}_{24}(f_1)[25]$ from $\text{REP}_{53}(f_1)[25:54]$ according to lemma 2.12. The implementation of the the 'sfta, sftmask and exponent' circuit has to be further specified. This circuit is responsible for the computation of the shift amount $\text{SFTA}[5:0]$, the mask $\text{SFTMASK}[0:53]$, the exponent e_1 and the incremented exponent $ei_1 = e_1 + 1$. We consider the biased exponents $e_{1b} =< \text{E}_1[13:0] >_2 = e_1 + bias_n$ and $ei_{1b} =< \text{EI}_1[13:0] >_2 = ei_1 + bias_n$, so that

$$e_1 =< \text{E}_1[13:0] >_2 - bias_n \quad \text{and} \quad ei_1 =< \text{EI}_1[13:0] >_2 - bias_n.$$

Moreover, the bits OVF1, OVF2a and TINY are computed, that indicate the conditions:

$$\begin{aligned} \text{OVF1} &\iff (e_1 > e_{max}) \\ \text{OVF2}a &\iff (e_1 = e_{max}) \\ \text{TINY} &\iff TINY(s_{RF}, e_{RF}, f_{RF}) \iff (e' < e_{min}). \end{aligned}$$

The following lemma specifies how all the outputs of the 'sfta, sftmask and exponent' circuit can be computed from the inputs $\text{E}_{RF}[12:0]$, $\text{LZII}[5:0]$, DBL and UNF_EN.

Lemma 4.2 *After the computation of the intermediate values*

$$he = <\text{HE}[13:6]>_2 = <(\text{E}_{RF}[12], \text{E}_{RF}[12:6], 0^6)>_2 + <(0^4, \text{DBL}^3, 1, 0^6)> \quad (4.4)$$

$$hei = <\text{HEI}[13:6]>_2 = he + 2^6 \quad (4.5)$$

$$hf = <\text{HF}[6:0]> = <\text{E}_{RF}[5:0]> + <\overline{\text{LZII}[5:0]}> + 1 \quad (4.6)$$

$$\text{MASK1} \iff (\overline{\text{HEI}[13]}\ \mathit{OR}\ \text{UNF\_EN}) \quad (4.7)$$

$$\text{MASK0} \iff (\overline{\text{HEI}[13]}\ \mathit{NOR}(\mathit{ANDtree}(\text{HEI}[12:6]))) \quad (4.8)$$

$$hb = <\text{HB}[13:0]>_2 = \begin{cases} <\text{HEI}[13:6], \text{HF}[5:0]>_2 & \mathit{if}\ \text{HF}[6] \\ <\text{HE}[13:6], \text{HF}[5:0]>_2 & \mathit{otherwise}, \end{cases} \quad (4.9)$$

the outputs of the 'sfta, sftmask and exponent' circuit can be computed by

$$\text{TINY} \iff \begin{cases} \text{HEI}[13] & \mathit{if}\ \text{HF}[6] \\ \text{HE}[13] & \mathit{otherwise} \end{cases} \quad (4.10)$$

$$\text{SFTA}[5:0] = \begin{cases} \text{LZII}[5:0] & \mathit{if}\ \overline{\text{TINY}}\ \mathit{OR}\ \text{UNF\_EN} \\ \text{E}_{RF}[5:0] & \mathit{otherwise.} \end{cases} \quad (4.11)$$

$$\text{SFTMASK}[0:53] = ((\text{HDEC}(sfta)[63:10]\ \mathit{NOR}\ \text{MASK1})\ \mathit{NOR}\ \text{MASK0}) \quad (4.12)$$

$$e_{1b} = <\text{E}_1[13:0]>_2 = <\text{HC}[13:0]>_2 + 1 \quad (4.13)$$

$$ei_{1b} = <\text{EI}_1[13:0]>_2 = <\text{HC}[13:0]>_2 + 2 \quad (4.14)$$

$$\text{OVF1} \iff \overline{\text{EI}_1[13]}\ \mathit{AND}\ (\mathit{ORtree}(\text{EI}_1[12:11], (\text{EI}_1[10:8]\ \mathit{AND}\ \overline{\text{DBL}}))) \quad (4.15)$$

$$\text{OVF2}a \iff \mathit{ANDtree}(\overline{\text{EI}_1[13:11]}, (\text{EI}_1[10:8] \oplus \text{DBL}), \text{EI}_1[7:0]) \quad (4.16)$$

using the definition of

$$\text{HC}[13:0] = \text{HB}[13:0]\ \mathit{AND}\ (\text{TINY}\ \mathit{NAND}\ \overline{\text{UNF\_EN}}).$$

Proof: First, we show some properties of the intermediate values, so that we can then prove the correctness of the output computations using these properties. Because $-e_{min} + 1 = bias = <0^4, \text{DBL}^3, 1^7>$ for single and double precision, we have

$$\begin{aligned} sft\_den &= <\text{SFT\_DEN}[13:0]>_2 = e_{RF} - e_{min} + 2 \\ &= <(\text{E}_{RF}[12], \text{E}_{RF}[12:0])>_2 + <(0^4, \text{DBL}^3, 1^7)> + 1 \\ &= <(\text{E}_{RF}[12], \text{E}_{RF}[12:0])>_2 + <(0^4, \text{DBL}^3, 1, 0^6)> + 2^6 \\ &= <\text{HEI}[13:6], \text{E}_{RF}[5:0]>_2 \end{aligned}$$

Based on this one can show, that the bits MASK0 and MASK1 implement the conditions

$$\begin{aligned} \text{MASK1} &\iff (\overline{\text{HEI}[13]}\ \text{OR}\ \text{UNF\_EN}) \iff ((sft\_den \geq 0)\ \text{OR}\ \text{UNF\_EN}) \\ \text{MASK0} &\iff (\overline{\text{HEI}[13]}\ \text{NOR}\ (\text{ANDtree}(\text{HEI}[12:6]))) \\ &\iff (\text{HEI}[13]\ \text{AND}\ (\text{NOT}(\text{ANDtree}(\text{HEI}[12:6])))) \\ &\iff (sft\_den < -64). \end{aligned}$$

Exactly these conditions are required to select the proper case in the computation of SFTMASK[0 : 53]. The intermediate value hb is defined by

$$hb = \begin{cases} <\text{HEI}[13:6], \text{HF}[5:0]>_2 & \text{if HF}[6] \\ <\text{HE}[13:6], \text{HF}[5:0]>_2 & \text{otherwise.} \end{cases}$$

$$
\begin{aligned}
&= < \text{HE}[13:6], 0^6 >_2 + < \text{HF}[6:0] > \\
&= < \text{HEI}[13:6], 0^6 >_2 + < 1^8, 0^6 >_2 \\
&\quad + < \text{E}_{RF}[5:0] > + < \overline{\text{LZII}[5:0]} > +1 \\
&= < \text{HEI}[13:6], \text{E}_{RF}[5:0] >_2 + < 1^8, \overline{\text{LZII}[5:0]} >_2 +1 \\
&= < \text{SFT\_DEN}[13:0] >_2 + < 1^8, \overline{\text{LZII}[5:0]} >_2 +1 \\
&= sft\_den - lzii \\
&= e_{RF} - e_{min} + 2 - lzii.
\end{aligned}
$$

Thus, starting from the definition of TINY, we get

$$
\begin{aligned}
\text{TINY} &\iff (e'(= e_{RF} + 2 - lzii) < e_{min}) \\
&\iff (e_{RF} - e_{min} + 2 - lzii < 0) \\
&\iff (hb < 0) \\
&\iff \begin{cases} \text{HEI}[13] & \text{if HF}[6] \\ \text{HE}[13] & \text{otherwise.} \end{cases}
\end{aligned}
$$

Because $bin_0^5(e_{RF} - e_{min} + 2) = bin_0^5(sft\_den) = \text{E}_{RF}[5:0]$, the computation formula of SFTA$[5:0]$ follows directly from the definition of SFTA$[5:0]$. Using the conditions MASK1 and MASK0, the definition of SFTMASK$[0:53]$ becomes

$$
\text{SFTMASK}[0:53] = \begin{cases} 1^{54} & \text{if MASK1} \\ 0^{54} & \text{if MASK0} \\ \text{HDEC}(sfta)[63:10] & \text{otherwise.} \end{cases} \tag{4.17}
$$

One can easily check that this is equivalent to

$$
\text{SFTMASK}[0:53] = ((\text{HDEC}(sfta)[63:10] \text{ NOR MASK1}) \text{ NOR MASK0}).
$$

The bit string HC$[13:0]$ can be written as:

$$
\begin{aligned}
\text{HC}[13:0] &= \text{HB}[13:0] \text{ AND } (\text{TINY NAND } \overline{\text{UNF\_EN}}) \\
&= \begin{cases} \text{HB}[13:0] & \text{if } \overline{\text{TINY}} \text{ OR UNF\_EN} \\ 0^{14} & \text{otherwise.} \end{cases}
\end{aligned}
$$

Using $< 0^{14} >_2 = e_{min} - 1 + bias_n$ and $hb = e_{RF} - e_{min} + 2 - lzii = e_{RF} + bias_n + 1 - lzii$, we get

$$
\begin{aligned}
hc &= < \text{HC}[13:0] >_2 = \begin{cases} e_{RF} + 1 - lzii + bias_n & \text{if } \overline{\text{TINY}} \text{ OR UNF\_EN} \\ e_{min} - 1 + bias_n & \text{otherwise} \end{cases} \\
&= e_1 - 1 + bias_n
\end{aligned}
$$

so that corresponding to the definition of e_{1b}, we get the computation formula

$$
e_{1b} = hc + 1 = < \text{HC}[13:0] >_2 +1
$$

The computation formula of ei_{1b} follows directly from the definition of the incremented biased exponent $ei_{1b} = e_{1b} + 1$. Because $e_{1b} = e_1 + bias_n$ and $e_{max} + bias_n = 2^n - 2 = < 0^3, \text{DBL}^3, 1^7, 0 >_2$ for single and double precision, the condition $(e_1 \geq e_{max})$ can be written as

$$
\begin{aligned}
(e_1 \geq e_{max}) &\iff (e_{1b} \geq 2^n - 2) \\
&\iff (ei_{1b} \geq < 0^3, \text{DBL}^3, 1^8 >_2).
\end{aligned}
$$

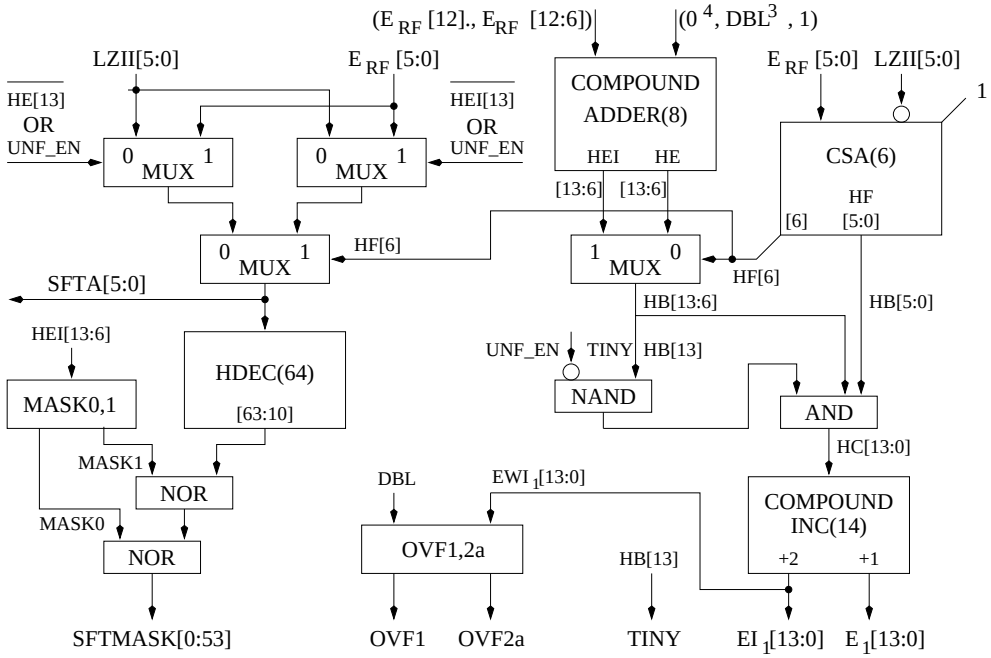

Figure 4.3: 'sfta, sftmask and exponent' circuit in the general rounding unit.

The condition OVF1 is the 'greater than' case of the above condition and the condition OVF2a is the equality case, so that

$$\begin{aligned}
\text{OVF1} &\iff (< \text{EI}_1[13:0] >_2 \; > \; < 0^3, \text{DBL}^3, 1^8 >_2) \\
&\iff \overline{\text{EI}_1[13]} \text{ AND } (\text{ORtree}(\text{EI}_1[12{:}11], (\text{EI}_1[10{:}8] \text{ AND } \overline{\text{DBL}}))) \\
\text{OVF2}a &\iff (\text{EI}_1[13:0] \; = \; (0^3, \text{DBL}^3, 1^8)) \\
&\iff \text{ANDtree}(\overline{\text{EI}_1[13:11]}, (\text{EI}_1[10:8] \overline{\oplus} \text{DBL}), \text{EI}_1[7:0])
\end{aligned}$$

as required by the lemma. □

In this way the 'sfta, sftmask and exponent' circuit can be implemented like depicted in figure 4.3. In this figure the mask0,1 and the ovf1,2a signal are implemented according to equations 4.8, 4.7 and 4.15, 4.16 respectively. This completes the description of the implementation of the normalization and representative computations.

Significand rounding (2.) In this paragraph, we consider the significand rounding:

$$(s_2, e_2 + wec, f_2) = sig\_rnd_{mode \star s_1}(s_1, e_1 + wec, rep_{53}(f_1)).$$

Because only the significand is affected by this operation, we have $s_2 = s_1$ and $e_2 + wec = e_1 + wec = e_{1b} - bias_n + wec$. Therefore, we only focus on the significand in the following. Depending on the bit DBL, we compute the significand rounding at significand position $(p-1)$ on the p-representative $rep_p(f_1)$ in single ($p = 24$) or double ($p = 53$) precision. From the representative computation we get the 53-representative $\text{REP}_{53}(f_1)[-1:54]$ and the bit $\text{REP}_{24}(f_1)[25]$, so that we also have the 24-representative by

$$\text{REP}_{24}(f_1)[-1:25] = (\text{REP}_{53}(f_1)[-1:24], \text{REP}_{24}(f_1)[25]).$$

The significand rounding on p-representatives is already described in section 2.5.1. Following this description the rounding of the p-representative $rep_p(f_1)$ results either in the *truncated significand* $ftr = <\text{FTR}[-1{:}52]>_{neg} =< \text{REP}_{53}(f_1)[-1:(p-1)] >_{neg}$ or the *incremented significand* $finc = <\text{FTRI}[-1{:}52]>_{neg} = ftr + 2^{-p+1}$ (see definition 2.14). Obviously, for both single and double precision these significands can be computed by

$$\begin{aligned} \text{FTR}[-1:52] &= (\text{REP}_{53}(f_1)[-1:23], \text{REP}_{53}(f_1)[24:52] \text{ AND DBL}) \\ < \text{FTRI}[-1:52] >_{neg} &= < (\text{REP}_{53}(f_1)[-1:23], \text{REP}_{53}(f_1)[24:52] \text{ OR } \overline{\text{DBL}}) >_{neg} + 2^{-52}. \end{aligned}$$

Moreover, the three least significant bits of the p-representative have to be selected:

$$\begin{aligned} (\text{L, R, STICKY}) &= \text{REP}_p(f_1)[(p-1){:}(p+1)] \\ &= \begin{cases} \text{REP}_{53}(f_1)[52{:}54] & \text{if DBL} \\ (\text{REP}_{53}(f_1)[23{:}24], \text{REP}_{24}(f_1)[25]) & \text{otherwise} \end{cases} \end{aligned}$$

and the IEEE rounding mode $mode \in \{RZ, RNE, RI, RMI\}$ encoded by RND_MODE[1:0] has to be reduced for the use on the positive significand to: $(mode \star s_1) \in \{RZ, RNE, RI\}$ encoded by SR_MODE[1:0] (according to equations 2.6-2.7 and table 2.3):

$$\text{SR\_MODE}[1] = \text{RND\_MODE}[1] \wedge (\text{RND\_MODE}[0] \overline{\otimes} s) \tag{4.18}$$

$$\text{SR\_MODE}[0] = \overline{\text{RND\_MODE}[1]} \wedge \text{RND\_MODE}[0], \tag{4.19}$$

to implement the rounding increment decision (equation 2.54):

$$\text{RINC} = \text{SR\_MODE}[1] \wedge (\text{R} \vee \text{STICKY}) \vee \text{SR\_MODE}[0] \wedge \text{R} \wedge (\text{L} \vee \text{STICKY}). \tag{4.20}$$

Based on RINC, the significand can be rounded according to equation 2.55:

$$\text{F}_2[-1:52] = \begin{cases} \text{FTRI}[-1:52] & \text{if RINC} \\ \text{FTR}[-1:52] & \text{otherwise.} \end{cases}$$

This results in an implementation of significand rounding like depicted in region (2.) of figure 4.4. In this region, the rounding decision circuit contains the implementation of equations 4.18, 4.19 and 4.20. Moreover, a conditional sum incrementer implementation is used for the implementation.

Because significand rounding could change the value of the factoring, in the rounding decision circuit we also compute the condition INX1, that recognizes the significand rounding inexactness condition according to lemma 2.17:

$$\text{INX1} \iff (f_2 \neq f_1) \iff (f_2 \neq rep_p(f_1)) \iff (\text{R OR STICKY}). \tag{4.21}$$

Post-normalization (3.) The post-normalization shift is the implementation of

$$\begin{aligned} (s_3, e_3 + wec, f_3) &= post\_norm(s_2, e_2 + wec, f_2) \\ &= post\_norm(s_2, e_{1b} - bias_n + wec, f_2) \\ &= \begin{cases} (s_2, ei_{1b} - bias_n + wec, 1) & \text{if } f_2 = 2 \\ (s_2, e_{1b} - bias_n + wec, f_2) & \text{otherwise.} \end{cases} \end{aligned}$$

Because f_2 can not become larger than 2, the condition $(f_2 = 2)$ is recognized by bit $\text{F}_2[-1]$, so that the post-normalization shift of the significand can be implemented by a simple OR-gate

$$\text{F}_3[0:52] = (\text{F}_2[-1] \text{ OR } \text{F}_2[0], \text{F}_2[1:52]).$$

We do not compute e_3, but the biased exponent $e_{3b} =< \text{E}_3[13:0] >_2 = e_3 + bias_n$, that can be selected from the previous computed e_{1b} and ei_{1b}:

$$e_{3b} = \begin{cases} ei_{1b} & \text{if } f_2[-1] \\ e_{1b} & \text{otherwise.} \end{cases} \tag{4.22}$$

The case $(f_2 = 2) \Longleftrightarrow f_2[-1]$ is called *significand overflow* and signaled by the bit SIGOVF. This results in an implementation of the post-normalization shift like depicted in figure 4.4, where region (3.a) includes the post-normalization of the significand and region (3.b) includes the exponent selection according to equation 4.22.

Exponent rounding (4.) and packing In this paragraph we describe first, how the exception conditions OVF, INX and UNF can be recognized and how the wrapping exponent correction is added to the exponent. We then describe the exponent rounding followed by the packing conversion of the rounded result to the packed representation, that is required as output of the general rounding unit.

Lemma 4.3 *With the bit*

$$\text{FINOP} = (\overline{\text{ZERO}}\ AND\ \overline{\text{INF}}\ AND\ \overline{\text{QNAN}}\ AND\ \overline{\text{SNAN}}), \tag{4.23}$$

(a) the overflow exception condition OVF*, (b) the inexact exception condition* INX *and (c) the underflow exception condition* UNF *can be computed by*

$$\text{OVF} \Longleftrightarrow (\text{OVF1}\ OR\ (\text{OVF2}a\ AND\ \text{SIGOVF}))\ AND\ \text{FINOP}. \tag{4.24}$$

$$\text{INX} \Longleftrightarrow (\text{INX1}\ OR\ \text{OVF}) \tag{4.25}$$

$$\text{UNF} \Longleftrightarrow (\text{TINY}\ AND\ (\text{INX}\ OR\ \text{UNF\_EN})). \tag{4.26}$$

Proof: (a) An overflow occurs, iff (i) the magnitude of the unbounded rounded result is larger than x_{max} and (ii) the rounding input is the representation of a non-zero finite number. A representative number representation is non-zero and finite, iff it does not represent a special value and ZERO = INF = QNAN = SNAN = 0, so that the condition FINOP = ($\overline{\text{ZERO}}$ AND $\overline{\text{INF}}$ AND $\overline{\text{QNAN}}$ AND $\overline{\text{SNAN}}$) is equivalent to part(ii) of the overflow condition. Part (i) of the overflow condition can be written as

$$|val(s_3, e_3, f_3)| \quad > \quad x_{max} = (2 - 2^{-p+1}) \cdot 2^{e_{max}}.$$

We first assume that no tininess occurs. Thus, the significand $f_3 =< \text{F}_3[0:p-1] >_{neg}$ is normalized and we have $1 \leq f_3 \leq (2 - 2^{-p+1})$, so that

$$\text{OVF} \Longleftrightarrow (|val(s_3, e_3, f_3)| > x_{max})\ \text{AND FINOP} \tag{4.27}$$

$$\Longleftrightarrow (e_3 > e_{max})\ \text{AND FINOP}. \tag{4.28}$$

For $e_{1b} = e_1 + bias_n$ and $ei_{1b} = ei_1 + bias_n$ we can extract the following formula for e_3 from equation 4.22 :

$$e_3 = e_{3b} - bias_n = \begin{cases} ei_1 & \text{if } (f_2[-1] = 1) \\ e_1 & \text{otherwise.} \end{cases}$$

Because $ei_1 = e_1 + 1$, from $(e_1 > x_{max})$ it follows that also $(ei_1 > x_{max})$. Therefore,

$$(e_3 > x_{max}) \Longleftrightarrow (e_1 > x_{max})\ \text{OR}\ ((ei_1 > x_{max})\ \text{AND}\ (f_2[-1] = 1)). \tag{4.29}$$

The substitution of the definitions OVF1 $\Longleftrightarrow (e_1 > x_{max})$, OVF2$a$ $\Longleftrightarrow (ei_1 > x_{max})$ and SIGOVF $\Longleftrightarrow (f_2[-1] = 1)$ in equation 4.29 in combination with equation 4.28 then yields part (a) of the lemma for non-tiny values. If tininess occurs, then $|val(s_3, e_3, f_3)| < 2^{e_{min}}$ and $e_3 = e_{min}$, so that OVF $= 0$, OVF1 $= 0$ and OVF2a $= 0$ by the overflow definitions. Therefore, the overflow formula follows also for tiny numbers and the proof of part (a) of the lemma is completed.
(b) An inexact exception occurs, iff the rounded result differs from the exact result. This can be caused by two parts of the rounding procedure: the significand and the exponent rounding. (The normalization shifts do not change the value of the factorings.) The inexactness caused by significand rounding is already recognized by the condition INX1. The exponent rounding (including exponent wrapping) changes the value of the operand, iff the unbounded rounded result would be larger than x_{max}. This is exactly the OVF condition, so that INX $\Longleftrightarrow$ (INX1 OR OVF), as required for part (b) of the lemma.
(c) For our choice of the loss-of-accuracy definition, the underflow exception is defined by

$$\text{UNF} \iff \begin{cases} \text{TINY} & \text{if } (\text{UNF\_EN} = 1) \\ \text{TINY AND INX} & \text{otherwise.} \end{cases}$$

Obviously, this is equivalent to part (c) of the lemma. □

Now, we consider the exponent wrapping. Because for UNF_EN $= 1$, we have UNF $\Longleftrightarrow$ TINY, the *trapped underflow condition*:

$$\text{TUNF} \iff (\text{TINY AND UNF\_EN}) \tag{4.30}$$

signals the case that a trapped underflow occurs. Moreover, we define the *trapped overflow condition* TOVF, that indicates the occurance of a trapped overflow:

$$\text{TOVF} \iff (\text{OVF AND OVF\_EN}). \tag{4.31}$$

Based on these definitions, the exponent wrapping on e_{3b} can be described by:

$$ew_3 = e_{3b} + wec = \begin{cases} < \text{E}_3[13:0] >_2 - \alpha & \text{if TOVF} \\ < \text{E}_3[13:0] >_2 + \alpha & \text{if TUNF} \\ < \text{E}_3[13:0] >_2 & \text{otherwise.} \end{cases} \tag{4.32}$$

Because the signal TUNF is valid ealier than TOVF, we define a *predicted wrapping exponent correction pwec* based on TUNF and we compute it by a selection using that for single and double precision $+\alpha =< +\text{ALPHA}[13:6] >_2 = 3 \cdot 2^{n-2} =< (0^3, \text{DBL}^2, 0, \overline{\text{DBL}}^2, 0^6) >_2$ and $-\alpha =< -\text{ALPHA}[13:6] >_2 = -3 \cdot 2^{n-2} =< (1^3, \overline{\text{DBL}}, 1, \overline{\text{DBL}}, 0, \overline{\text{DBL}}, 0^6) >_2$:

$$pwec =< \text{PWEC}[13:6] >_2 \quad = \quad \begin{cases} +\alpha =< +\text{ALPHA}[13:6] >_2 & \text{if TUNF} \\ -\alpha =< -\text{ALPHA}[13:6] >_2 & \text{otherwise.} \end{cases}$$

Lemma 4.4 *After the computation of a predicted wrapped exponent pew_3 by the addition of the predicted wrapped exponent correction,*

$$pew_3 =< \text{PEW}_3[13:0] >_2 = e_{3b} + pwec \tag{4.33}$$

and the definition of the wrapping exponent condition EWRAP*:*

$$\text{EWRAP} = (\text{TUNF } \mathit{OR} \text{ TOVF}) \tag{4.34}$$

the exponent wrapping on e_{3b} can be computed by the selection:

$$ew_3 = e_{3b} + wec = \begin{cases} pew_3 & \textit{if } \text{EWRAP} \\ e_{3b} & \textit{otherwise.} \end{cases} \tag{4.35}$$

Proof: For TOVF = 1, the predicted wrapped exponent correction is $pwec = -\alpha$, so that $ew_3 = e_{3b} - \alpha$ in equation 4.32 and in equation 4.35. In the same way, the identity of these two equations can be shown for the remaining two cases: TUNF = 1 and EWRAP = 0. □

The exponent rounding is influenced by the reduced rounding mode ($mode \star s$) that is encoded by SR_MODE[1 : 0] according to table 2.3. We already get SR_MODE[1 : 0] from the significand rounding circuit, so that we can compute the condition

$$\begin{aligned} \text{RNDUP} &\iff \text{SR\_MODE}[1] \text{ OR } \text{SR\_MODE}[0] && (4.36)\\ &\iff (mode \star s \in \{RNE, RI\}) && (4.37)\\ \overline{\text{RNDUP}} &\iff (mode \star s = RZ) && (4.38) \end{aligned}$$

Moreover, we define the *untrapped overflow condition* UOVF, that indicates the occurance of an untrapped overflow:

$$\text{UOVF} \iff (\text{OVF AND } \overline{\text{OVF\_EN}}). \tag{4.39}$$

Because $(|val(s_3, e_3 + wec, f_3)| > x_{max})$, iff an untrapped overflow occurs (UOVF = 1), the exponent rounding can be described by:

$$\begin{aligned} (s_{PF}, e_{PF}, f_{PF}) &= exp\_rnd_{mode \star s_3}(s_3, e_3 + wec, f_3) && (4.40)\\ &= \begin{cases} (s_3, e_\infty, f_\infty) & \text{if UOVF AND RNDUP} \\ (s_3, e_{max}, f_{max}) & \text{if UOVF AND } \overline{\text{RNDUP}} \\ (s_3, ew_3 - bias_n, f_3) & \text{otherwise.} \end{cases} && (4.41) \end{aligned}$$

This selection of the exponent and the significand for the exponent rounding is computed in combination with the conversion step from the IEEE factoring (s_{PF}, e_{PF}, f_{PF}) to the corresponding packed representation, that consists of $(\text{S}_{PF}, \text{E}_{PF}[n{-}1{:}0], \text{F}_{PF}[1{:}p{-}1], 0^{64-n-p})$.

Lemma 4.5 *With the definition of the conditions*

$$\begin{aligned} \text{RINF} &= (\text{UOVF } \mathit{AND} \text{ RNDUP}) && (4.42)\\ \text{RMAX} &= (\text{UOVF } \mathit{AND} \overline{\text{RNDUP}}) && (4.43) \end{aligned}$$

we can compute the bits of the packed representation of the rounded result by

$$\begin{aligned} \text{S}_{PF} &= \text{S}_3 && (4.44)\\ \text{E}_{PF}[10{:}1] &= ((\text{EW}_3[10{:}1] \mathit{\ NOR\ } \text{UOVF}) \mathit{\ NOR\ } \overline{\text{F}_3[0]}) && (4.45)\\ \text{E}_{PF}[0] &= ((\text{EW}_3[0] \mathit{\ NOR\ } \text{RINF}) \mathit{\ NOR\ } (\text{RMAX } \mathit{OR}\ \overline{\text{F}_3[0]})) && (4.46)\\ \text{F}_{PF}[1{:}52] &= ((\text{F}_3[0{:}52] \mathit{\ NOR\ } \text{RMAX}) \mathit{\ NOR\ } \text{RINF}) && (4.47) \end{aligned}$$

Proof: The conversion from the unpacked representation to the packed representation can be computed according to section 2.6.2.2. Thus, the sign and the significand are unchanged, only the hidden bit $\text{F}_3[0]$ is removed from the representation of the significand. With the use of $\text{F}_{max}[1{:}p{-}1] = 1^{p-1}$ and $\text{F}_\infty[1{:}p{-}1] = 0^{p-1}$, we get

$$\begin{aligned} \text{S}_{PF} &= \text{S}_{UF} && (4.48)\\ \text{F}_{PF}[1{:}p{-}1] &= \text{F}_{UF}[1{:}p{-}1] && (4.49)\\ &= \begin{cases} \text{F}_\infty[1{:}p{-}1] = 0^{p-1} & \text{if RINF} \\ \text{F}_{max}[1{:}p{-}1] = 1^{p-1} & \text{if RMAX} \\ \text{F}_3[1{:}p{-}1] & \text{otherwise.} \end{cases} && (4.50)\\ &= ((\text{F}_3[1{:}p{-}1] \text{ NOR RMAX}) \text{ NOR RINF}) && (4.51) \end{aligned}$$

Because in the packing for single precision only the significand bits $\mathrm{F}_{PF}[1:23]$ are regarded and the bits $\mathrm{F}_{PF}[24:52]$ are ignored, we can compute the packed significand representation for both precisions by equation 4.51 with $p = 53$ as stated by the lemma.

For the conversion of the exponent, we have to consider the n-bit biased representation of e_{UF} and to integrate the redundant exponent representation for e_{min}, where $\mathrm{E}_{min}[n-1:0] = 0^n$ for denormalized numbers and zeros according to equation 2.84. Because after the conditional exponent wrapping and the exponent rounding, the exponent is representable in this n-bit packed format for all cases, it is sufficient for both single and double precision to regard only the exponent bits at positions $[10:0]$. For single precision the exponent bits at positions $[10:8]$ are ignored later.

The biased representations of e_{max} and e_∞ are given by $(1^{n-1}, 0)$ and 1^n respectively. For $(\mathrm{RINF} = 1) \Longrightarrow (\mathrm{F}_3[0] = 1)$ and $(\mathrm{RMAX} = 1) \Longrightarrow (\mathrm{F}_3[0] = 1)$, we can compute the packed exponent representation by

$$\mathrm{E}_{PF}[10:0] = \begin{cases} bin_0^{10}(e_{PF} + bias_n) & \text{if } \mathrm{F}_3[0] \\ 0^{11} & \text{otherwise.} \end{cases} \tag{4.52}$$

$$= \begin{cases} 1^n & \text{if RINF} \\ (1^{n-1}, 0) & \text{if RMAX} \\ 0^{11} & \text{if } \overline{\mathrm{F}_3[0]} \\ \mathrm{EW}_3[10:0] & \text{otherwise.} \end{cases} \tag{4.53}$$

If we separate equation 4.53 into an equation regarding the exponent positions $[10:1]$ and an equation regarding exponent position $[0]$, we can simplify these equations to

$$\begin{aligned} \mathrm{E}_{PF}[10:1] &= ((\mathrm{EW}_3[10:1] \text{ NOR } \mathrm{UOVF}) \text{ NOR } \overline{\mathrm{F}_3[0]}) \\ \mathrm{E}_{PF}[0] &= ((\mathrm{EW}_3[0] \text{ NOR } \mathrm{RINF}) \text{ NOR } (\mathrm{RMAX} \text{ OR } \overline{\mathrm{F}_3[0]})) \end{aligned}$$

This completes the proof of the lemma. □

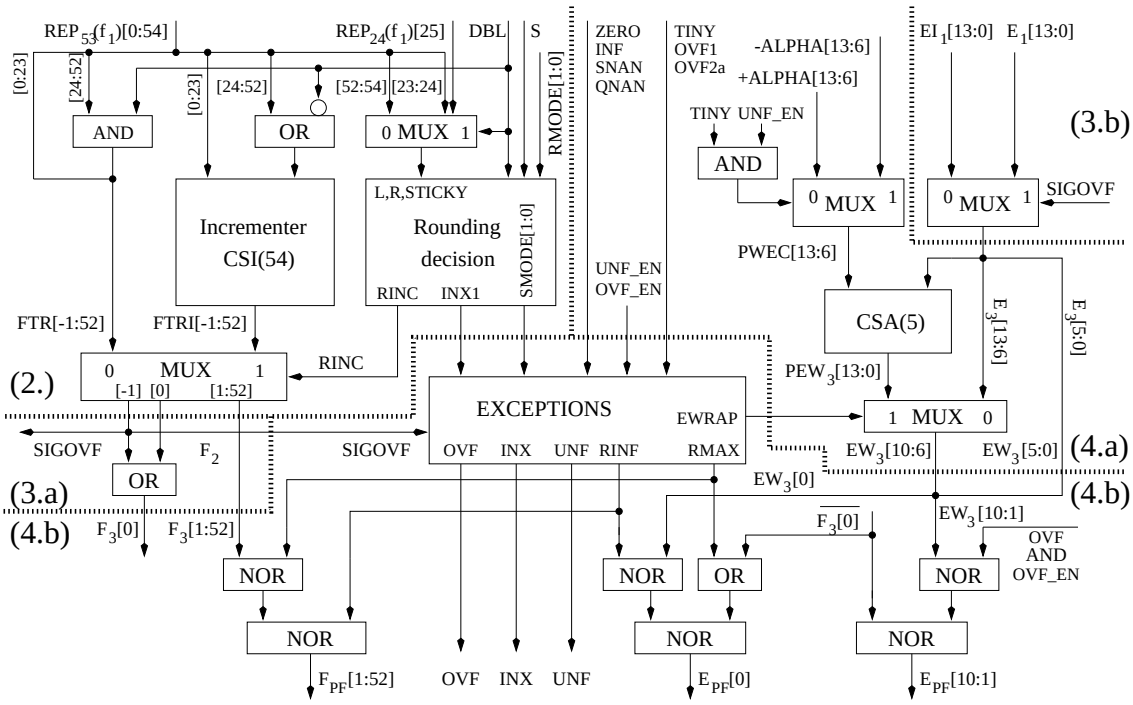

Figure 4.4: Significand rounding (2.), post-normalization (3.a, 3.b), exponent wrapping (4.a) and exponent rounding (4.b) implementation.

Figure 4.4 depicts the implementation of the exponent wrapping (region 4.a), the exponent rounding and the computation of the exceptions (region 4.b) corresponding to the descriptions of this paragraph. In region (4.a) of figure 4.4 a 5-bit conditional sum adder is used for the implementation.

Because only the bit positions $[10:0]$ of the unpacked exponent representation are required, it suffices to compute all additions for the exponent computation modulo 2^{10}. This is already considered in figures 4.4 and 4.5, where we only consider $\mathrm{EI}_1[10:0]$, $\mathrm{E}_1[10:0]$, $\mathrm{E}_3[10:0]$, $\mathrm{PEW}_3[10:0]$, $\mathrm{PWEC}[10:0]$, $+\alpha[10:6]$ and $-\alpha[10:6]$ instead of $\mathrm{EI}_1[13:0]$, $\mathrm{E}_1[13:0]$, $\mathrm{E}_3[13:0]$, $\mathrm{PEW}_3[13:0]$, $\mathrm{PWEC}[13:0]$, $+\alpha[13:6]$ and $-\alpha[13:6]$.

The *exceptions* circuit in region (4.b) of figure 4.4 implements the exception conditions OVF, INX and UNF according to equations 4.23-4.26. Additionaly, in this circuit the bits RINF, RMAX and EWRAP are computed. In the previous descriptions we used many intermediate conditions from that these bits could be derived. Based on the input bits of the *exceptions* circuit the computations can be summarized by the following three equations:

$$\text{EWRAP} = (\text{OVF AND OVF\_EN}) \text{ OR } (\text{TINY AND UNF\_EN}) \tag{4.54}$$

$$\text{RMAX} = (\text{OVF AND } \overline{\text{OVF\_EN}} \text{ AND } (\text{SR\_MODE}[1] \text{ NOR SR\_MODE}[0])) \tag{4.55}$$

$$\text{RINF} = (\text{OVF AND } \overline{\text{OVF\_EN}} \text{ AND } (\text{SR\_MODE}[1] \text{ OR SR\_MODE}[0])) \tag{4.56}$$

Finally, we pack the result representation into the packed format $BUS_{PF}[63:0]$ for single and double precision results and get according to equations 2.75-2.80 the following selection

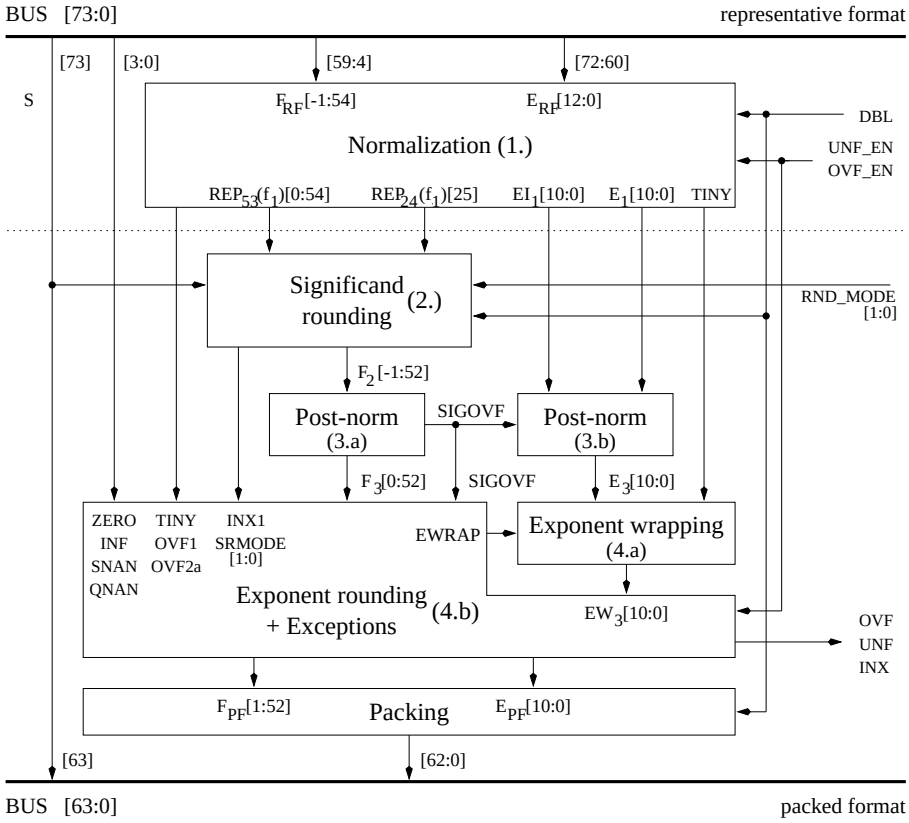

Figure 4.5: Structure of the general rounding unit.

that can be computed by a row of muxes:

$$BUS_{PF}[63{:}0] \quad = \quad \begin{cases} (\mathrm{S}_{PF}, \mathrm{E}_{PF}[10:0], f_{PF}[1:52]) & \text{if DBL} \\ (\mathrm{S}_{PF}, \mathrm{E}_{PF}[7:0], f_{PF}[1:23], 0^{32}) & \text{otherwise.} \end{cases} \tag{4.57}$$

This completes the description of the general rounding unit that has a structure like depicted in figure 4.5.

4.1.3 Gradual Rounding II (gradual result $\longrightarrow$ packed format)

This section describes a dual mode gradual rounding unit, that is able to round and to compress a FP number from the gradual result format $BUS_{GF}[72:0]$ (section 2.6.5) to the single precision or the double precision packed FP representation $BUS_{PF}[63:0]$ (section 2.6.1). Like for the general rounding the IEEE rounding options are given by the precision of the destination DBL, the rounding mode RMODE$[1:0]$ and the trap handlers UNF_EN and OVF_EN. The gradual rounding unit outputs the packed representation $BUS_{PF}[63:0]$ corresponding to the value of the IEEE rounded result and the exception flags OVF, INX and UNF corresponding to the occurance of an overflow, inexact or underflow exception. In this case the IEEE rounding is computed on the GF factoring:

$$((s_{GF}, e_{GF}, f_{GF}), \text{TINC}, \text{TINX}) = fact_{GF}(BUS_{GF}[73:0]),$$

so that according to corollary 2.21 the rounded result can be specified by

$$BUS_{PF}[63:0] = \text{PF}(ground2((s_{GF}, e_{GF} + wec, f_{GF}), \text{TINC}, \text{TINX})).$$

The only difference between this specification with the rounding function *ground*2, and the specification of the general rounding unit from the previous section with the rounding function *iround*, is the significand rounding according to lemma 2.7 and 2.20. For the significand rounding in the gradual rounding unit, the computation of the rounding decision RINC from the previous section, has to be substituted by the gradual rounding decision GRINC according to equation 2.60. Moreover, the rounding inexactness could also be caused in the previous gradual rounding step, so that the rounding inexact signal INX1 has to be substituted by TINX_2 according to equation 2.61.

All remaining computations are independent of the tag bits. From this point of view, the GF factorings (without tag bits) are a subset of the RF factorings and with the definition of $\text{F}_{GF}[-1] = \text{F}_{GF}[53] = \text{F}_{GF}[54] = 0$ we can interpret a GF factoring input as a normalized RF factoring. Thus, the remaining part of the general rounding implementation from the previous section could be used identically also for the implementation of the gradual rounding unit.

Nevertheless, we will consider some additional changes to optimize the gradual rounding implementation. These optimizations are based on the property that the significands of all non-zero numbers are already normalized in the gradual result format. Not the whole implementation will be involved in the optimizations. The implementation of the post-normalization(see figure 4.5(3.a)), the *exponent rounding + exceptions* (see figure 4.5(4.b)) and the packing are used from the previous section like depicted in figure 4.6. Therefore, only the *denormalization, gradual rounding and exponent wrapping* circuit in this figure will be further specified. The computations in this circuit combine the computations of the normalization shift (see figure 4.5 circuit (1.)), the significand rounding (see figure 4.5 circuit (2.)), the exponent part of the post-normalization shift (see figure 4.5 circuit (3.b)) and the exponent wrapping circuit (see figure 4.5 circuit (4.a)) from the previous section.

For the considerations about the normalization shift distance we can assume non-zero operands like in the previous section. Thus, for the gradual result format we can use $lzii = 2$, so that in lemma 4.1 and 4.2, the computations of F''', SFTMASK and HF can be optimized in the following way:

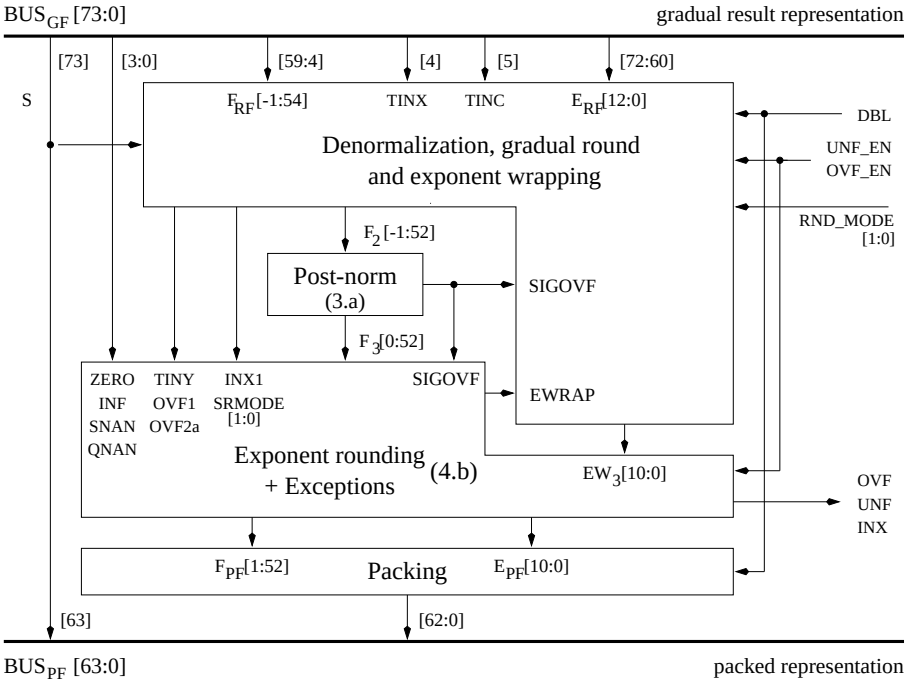

Figure 4.6: Structure of the gradual round unit.

Lemma 4.6 *In the gradual rouning unit* F''', SFTMASK, *and* HF *and can be computed by*

$$\begin{aligned}
\text{F}'''[0{:}63] &= \begin{cases} (\text{F}_{GF}[0:52], 0^{11}) & \textit{if } \overline{\text{TINY}} \textit{ OR } \text{UNF_EN} \\ \text{CLS}((0^2, \text{F}_{GF}[0{:}52], 0^9), <\text{E}_{GF}[5{:}0]>) & \textit{otherwise} \end{cases} \\
\text{SFTMASK}[0{:}53] &= (\text{HDEC}(<\text{E}_{GF}[5{:}0]>)[63{:}10] \textit{ NOR } \text{MASK1}) \textit{ NOR } \text{MASK0} \\
<hf[6{:}0]> &= <e_{GF}[5{:}0]> + <(111110)> .
\end{aligned}$$

Proof: Because $lzii = 2$ and $\text{E}_{GF}[5{:}0] = bin_0^5(e_{GF} - e_{min} + 2)$, we get

$$sfta = \begin{cases} 2 & \text{if } \overline{\text{TINY}} \text{ OR UNF\_EN} \\ <\text{E}_{GF}[5{:}0]> & \text{otherwise,} \end{cases} \tag{4.58}$$

so that with $\text{CLS}((0^2, \text{F}_{GF}[0{:}52], 0^9), 2) = (\text{F}_{GF}[0:52], 0^{11})$, we have as required

$$\text{F}'''[0{:}63] = \begin{cases} (\text{F}_{GF}[0:52], 0^{11}) & \text{if } \overline{\text{TINY}} \text{ OR UNF\_EN} \\ \text{CLS}((0^2, \text{F}_{GF}[0{:}52], 0^9), <\text{E}_{GF}[5{:}0]>) & \text{otherwise.} \end{cases}$$

The formula for SFTMASK in this lemma can be written as

$$\text{SFTMASK}[0{:}53] = \begin{cases} 1^{54} & \text{if MASK1} \\ 0^{54} & \text{if MASK0} \\ \text{HDEC}(<\text{E}_{GF}[5:0]>)[63{:}10] & \text{otherwise.} \end{cases}$$

and differs from equation 4.17 by the substitution of $sfta$ with $<\text{E}_{GF}[5:0]>$. According to equation 4.58 the value $sfta$ could only differ from $<\text{E}_{GF}[5:0]>$, if ($\overline{\text{TINY}}$ OR UNF_EN). But in the case ($\overline{\text{TINY}}$ OR UNF_EN) $\Longleftrightarrow$ (($e_{GF} - e_{min} \geq 0$) OR UNF_EN), we also have MASK1 $\Longleftrightarrow$ (($e_{GF} - e_{min} + 2 \geq 0$) OR UNF_EN) and $\text{SFTMASK}[0{:}53] = 1^{54}$. Thus, $<\text{E}_{GF}[5:0]>$ is equal to $sfta$, whenever these values are involved in the computations. This completes the proof of the equation for SFTMASK. The formula for $<\text{HF}[6{:}0]>$ follows directly from lemma 4.2 and $\text{LZII}[5{:}0] = (000010)$. □

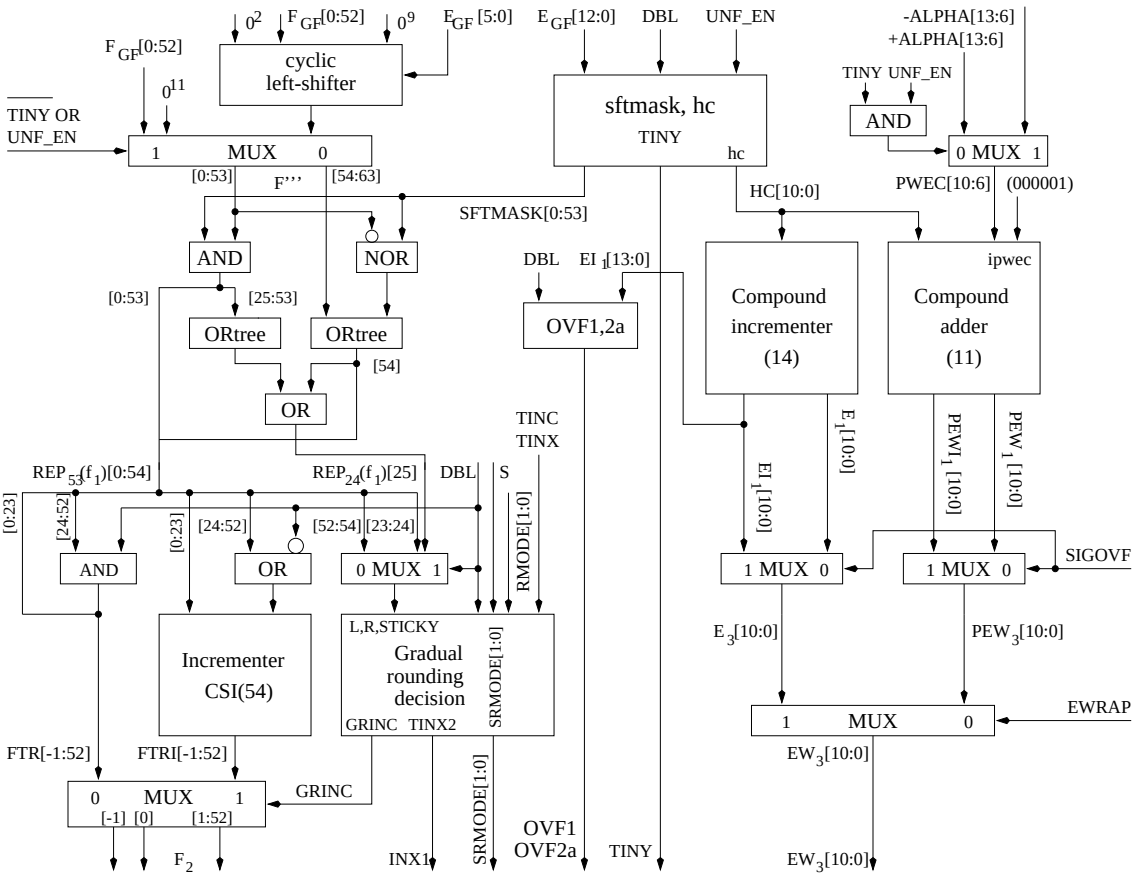

Figure 4.7: Implementation of the '*denormalization, gradual round and exponent wrapping*' circuit in the gradual rounding unit.

Lemma 4.7 *With the incremented predicted wrapping exponent correction*

$$ipwec \quad = \quad pwec + 1 \quad = \quad <(\text{PWEC}[13{:}6], 000001)>_2$$

the predicted exponents $pew_1 = e_{1b} + pwec$ *and* $pewi_1 = ei_{1b} + pwec$ *can be computed by*

$$\begin{aligned} pew_1 &= hc + ipwec \\ pewi_1 &= hc + ipweci + 1 \end{aligned}$$

using a compound adder. Based on them, the predicted wrapped exponent pew_3 *can be computed by the selection*

$$pew_3 \quad = \quad \begin{cases} pewi_1 & \textit{if } \text{SIGOVF} \\ pew_1 & \textit{otherwise.} \end{cases}$$

Proof: The equations for pew_1 and $pewi_1$ follow directly from $hc + 1 = e_{1b}$ in lemma 4.2. Starting from equation 4.33 we get

$$pew_3 \quad = \quad e_{3b} + pwec \quad = \quad \begin{cases} ei_{1b} + pwec & \text{if SIGOVF} \\ e_{1b} + pwec & \text{otherwise} \end{cases} \quad = \quad \begin{cases} pewi_1 & \text{if SIGOVF} \\ pew_1 & \text{otherwise.} \end{cases}$$

as required by the lemma. □

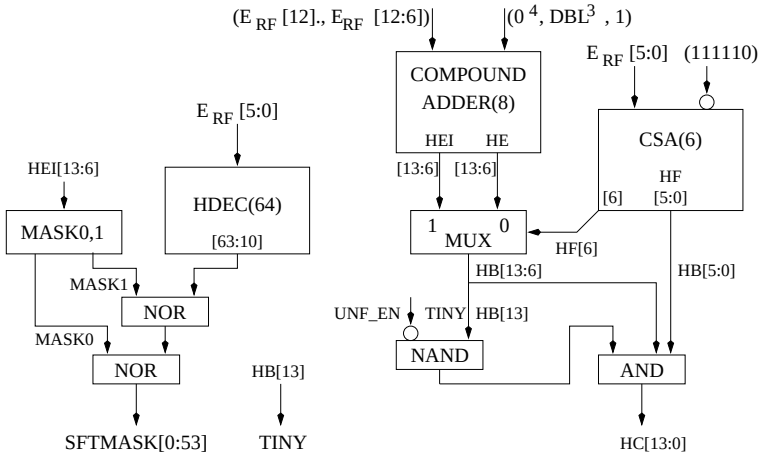

Figure 4.8: *'sftmask, hc and* TINY' circuit in the gradual rounding unit.

Figure 4.7 depicts the implementation of the *'denormalization, gradual rounding and exponent wrapping'* circuit including the necessary changes for the gradual rounding and the optimizations corresponding to lemma 4.6 and 4.7. The computations of pew_1 and $pewi_1$ corresponding to lemma 4.7 use a 11-bit compound adder, the *gradual rounding decision* circuit contains the implementation of equations 2.60, 2.61, 4.18 and 4.19 and the *'*OVF*1 and* OVF*2a'* circuit contains the implementation of the equations 4.15 and 4.16. The optimized implementation of the *'sftmask, hc and* TINY*'* circuit corresponding to lemma 4.6 is depicted in figure 4.8.

4.1.4 Packing III (normalized $\longrightarrow$ packed format)

This section describes a packing unit, that is able to convert a FP number from the normalized format $BUS_{NF}[69:0]$ (section 2.6.3) to the single precision or the double precision packed FP representation $BUS_{PF}[63:0]$ (section 2.6.1). Like in the previous sections the precision of the packed FP format is signaled by the bit DBL.

The packing conversion is the combination of the two steps: (i) a bounded normalization shift from the normalized to the unpacked format following section 2.6.3, p.43 and (ii) a conversion from the unpacked to the packed format following the descriptions of section 2.6.2, p.41.

The bounded normalization shift can be implemented like in the gradual rounding unit, because also in the case of the normalized input format, the input factoring has a normalized significand for all non-zero values. Because in this case the input factoring already has the correct value of the result and no rounding has to be computed on the significand, all bits of the significand are used for the result and no masking of the shifted significand is necessary. Moreover, trapped underflows do not have to be considered in this case. Thus,

$$\begin{aligned} \mathrm{F}_{PF}[0:52] &= \mathrm{F}''[0:52] = \mathrm{F}'''[0:53] \\ &= \begin{cases} \mathrm{CLS}((0^2, \mathrm{F}_{NF}[0:52], 0^9), < \mathrm{E}_{NF}[5:0] >)[0:52] & \text{if TINY} \\ \mathrm{F}_{NF}[0:52] & \text{otherwise.} \end{cases} \end{aligned}$$

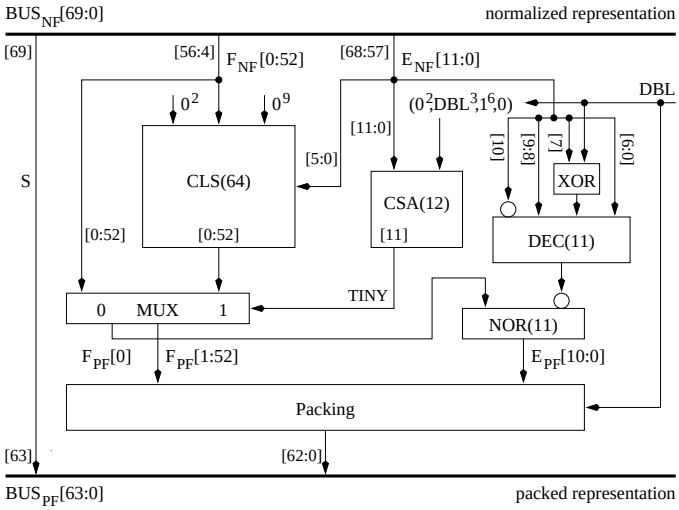

Figure 4.9: Structure of the packing unit.

With $-e_{min} =< (0^2, \text{DBL}^2, 1^6, 0) >_2$ for single and double precision, the condition

$$\text{TINY} \Longleftrightarrow (e_{NF} - e_{min} < 0)$$

is detected as sign bit of the sum $< \text{E}_{NF}[11:0] >_2 + < (0^2, \text{DBL}^2, 1^6, 0) >_2$. Because for normalized numbers with $(\text{F}_{UF}[0] = 1)$, the exponent is not changed in equation 2.90, namely $\text{E}_{UF}[11{:}0] = \text{E}_{NF}[11{:}0]$, the packed exponent representation can be computed according to equation 2.84 that is equivalent to

$$\text{E}_{PF}[10:0] = \overline{bin_0^{10}(< (\overline{\text{E}_{NF}[10]}, \text{E}_{NF}[9:8], \text{E}_{NF}[7] \oplus \text{DBL}, \text{E}_{NF}[6:0]) > -1)} \text{ NOR } \text{F}_{UF}[0].$$

A final packing selection according to equation 4.57 then yields the packed result representation $BUS_{PF}[63{:}0]$. This completes the description of the packing unit, that is implemented like depicted in figure 4.9.

4.2 Addition/Subtraction

4.2.1 Addition/Subtraction I (normalized $\longrightarrow$ representative format)

This section describes a FP addition/subtraction unit, that is able to add or to subtract two FP numbers given in the normalized representations (section 2.6.3):

$$BUSa_{NF}[69:0] = (\text{SA}, \text{EA}[11:0], \text{FA}[0:52], \text{ZEROA}, \text{INFA}, \text{QNANA}, \text{SNANA}) \quad (4.59)$$

$$BUSb_{NF}[69:0] = (\text{SB}, \text{EB}[11:0], \text{FB}[0:52], \text{ZEROB}, \text{INFB}, \text{QNANB}, \text{SNANB}), \quad (4.60)$$

which represent the factorings $(sa, ea, fa) = fact_{NF}(BUSa_{NF}[69:0])$ and $(sb, eb, fb) = fact_{NF}(BUSb_{NF}[69:0])$. The mode, whether the addition or the subtraction should be computed, is selected by the input bit SOP. For the special computation of the sign of zero results, also the input of the rounding mode by RMODE$[1:0]$ is required.

In the case, that both operands have representable values, the exact sum/difference $exact_{add/sub}$ is defined by (section 2.2.4):

$$exact_{add/sub} = (-1)^{\text{SA}} \cdot 2^{ea} \cdot fa + (-1)^{\text{SOP}\oplus\text{SB}} \cdot 2^{eb} \cdot fb.$$

If (s_{rc}, e_{rc}, f_{rc}) is a RF factoring of $exact_{add/sub}$ for non-zero representable inputs, then for the general case of arbitrary input values, a RF factoring of the addition/subtractionI output is given by:

$$(s_{RF}, e_{RF}, f_{RF}) = \begin{cases} (0, e_{qNaN}, f_{qNaN}) & \text{if SCQNAN} \\ (s_{inf}, e_{\infty}, f_{\infty}) & \text{if SCINF} \\ (sa, ea, fa) & \text{if SCX} \\ (sb, eb, fb) & \text{if SCY} \\ (s_0, e_0, 0) & \text{if SCZERO} \\ (s_{rc}, e_{rc}, f_{rc}) & \text{otherwise,} \end{cases} \quad (4.61)$$

so that the sum output of the addition/subtraction I unit is specified by the corresponding representation in the representative format $BUS_{RF}[73:0] = \text{RF}(s_{RF}, e_{RF}, f_{RF})$. Moreover, the invalid flag INV should be signaled according to the occurance of an invalid exception.

The computations of the special conditions in equation 4.61 are already summarized in section 2.4.4 by equations 2.20-2.26. We postpone the discussion of the special sign, significand and exponent selections and consider the computation of (s_{rc}, e_{rc}, f_{rc}) for the regular case in the following. For this we assume non-zero representable input operands.

Definition 4.1 *Let* SEFF $=$ SOP $\oplus$ SA $\oplus$ SB. *The case that* SEFF $= 0$ *is called* effective addition *and the case that* SEFF $= 1$ *is called* effective subtraction. *For effective subtractions, we multiply the significands of both operands by* 2. *This operation is called the* pre-shift *and can be computed by a left-shift of the binary significand representations by one bit position. The significands fa' and fb' that include the conditional pre-shift are defined by:*

$$fa' = \begin{cases} 2 \cdot fa & \textit{if } \text{SEFF} = 1 \\ fa & \textit{otherwise} \end{cases} \qquad fb' = \begin{cases} 2 \cdot fb & \textit{if } \text{SEFF} = 1 \\ fb & \textit{otherwise} \end{cases}$$

We define the exponent difference $\delta = ea - eb$ *and the sign of the exponent difference* SDELTA $\Longleftrightarrow (\delta < 0)$. *The "large" operand, (sl, el, fl), the significand of the "small" operand, fs, and the exponent e_1 are defined as follows:*

$$(sl, el, fl) = \begin{cases} (\text{SA}, ea, fa') & \textit{if } \text{SDELTA} = 0 \\ (\text{SOP} \oplus \text{SB}, eb, fb') & \textit{otherwise} \end{cases} \qquad fs = \begin{cases} fb' & \textit{if } \text{SDELTA} = 0 \\ fa' & \textit{otherwise.} \end{cases}$$

$$e_1 = \begin{cases} el - 1 & \textit{if } \text{SEFF} = 1 \\ el & \textit{otherwise.} \end{cases} \quad (4.62)$$

Lemma 4.8 *Based on the previous definitions, the exact sum can be written as*

$$exact_{add/sub} = (-1)^{sl} \cdot 2^{e_1} \cdot (fl + (-1)^{\mathrm{SEFF}}(fs \cdot 2^{-|\delta|})). \tag{4.63}$$

Proof:

$$\begin{aligned}
exact_{add/sub} &= (-1)^{\mathrm{SA}} \cdot 2^{ea} \cdot fa + (-1)^{\mathrm{SOP} \oplus \mathrm{SB}} \cdot 2^{eb} \cdot fb. \\
&= \begin{cases} (-1)^{\mathrm{SA}} \cdot 2^{ea} \cdot (fa + (-1)^{\mathrm{SOP} \oplus \mathrm{SB} \oplus \mathrm{SA}}(fb \cdot 2^{eb-ea})) & \text{if } \delta \geq 0 \\ (-1)^{\mathrm{SOP} \oplus \mathrm{SB}} \cdot 2^{eb} \cdot (fb + (-1)^{\mathrm{SOP} \oplus \mathrm{SB} \oplus \mathrm{SA}}(fa \cdot 2^{ea-eb})) & \text{otherwise} \end{cases} \\
&= \begin{cases} (-1)^{sl} \cdot 2^{el-1} \cdot (fl + (-1)^{\mathrm{SEFF}}(fs \cdot 2^{-|\delta|})) & \text{if } \mathrm{SEFF} = 1 \\ (-1)^{sl} \cdot 2^{el} \cdot (fl + (-1)^{\mathrm{SEFF}}(fs \cdot 2^{-|\delta|})) & \text{otherwise} \end{cases} \\
&= (-1)^{sl} \cdot 2^{e_1} \cdot (fl + (-1)^{\mathrm{SEFF}}(fs \cdot 2^{-|\delta|})).
\end{aligned}$$

□

The most complex part in the addition/subtraction computation corresponding to equation 4.63 is the computation of the *significand sum* $fsum$:

$$fsum = fl + (-1)^{\mathrm{SEFF}} \cdot fs \cdot 2^{-|\delta|}.$$

With the definition of the absolute significand sum $abs\_fsum = |fsum|$, and the sign of $fsum$: $\mathrm{SFSUM} \Longleftrightarrow (fsum < 0)$, we can write $fsum = (-1)^{\mathrm{SFSUM}} \cdot abs\_fsum$, so that

$$exact_{add/sub} = (-1)^{\mathrm{SL} \oplus \mathrm{SFSUM}} \cdot 2^{e_1} \cdot abs\_fsum. \tag{4.64}$$

In the following lemma it is shown, that the 53-representative of the absolute significand sum, $rep_{53}(abs\_fsum)$, meets the requirements for significands in the representative format:

Lemma 4.9 *The 53-representative of the absolute significand sum $rep_{53}(abs\_fsum)$ is smaller than 4 and is either an integral multiple of 2^{-52} or is larger than or equal to 1, as required for significands in the representative format (see section 2.6.4).*

Proof: The absolute significand sum is defined by:

$$abs\_fsum = |fl + (-1)^{\mathrm{SEFF}} \cdot fs \cdot 2^{-|\delta|}|. \tag{4.65}$$

We separate the proof for: (a) effective additions; and (b) effective subtractions. (a) For effective additions, $abs\_fsum = |fl + fs \cdot 2^{-|\delta|}|$. Because $1 \leq fl < 2$, $1 \leq fs < 2$ and $0 < 2^{-|\delta|} \leq 1$, the absolute significand sum is in the range $1 \leq abs\_fsum < 4$. Thus, also the 53-representative of the absolute significand sum is in the range $1 \leq rep_{53}(abs\_fsum) < 4$ and the proof of case (a) is completed.

(b) For effective subtractions, $abs\_fsum = |fl - fs \cdot 2^{-|\delta|}|$. Because of the preshifts fl and fs are now both in the range $[2, 4[$ and fl and fs are both multiples of 2^{-51}. From this, it follows directly, that $0 < abs\_fsum < 4$. For the remaining part of the proof, we differ between the two cases: (i) $|\delta| \leq 1$; and (ii) $|\delta| > 1$.

(i) Because $|\delta| \leq 1$, $fs \cdot 2^{-|\delta|}$ is a multiple of 2^{-52} and abs_fsum is a multiple of 2^{-52}. Thus, also the 53-representative $rep_{53}(abs\_fsum)$ is a multiple of 2^{-52} and the lemma follows for case (i). (ii) Because $|\delta| > 1$, $(fs \cdot 2^{-|\delta|}) < 1$. Thus, $abs\_fsum > 2 - 1 = 1$ and also $rep_{53}(abs\_fsum) > 1$. This completes case (b) and the proof of the whole lemma. □

Because of equation 4.64 and lemma 4.9, the value $val(\mathrm{SL} \oplus \mathrm{SFSUM}, e_1, rep_{53}(abs\_fsum))$ is e_1-53-equivalent to $exact_{add/sub}$. Thus, $(s_{rc}, e_{rc}, f_{rc}) = (\mathrm{SL} \oplus \mathrm{SFSUM}, e_1, rep_{53}(abs\_fsum))$ is a RF factoring of the exact sum $exact_{add/sub}$. In the following the computation of this RF factoring is described.

Definition 4.2 *We define the limited absolute exponent difference deltalim by*

$$deltalim \;=\; \begin{cases} |\delta| & \text{if } |\delta| \le 63 \\ 63 & \text{otherwise.} \end{cases} \tag{4.66}$$

Because $0 \le deltalim \le 63$*, we can use the 6 bit binary representation:* $deltalim = <\text{DELTALIM}[5:0]>$*. Moreover, we define the negated significand* $fsn = (-1)^{\text{SEFF}} \cdot fs$ *and the aligned significand* $fsa = fsn \cdot 2^{-|\delta|}$*.*

Lemma 4.10 *The 53-representative of the absolute significand sum* $rep_{53}(abs\_fsum)$ *can be computed by using the limited absolute exponent difference deltalim instead of* $|\delta|$*:*

$$rep_{53}(abs\_fsum) = rep_{53}(|fl + (-1)^{\text{SEFF}} \cdot fs \cdot 2^{-deltalim}|). \tag{4.67}$$

Proof: We separate the proof for: (a) the case of $|\delta| \le 63$; and (b) the case of $|\delta| > 63$.

(a) For $|\delta| \le 63$, we have $deltalim = |\delta|$, so that the lemma follows directly from equation 4.65.

(b) For $|\delta| > 63$, we have $fsum > 0$ for both effective subtractions and additions, so that $abs\_fsum = fsum$ and $rep_{53}(abs\_fsum) = rep_{53}(fsum)$. Remember, that the significand fl is a multiple of 2^{-53} and $fs < 4$. Let $x_h = fl$, $x_l = (-1)^{\text{SEFF}} \cdot fs \cdot 2^{-|\delta|}$, and $q = 2^{|\delta| - deltalim}$. We then get $abs\_fsum = x_h + x_l$, $x_h = k \cdot 2^{-53}$ for an integer k, $|x_l| < 4 \cdot 2^{-63} < 2^{-53}$, and

$$\begin{aligned} q \cdot x_l &= (-1)^{\text{SEFF}} \cdot fs \cdot 2^{-|\delta|} \cdot 2^{|\delta| - deltalim} \\ &= (-1)^{\text{SEFF}} \cdot fs \cdot 2^{-deltalim} \\ &= (-1)^{\text{SEFF}} \cdot fs \cdot 2^{-63}, \end{aligned}$$

so that also $q \cdot |x_l| < 2^{-53}$. Therefore, lemma 2.16 with $p = 53$ can be used and we get $rep_{53}(x_h + x_l) = rep_{53}(x_h + q \cdot x_l)$. This equation can be written as $rep_{53}(abs\_fsum) = rep_{53}(|fl + (-1)^{\text{SEFF}} \cdot fs \cdot 2^{-deltalim}|)$, so that the proof of the lemma is completed. □

The computation of $rep_{53}(abs\_fsum)$ according to lemma 4.10 is partitioned into the following steps:

1. computation of the limited absolute exponent difference $deltalim = <deltalim[5:0]>$, and the sign of the exponent difference SDELTA.

2. operand swapping (computation of SL,$el =<\text{EL}[11:0]>_2$,$fl =<\text{FL}[-2:52]>_{2neg}$ and $fs =<\text{FS}[-2:52]>_{2neg}$ including the preshifts for effective subtractions)

$$(\text{FA}'[-1:52], \text{FB}'[-1:52]) \;=\; \begin{cases} (\text{FA}[0:52], 0, \text{FB}[0:52], 0) & \text{if SEFF} = 1 \\ (0, \text{FA}[0:52], 0, \text{FB}[0:52]) & \text{otherwise} \end{cases} \tag{4.68}$$

$$(\text{SL}, \text{EL}[11:0], \text{FL}[-2:52]) \;=\; \begin{cases} (\text{SB} \oplus \text{SOP}, \text{EB}[11:0], 0, \text{FB}'[-1:52]) & \text{if SDELTA} \\ (\text{SA}, \text{EA}[11:0], 0, \text{FA}'[-1:52]) & \text{otherwise} \end{cases} \tag{4.69}$$

$$\text{FS}[-2:52] \;=\; \begin{cases} (0, \text{FA}'[-1:52]) & \text{if SDELTA} \\ (0, \text{FB}'[-1:52]) & \text{otherwise} \end{cases} \tag{4.70}$$

3. significand negation of fs for effective subtractions. Because $fs =<\text{FS}[-2:52]>_{neg}$ and $\text{FS}[-2] = 0$, $fsn = (-1)^{\text{SEFF}} fs$ can be computed by

$$fsn \;=\; <\text{FSN}[-2:52]>_{2neg} \tag{4.71}$$

$$\;=\; \begin{cases} <(\overline{\text{FS}[-2:52]})>_{2neg} + 2^{-52} & \text{if SEFF} \\ <\text{FS}[-2:52]>_{2neg} & \text{otherwise.} \end{cases} \tag{4.72}$$

This equation is implemented by a 55-bit incrementer and a 55-bit mux selection.

4. alignment shift of FSN[−2:52] by *deltalim* positions ($fsa = fsn \cdot 2^{-deltalim}$). Because $0 \leq deltalim \leq 63$, fsa can be represented by $fsa =< fsa[-2:115] >_{2neg}$ and because fsn is also represented in the two's complement representation, the fill bit FSA[−2] has to be shifted in for sign extension:

$$\begin{aligned} \text{FSA}[-2:115] &= (\text{FSN}[-2]^{deltalim}, \text{FSN}[-2:52], 0^{63-deltalim}) && (4.73)\\ &= \text{RSFT}(\text{FSN}[-2:52], <\text{DELTALIM}[5:0]>, \text{FSN}[-2], 0) && (4.74) \end{aligned}$$

This right shift is implemented with a 55-bit shifter.

5. significand addition $fsum = fl + fsa$:

$$<\text{FSUM}[-2:115]>_{2neg} = <\text{FL}[-2:52]>_{2neg} + <\text{FSA}[-2:115]>_{2neg}$$

This addition is partitioned into a lower part and into an upper part:

$$\begin{aligned} \text{FSUM}[53:115] &= \text{FSA}[53:115] && (4.75)\\ <\text{FSUM}[-2:52]>_{2neg} &= <\text{FL}[-2:52]>_{2neg} + <\text{FSA}[-2:52]>_{2neg} && (4.76) \end{aligned}$$

The addition of the upper part is implemented by a 55-bit carry-look-ahead adder implementation.

6. conversion for negative $fsum$ (computation of $abs\_fsum =< \text{ABS\_FSUM}[-1:115] >_{neg} = |fsum|$). Because $fsum$ is negative, iff $fsa > fl$, in this case $deltalim = 0$ and $\text{ABS\_FSUM}[53:115] = \text{FSUM}[53:115] = \text{FSA}[53:115] = 0^{63}$. Thus, only the upper part $[-2:52]$ is involved in the conversion

$$\begin{aligned} \text{ABS\_FSUM}[53:115] &= \text{FSUM}[53:115] && (4.77)\\ <\text{ABS\_FSUM}[-1:52]>_{2neg} &= \begin{cases} <(\overline{\text{FSUM}[-1:52]})>_{2neg} + 2^{-52} & \text{if FSUM}[-2]\\ <\text{FSUM}[-1:52]>_{2neg} & \text{otherwise.} \end{cases} && (4.78) \end{aligned}$$

This equation is implemented by a 55 bit incrementer and a 55-bit mux selection. The sign of $fsum$ is given by SFSUM = FSUM[−2].

7. representative computation according to lemma 2.11:

$$\begin{aligned} f_{rc} &= < \text{F}_{rc}[-1:54] >_{neg}\\ &= < \text{REP}_{53}(abs\_fsum)[-1:54] >_{neg}\\ &= < (\text{ABS\_FSUM}[-1:53], Ortree(\text{ABS\_FSUM}[54:115]) >_{neg} \end{aligned}$$

Among these steps only the implementation of the first step has to be further specified. This is done by the following lemma:

Lemma 4.11 *With the computation of*

$$\begin{aligned} \delta &= <\text{DELTA}[13:0]>_2 = ea - eb && (4.79)\\ &= <(0, \text{EA}[12:0])>_2 + <(1, \overline{\text{EA}[12:0]})>_2 + 1 && (4.80)\\ |\delta| &= <\text{ABS\_DELTA}[13:0]> && (4.81)\\ &= \begin{cases} <\overline{\text{DELTA}[13:0]}> + 1 & \textit{if } \text{DELTA}[13]\\ <\text{DELTA}[13:0]> & \textit{otherwise.} \end{cases} && (4.82)\\ \text{DELTAOVF} &= ORtree(\text{ABS\_DELTA}[13:6]) && (4.83) \end{aligned}$$

we get

$$\begin{aligned} \text{SDELTA} &= \text{DELTA}[13] && (4.84)\\ deltalim &= <\text{DELTALIM}[5:0]> && (4.85)\\ &= <(\text{ABS\_DELTA}[5:0]\ OR\ \text{DELTAOVF})> && (4.86)\end{aligned}$$

Proof: The equations for $\delta = <\text{DELTA}[13:0]>_2$ and $|\delta| = <\text{ABS\_DELTA}[13:0]>$ are a straight-forward implementation of the definitions using the properties of two's complement numbers. Obviously, in the two's complement representation DELTA$[13:0]$, the sign of δ is given by SDELTA = DELTA$[13]$. The bit DELTAOVF implements the condition $(|\delta| > 63)$. In equation 4.86, *deltalim* is set to $<111111> = 63$ for DELTAOVF $= 1$ and $deltalim = <\text{ABS\_DELTA}[5:0]> = |\delta|$ for DELTAOVF $= 0$, as required by the definition of *deltalim* in equation 4.66. □

The exponent $e_{rc} = <\text{E}_{rc}[12:0]>_2 = e_1$ (equation 4.62) is computed by:

$$\begin{aligned} e_{rc} &= <e_1[11:0]>_2 && (4.87)\\ &= \begin{cases} <\text{EL}[11:0]>_2 - 1 & \text{if SEFF} = 1\\ <\text{EL}[11:0]>_2 & \text{otherwise.}\end{cases} && (4.88)\end{aligned}$$

This equation is implemented by a 12-bit decrementer and a 12-bit selection mux. The sign computation implements the equation $\text{S}_{rc} = (\text{SL} \oplus \text{SFSUM})$. This completes the description of the computation of (s_{rc}, e_{rc}, f_{rc}). In the following we will integate this result for the regular case with the special cases results according to equation 4.61 and we will consider the recognition of the invalid exception.

We separate the final result selection according to equation 4.61 for the significand, the exponent and the sign of the result. The definitions of the special case conditions SCQNAN, SCINF, SCX, SCY, and SCZERO are given in equations 2.20-2.24. For the computation of the zero condition SCZERO, we first have to detect the condition ZERO_{rc} of zero results for regular operands. Because the computation of ZERO_{rc} based on the result of the regular path would be quite slow, we compute this signal directly from the input operands. Obviously,

$$\text{ZERO}_{rc} \iff \text{SEFF} \wedge \text{ZEROTEST}\left((\text{EA}[11:0], \text{FA}[0:52]) \oplus (\text{EB}[11:0], \text{FB}[0:52])\right) \quad (4.89)$$

This equation will be implemented in the *special cases* circuit.

Based on the special case conditions we get for the significand:

$$\begin{aligned} f_{RF} &= <\text{F}_{RF}[-1:54]>_{neg}\\ &= \begin{cases} f_{\text{QNAN}} = <(0, 1010^{52})>_{neg} & \text{if SCQNAN}\\ f_{\infty} = <(0, 10^{54})>_{neg} & \text{if SCINF}\\ fa = <(0, \text{FA}[0:52], 0^2)>_{neg} & \text{if SCX}\\ fb = <(0, \text{FB}[0:52], 0^2)>_{neg} & \text{if SCY}\\ f_0 = <(0^{56})>_{neg} & \text{if SCZERO}\\ f_{rc} = <\text{F}_{rc}[-1:54]>_{neg} & \text{otherwise.}\end{cases}\end{aligned}$$

By the definition of the special case significand representation $\text{F}_{sc}[-1:54]$

$$\text{F}_{sc}[-1:54] = \begin{cases} (0, 1010^{52}) & \text{if SCQNAN}\\ (0, 10^{54}) & \text{if SCINF}\\ (0, \text{FA}[0:52], 0^2) & \text{if SCX}\\ (0, \text{FB}[0:52], 0^2) & \text{if SCY}\\ 0^{56} & \text{otherwise}\end{cases} \quad (4.90)$$

and the special case condition

$$\text{SPCA} \quad = \quad \text{SCQNAN} \vee \text{SCX} \vee \text{SCY} \vee \text{SCINF} \vee \text{SCZERO}, \tag{4.91}$$

the representation of the significand f_{RF} can be selected by

$$\text{F}_{RF}[-1:54] \quad = \quad \begin{cases} \text{F}_{sc}[-1:54] & \text{if SPCA} \\ \text{F}_{rc}[-1:54] & \text{otherwise.} \end{cases} \tag{4.92}$$

The computations for $\text{F}_{sc}[-1:54]$ and SPCA are implemented in the *special cases* circuit.

Based on the special case conditions, already all four aditional bits of the result representation are given by $\text{ZERO}_{RF} = \text{SCZERO}$, $\text{INF}_{RF} = \text{SCINF}$, $\text{SNAN}_{RF} = 0$ and

$$\text{QNAN}_{RF} \quad \Longleftrightarrow \quad \text{QNANA} \vee \text{QNANB} \vee \text{SCQNAN}. \tag{4.93}$$

For the exponent e_{RF} the selection is even simpler, because for all special value results, we defined $e_{RF} = e_{max} + 1$. Because in addition/subtraction for a special value result, at least one of the input operands has also a special value, and to avoid the distinction between the single and the double precision case, the special exponent representation can be copied from one special input operand for all special value results. We define the condition

$$\text{NREGA} \quad \Longleftrightarrow \quad \text{INFA} \vee \text{QNANA} \vee \text{SNANA}. \tag{4.94}$$

If there is at least one special input operand, then a special exponent representation is copied from the inputs by

$$\text{E}_{sc}[11:0] \quad = \quad \begin{cases} \text{EA}[11:0] & \text{if NREGA} \\ \text{EB}[11:0] & \text{otherwise,} \end{cases} \tag{4.95}$$

so that the exponent e_{RF} of the representative result can be selected by:

$$e_{RF} \quad = \quad <\text{E}_{RF}[12:0]>_2 \tag{4.96}$$

$$= \quad \begin{cases} e_{sc} =<(\text{E}_{sc}[11], \text{E}_{sc}[11:0])>_2 & \text{if SPCA} \\ e_{rc} =<(\text{E}_{rc}[11], \text{E}_{rc}[11:0])>_2 & \text{otherwise.} \end{cases} \tag{4.97}$$

Note, that for zero results the exponent e_{sc} is selected, but in this case it does not matter which value this exponent has, because zero representations in the representative format may contain an arbitrary exponent value.

We define the special case sign S_{sc} and the preliminary sign S'_{RF} by

$$\text{S}_{sc} \quad = \quad \begin{cases} 0 & \text{if SCQNAN} \\ \text{SA} & \text{if SCX} \\ \text{SB} & \text{if SCY} \\ (\text{SA} \wedge \text{INFA}) \vee (\text{SB} \wedge \text{INFB}) & \text{otherwise} \end{cases} \tag{4.98}$$

$$\text{S}'_{RF} \quad = \quad \begin{cases} \text{S}_{sc} & \text{if SPCA} \\ \text{S}_{rc} & \text{otherwise.} \end{cases} \tag{4.99}$$

Because the rounding mode RMI is encoded by $\text{RMODE}[1:0] = (11)$ according to table 2.3, and $(\text{SA} \wedge (\text{SB} \oplus \text{SOP})) \equiv (\text{SL} \wedge \overline{\text{SEFF}})$ we then get the sign of the result s_{RF} according to equation 2.15 by

$$\text{S}_{RF} \quad = \quad \begin{cases} \text{S}_0 & \text{if SCZERO} \\ \text{S}'_{RF} & \text{otherwise.} \end{cases} \tag{4.100}$$

$$= \quad \begin{cases} (\text{SEFF} \wedge \text{RMODE}[1] \wedge \text{RMODE}[0]) \vee (\text{SL} \wedge \overline{\text{SEFF}}) & \text{if SCZERO} \\ \text{S}'_{RF} & \text{otherwise.} \end{cases} \tag{4.101}$$

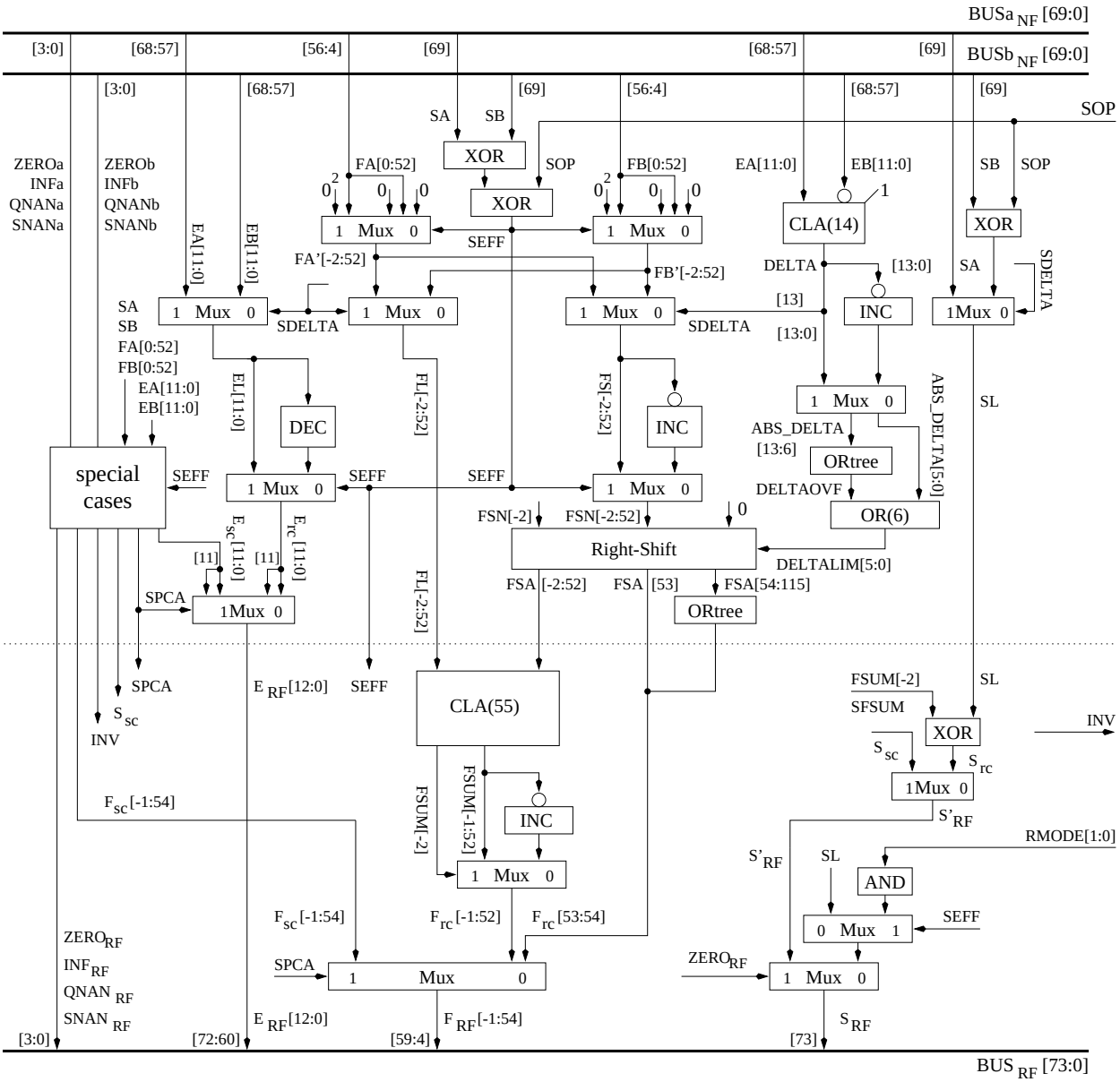

Figure 4.10: Structure of the addition/subtraction unit I.

In this way, the sum output in the representative format is given by:

$$BUS_{RF}[72:0] \;=\; (\mathrm{S}_{RF}, \mathrm{E}_{RF}[12:0], \mathrm{F}_{RF}[-1:54], \mathrm{ZERO}_{RF}, \mathrm{INF}_{RF}, \mathrm{QNAN}_{RF}, \mathrm{SNAN}_{RF}) \tag{4.102}$$

The cases for the occurance of an invalid exception are listed in table 2.5. Obviously, the invalid exception occurs, iff the addition/subtraction results in a quiet NaN, where $\mathrm{SCQNAN} = 1$, so that

$$\mathrm{INV} \iff \mathrm{SCQNAN}. \tag{4.103}$$

This completes the description of the addition/subtraction I implementation which is depicted in figure4.10. The only part which is included in this figure without details is the special cases circuit. This special cases circuit includes the computations of equations 2.20-2.24, 4.89, 4.90-4.91,4.93-4.95, and 4.98.

4.2.2 Addition/Subtraction II (normalized $\longrightarrow$ gradual result format)

Like in the previous section also in this section the FP addition/subtraction is computed from the inputs of the normalized representations $BUSa_{NF}[69:0]$ and $BUSb_{NF}[69:0]$ (section 2.6.3), the rounding mode represented by RMODE$[1:0]$ and the bit SOP that signals the case of addition or subtraction. But in contrast to the previous implementation where a representative of the exact operation result had to be delivered, in this case the gradual rounding function $ground1$ has to be computed on the exact operation result. After this gradual rounding step the sum/difference should be output in the gradual result format $BUS_{GF}[73:0]$ (section 2.6.5). Formally, with the notation from the previous section and with $((s_{grc}, e_{grc}, f_{grc}), \text{TINC}, \text{TINX}) = ground1_{mode}(s_{rc}, e_{rc}, f_{rc})$, the required addition/subtraction result is based on the following GF factoring (Note, that the rounding can be computed on the RF factoring (s_{rc}, e_{rc}, f_{rc}) instead of a factoring of the exact operation result according to lemma 2.7):

$$((s_{GF}, e_{GF}, f_{GF}), \text{TINC}_{GF}, \text{TINX}_{GF}) = \begin{cases} ((0, e_{qNaN}, f_{qNaN}), 0, 0) & \text{if SCQNAN} \\ ((s_{inf}, e_{\infty}, f_{\infty}), 0, 0) & \text{if SCINF} \\ ((sa, ea, fa), 0, 0) & \text{if SCX} \\ ((sb, eb, fb), 0, 0) & \text{if SCY} \\ ((s_0, e_0, 0), 0, 0) & \text{if SCZERO} \\ ((s_{grc}, e_{grc}, f_{grc}), \text{TINC}, \text{TINX}) & \text{otherwise,} \end{cases} \tag{4.104}$$

so that the sum output of the addition/subtraction unit in this section is specified by the corresponding gradual result representation $BUS_{GF}[73:0] = \text{GF}((s_{GF}, e_{GF}, f_{GF}), \text{TINC}, \text{TINX})$. The occurance of an invalid exception should be signaled by the bit INV also in this case.

The special cases conditions and values in equation 4.104 are identical to that in the specification of the previous section. In the implementation of this special cases selection, the only difference to the previous section is that a representation in the gradual result format has 3 bits less in the significand, which have been filled with zeros in the representative format. Moreover, the gradual result format requires two additional rounding tags, which have to be zero for special value results. For the special cases selections, these small adjustments are integrated in the implementation depicted in figure 4.11. Also in the equations, that are implemented in the special cases circuit, the selections for bit positions $[-1]$ and $[53:54]$ have to be neglected. This already completes the description of the special cases computation and we only have to describe the computation of the gradual result representation of $((s_{grc}, e_{grc}, f_{grc}), \text{TINC}, \text{TINX})$ in the following.

The computation of the GF factoring $((s_{grc}, e_{grc}, f_{grc}), \text{TINC}, \text{TINX})$ can be based on the computation of the RF factoring (s_{rc}, e_{rc}, f_{rc}) from the previous section:

$$(s_{rc}, e_{rc}, f_{rc}) = (\text{SL} \oplus \text{SFSUM}, e_1, rep_{53}(|fl + (-1)^{\text{SEFF}} \cdot fs \cdot 2^{-deltalim}|)),$$

so that

$$\begin{aligned} ((s_{grc}, e_{grc}, f_{grc}), \text{TINC}, \text{TINX}) &= ground1_{mode}(s_{rc}, e_{rc}, f_{rc}) && (4.105) \\ &= post\_norm(sgrnd1_{mode \star s}(\eta(s_{rc}, e_{rc}, f_{rc}))). && (4.106) \end{aligned}$$

The three additional steps of the normalization shift, the rounding computation and the post-normalization shift could have a large additional delay in a straight-forward implementation. To speed up the computations, we divide the implementation into two parallel paths that work under different assumptions. The computations in each path can then be simplified and some of the computation steps only have to be considered exclusively in one of the two paths. Such a 'two path' approach for floating-point addition was first described in [14]. In this description the two paths differ by the assumptions on the magnitude of the exponent difference: the *far path* is defined for large exponent differences $|\delta| > 1$, and the *near path* is defined for small exponent differences $|\delta| \leq 1$.

Our partitioning is slightly different. Based on the following definition of the path selection condition IS_R, we define the *'R'-path* (R for Rounding) for IS_R $= 1$ and the *'N'-path* (N like Near, Negation and Normalization) for IS_R $= 0$. As will be shown later, the advantage of our approach is that a conventional implementation of a far path can be used to implement also the *'R'-path*, but the implementation of the *'N'-path* could be simplified in comparison to a *near path* implementation.

Definition 4.3 *We define the path selection condition* IS_R *based on the computation of* (s_{rc}, e_{rc}, f_{rc}) *from the previous section with* $fsum = fl + (-1)^{\text{SEFF}} \cdot fs \cdot 2^{-deltalim}$*:*

$$\text{IS\_R} \iff (\overline{\text{SEFF}} \vee (fsum \in [1,4[)\,, \tag{4.107}$$

i.e., the results of the 'R'-path have to be valid, if (IS_R $= 1$) *and the results of the 'N'-path have to be valid, if* (IS_R $= 0$)*, so that a valid result could be selected by*

$$((s_{grc}, e_{grc}, f_{grc}), \text{TINC}, \text{TINX}) = \begin{cases} ((r\_s, r\_e, r\_f), \text{R\_TINC}, \text{R\_TINX}) & \textit{if } \text{IS\_R} \\ ((n\_s, n\_e, n\_f), \text{N\_TINC}, \text{N\_TINX}) & \textit{otherwise.} \end{cases}$$

Lemma 4.12 *For the two paths we get the following properties*

(a) 'R'-path: IS_R $\Longrightarrow fsum \in [1,4[$

(b) 'N'-path: $\overline{\text{IS\_R}} \Longrightarrow \text{SEFF} = 1$

(c) 'N'-path: $\overline{\text{IS\_R}} \Longrightarrow \delta \in \{-1, 0, 1\}$

(d) 'N'-path: $\overline{\text{IS\_R}} \Longrightarrow fsum \in]-2, 1[$ *AND* $fsum$ *is an integral multiple of* 2^{-52}.

Proof: (a) Because one part of the definition of IS_R already includes the condition $fsum \in [1,4[$, we have to show part (a) only for the case of SEFF $= 0$. For this case of effective additions it was already shown in part (a) of the proof of lemma 4.9, that $fsum \in [1,4[$.

Part (b) follows directly from the definition of the path selection condition IS_R.

(c) For IS_R $= 1$, we have an effective subtraction with $fsum < 1$. In effective subtractions both significands have been preshifted, so that both fl and fs are in the range $[2,4[$. Thus,

$$\begin{aligned} (fsum < 1) &\Longrightarrow (fl - fs \cdot 2^{-deltalim} < 1) \\ &\Longrightarrow (fs \cdot 2^{-deltalim} > 1) \\ &\Longrightarrow (2^{-deltalim} > 0.25) \\ &\Longrightarrow (deltalim < 2) \end{aligned}$$

This last condition is only fulfilled for exponent differences $\delta \in \{-1, 0, 1\}$, as required by the lemma.

(d) From part (c) we know that for IS_R $= 1$, we get effective subtractions with $deltalim \leq 1$. Because of the preshifts, the significands fl and fs are both integral multiples of 2^{-51}, so that the aligned significand $fs \cdot 2^{-deltalim}$ is an integral multiple of 2^{-52}. Thus, also the significand sum $fsum$ is an integral multiple of 2^{-52}. Because in general for effective subtractions, $fsum \in]-2, 4[$, and for IS_R $= 1$, we have $fsum < 1$, we also get $fsum \in]-2, 1[$, as required. □

These properties of the two paths make the following optimizations possible:

(a) Because of (a), there can be no negative $fsum$ in the 'R'-path, so that the conversion step can be avoided in this path. Moreover, we know from (a), that in the 'R'-path the range of the significand sum $fsum$ consists only of two binades, so that only a very small a normalization shift by at most one position is required in the 'R'-path.

(b) Because of (b), we can use SEFF $= 1$ in the whole computations of the 'N'-path and optimize the implementation accordingly.

(c) Because of (c), $deltalim$ in the 'N'-path can be determined already from the two least significant bits in the two's complement representation of the exponent difference.

(d) Because of (d), also after the conversion step and the normalization shift, the significand is a multiple of 2^{-52}, so that the rounding computation by the function $ground1$ does no change on the significand and the rounding can be neglegted in the 'N'-path.

The additional advantanges of the 'N'-path in our approach in comparison to the 'near'-path from [], are the properties in (b) and in (d). The main structure of our implementation of the addition/subtraction II unit is depicted in figure 4.11, that uses the results of the 'R'-path computations $((r\_s, r\_e, r\_f)$, R_TINC, R_TINX), the result of the 'N'-path computations $((n\_s, n\_e, n\_f)$, N_TINC, N_TINX) and the condition IS_R, that decides, which result has to be selected according to definition 4.3. Moreover, the special case computations from the previous section according to equation 4.104 are adopted for the output in the gradual result format in this figure.

In the following, we describe the implementation of the 'R'-path and the 'N'-path separately, after giving a definition of some values that will be used in both paths:

Definition 4.4 *We define the significand fso (o for one's complement), where the conditional two's complement negation from the significand fsn is replaced by a conditional one's complement negation:*

$$\begin{aligned} fso &= <\text{FSO}[-2:52]>_{2neg} && (4.108)\\ &= <\text{FS}[-2:52] \oplus \text{SEFF}>_{2neg} && (4.109)\\ &= <\text{FSN}[-2:52]>_{2neg} - \text{SEFF} \cdot 2^{-52} && (4.110)\\ &= fsn - \text{SEFF} \cdot 2^{-52} && (4.111) \end{aligned}$$

and we define the corresponding values that are based on fso instead of fsn:

$$\begin{aligned} fsoa &= <\text{FSOA}[-2:115]>_{2neg} && (4.112)\\ &= <(\text{SEFF}^{deltalim}, \text{FSO}[-1:52], \text{SEFF}^{63-deltalim})>_{2neg} && (4.113) \end{aligned}$$

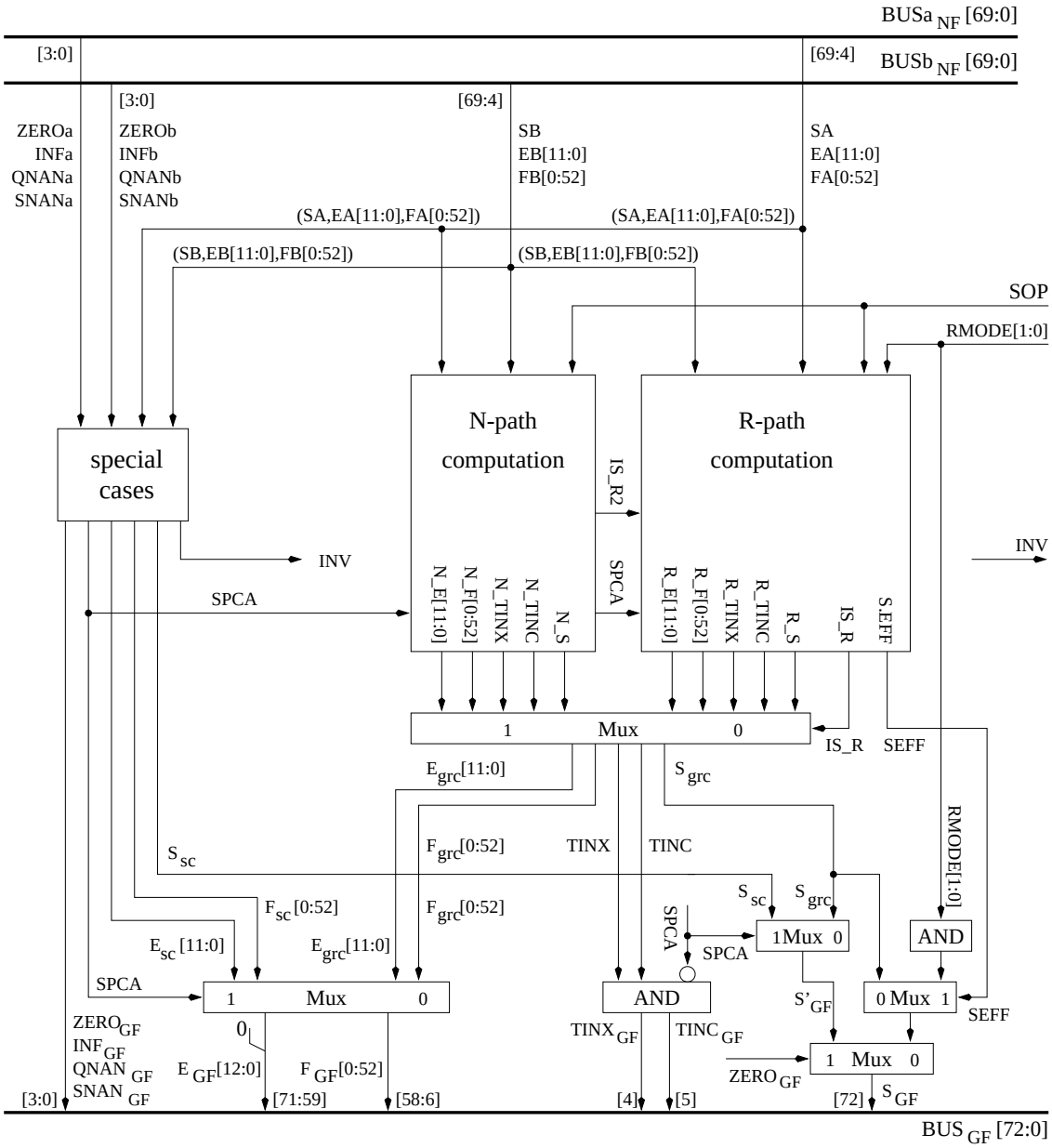

Figure 4.11: Structure of the addition/subtraction unit II.

$$
\begin{aligned}
&= fso \cdot 2^{-deltalim} + \text{SEFF} \cdot (2^{-52-deltalim} - 2^{-115}) && (4.114)\\
&= fsn \cdot 2^{-deltalim} - \text{SEFF} \cdot 2^{-115} && (4.115)\\
&= fsa - \text{SEFF} \cdot 2^{-115} && (4.116)\\
fosum &= <\text{FOSUM}[-2:116]>_{2neg} && (4.117)\\
&= fl + fsoa && (4.118)\\
&= fsum - \text{SEFF} \cdot 2^{-115} && (4.119)
\end{aligned}
$$

'R'-path The computations in the 'R'-path are described on the basis of the adder implementation from the previous section. As discussed above (see lemma 4.12(a)), we can use for the 'R'-path: $abs\_fsum = fsum \in [1,4[$ and SFSUM $= 0$. Based on this, the required factoring in the 'R'-path $((r\_s, r\_e, r\_f), \text{R\_TINC}, \text{R\_TINX})$ can be written as

$$((r\_s, r\_e, r\_f), \text{R\_TINC}, \text{R\_TINX}) \quad = \quad post\_norm(sgrnd1_{mode\star \text{SL}}(\eta(\text{SL}, e_1, fsum))) \tag{4.120}$$

Definition 4.5 *For $f \in [1,4[$, we define the generalized post-normalization shift by*

$$gpost\_norm(s,e,f) = \begin{cases} (s, e+1, f/2) & \text{if } f \in [2,4[\\ (s,e,f) & \text{if } f \in [1,2[\end{cases} \tag{4.121}$$

Lemma 4.13 *In the 'R'-path, the computation of (r_s, r_e, r_f) can be simplified to*

$$(r\_s, r\_e, r\_f) \quad = \quad \begin{cases} gpost\_norm(\text{SL}, e_1, rnd_{mode\star \text{SL},52}(fsum)) & \text{if } fsum \in [1,2[\\ gpost\_norm(\text{SL}, e_1, rnd_{mode\star \text{SL},51}(fsum)) & \text{if } fsum \in [2,4[\end{cases} \tag{4.122}$$

Proof: It follows directly from the definition of the gradual rounding function $grnd$, that for zero rounding tags at the input like in $(rfsum, \text{TINX}, \text{TINC}) = grnd_{mode\star s,\lambda}(fsum, 00))$, conventional rounding delivers the same rounded result, so that we also have $rfsum = rnd_{mode\star s,\lambda}(fsum)$. Moreover, we use for the reduction that $fa, fb < (2 - 2^{-52})$, so that $fsum \in [1, (4 - 2^{-51})]$. Because 1 and $(4 - 2^{-51})$ are integral multiples of 2^{-51}, for $fsum \in [1, (4 - 2^{-51})]$ and $\lambda \geq 51$, also the rounded result $rfsum = rnd_{mode\star s,\lambda}(fsum)$ is in the same range, namely $rfsum \in [1, (4 - 2^{-51})]$. In the following we differ between the two cases: (a) $fsum \in [1,2[$ and (b) $fsum \in [2,4[$.

(a) For $fsum \in [1,2[$, the normalization shift can be neglected, so that with definition of the function $sgrnd1$ we get

$$post\_norm(sgrnd1_{mode\star \text{SL}}(\eta(\text{SL}, e_1, fsum))) \quad = \quad post\_norm(\text{SL}, e_1, grnd1_{mode\star \text{SL}}(fsum))$$

and thus $(r\_s, r\_e, r\_f) \;=\; gpost\_norm(\text{SL}, e_1, rnd_{mode\star \text{SL},52}(fsum))$.

(b) For $fsum \in [2,4[$, we get $\eta(\text{SL}, e_1, fsum) = (\text{SL}, e_1 + 1, fsum/2)$, so that because of $fsum \leq (4 - 2^{-51})]$:

$$\begin{aligned} (r\_s, r\_e, r\_f) \quad &= \quad gpost\_norm(\text{SL}, e_1 + 1, rnd_{mode\star \text{SL},52}(fsum/2)) \\ &= \quad gpost\_norm(\text{SL}, e_1, rnd_{mode\star \text{SL},51}(fsum)) \end{aligned}$$

□

Definition 4.6 *Based on the previous lemma and with the definition of the significand overflow condition*

$$\text{COND}_{[2,4[} \iff fsum \in [2,4[, \tag{4.123}$$

we define the rounded significand rnd_fsum *by*

$$\begin{aligned} rnd\_fsum &= <\text{RND\_FSUM}[-1{:}52]>_{neg} \\ &= \begin{cases} rnd_{mode\star\text{SL},51}(fsum) & \textit{if } \text{COND}_{[2,4[} \\ rnd_{mode\star\text{SL},52}(fsum) & \textit{otherwise,} \end{cases} \end{aligned}$$

so that $(r\_s, r\_e, r\_f) = gpost\_norm(\text{SL}, e_1, rnd\_fsum)$.

In the following, the computation of the rounded significand rnd_fsum and the rounding functions $rnd_{mode\star\text{SL},52}(fsum)$ and $rnd_{mode\star\text{SL},51}(fsum)$ are described using the injection-based rounding reduction from section 2.5.2. We denote the additive rounding injection by $inj_{[1,2[}$ for $fsum \in [1,2[$ and by $inj_{[2,4[}$ for $fsum \in [2,4[$. With $srmode = mode \star \text{SL}$, these injections are defined by

$$inj_{[1,2[} = \begin{cases} 0 & \text{if } srmode = RZ \\ 2^{-53} & \text{if } srmode = RN \\ 2^{-52} - 2^{-115} & \text{otherwise} \end{cases}$$

$$inj_{[2,4[} = \begin{cases} 0 & \text{if } srmode = RZ \\ 2^{-52} & \text{if } srmode = RN \\ 2^{-51} - 2^{-115} & \text{otherwise.} \end{cases}$$

Based on the injections, we can reduce the previous rounding functions to

$$\begin{aligned} rnd_{srmode,51}(fsum) &= rnd_{RZ,51}(fsum + inj_{[2,4[}) && (4.124) \\ &= rnd_{RZ,51}(fosum + \text{SEFF} \cdot 2^{-115} + inj_{[2,4[}) && (4.125) \\ rnd_{srmode,52}(fsum) &= rnd_{RZ,52}(fsum + inj_{[1,2[}) && (4.126) \\ &= rnd_{RZ,52}(fosum + \text{SEFF} \cdot 2^{-115} + inj_{[1,2[}). && (4.127) \end{aligned}$$

According to definition 4.4, the significand sum $fosum$ consists of the significands fl and $fsoa$. Thus, $\text{FL}[-1:52]$ and $\text{FSOA}[-1:115]$ can be interpreted as a carry-save representation of $fosum$. We compress this carry-save representation by a half-adder-line with the sum outputs $\text{SFOSUM}[-1:115]$ and carry outputs $\text{CFOSUM}[-1:51]$, so that $<\text{SFOSUM}[-1{:}115]>_{neg} + <\text{CFOSUM}[-1{:}51]>_{neg} = <\text{FL}[-1{:}52]>_{neg} + <\text{FSOA}[-1{:}115]>_{neg}$, and $fosum = sfosum + cfosum$. After that, we partition the addition of

$$\begin{aligned} finj = <\text{FINJ}[-1{:}115]>_{neg} &= fsum + inx_X && (4.128) \\ &= fosum + \text{SEFF} \cdot 2^{-115} + inj_X && (4.129) \\ &= sfosum + cfosum + \text{SEFF} \cdot 2^{-115} + inj_X && (4.130) \end{aligned}$$

into three parts: an upper part with positions $[-1:51]$, a mid part including positions $[52{:}53]$, and a lower part with positions $[54{:}115]$. The additions are computed separately for these three parts considering the carries from the lower to the mid part and from the mid part to the upper part.

The binary representation of the injection constants $inj_{[1,2[}$ and $inj_{[2,4[}$ could have non-zero digits only in positions $[52:115]$, which are in the mid part and in the lower

part, so that the injections can be represented by $inj_{[1,2[} = <\text{INJ}_{[1,2[}[52:115]>_{neg}$ and $inj_{[2,4[} = <\text{INJ}_{[2,4[}[52:115]>_{neg}$ with:

$$\text{INJ}_{[1,2[}[52:53] = \begin{cases} 00 & \text{if } srmode = RZ \\ 01 & \text{otherwise} \end{cases} \tag{4.131}$$

$$\text{INJ}_{[2,4[}[52:53] = \begin{cases} 11 & \text{if } srmode = RI \\ 10 & \text{if } srmode = RN \\ 00 & \text{otherwise} \end{cases} \tag{4.132}$$

$$\text{INJ}[54:115] = \text{INJ}_{[1,2[}[54:115] = \text{INJ}_{[2,4[}[54:115] \tag{4.133}$$

$$= \begin{cases} 1^{60} & \text{if } srmode = RI \\ 0^{60} & \text{otherwise.} \end{cases} \tag{4.134}$$

Because in the lower part we have

$$\begin{aligned} lpart &= <\text{SFOSUM}[54:115]>_{neg} + <\text{INJ}[54:115]>_{neg} + \text{SEFF} \cdot 2^{-115} \\ &< 2^{-52}, \end{aligned}$$

there can be at most one carry bit from the lower part into position [53] of the mid part. This carry bit into position [53] is called C53 with $\text{C53} \iff (lpart \geq 2^{-53})$.

With the consideration of the carry C53 we have in the mid part:

$$mpart = <\text{SFOSUM}[52:53]>_{neg} + <\text{INJ}[52:53]>_{neg} + \text{C53} \cdot 2^{-53} \tag{4.135}$$

$$= <(\text{C51}, \text{L}, \text{R})>_{neg} \tag{4.136}$$

$$< 2^{-50} \tag{4.137}$$

Thus, there can also be at most one carry bit from the mid part into position [51] of the upper part. This carry bit into position [51] is called C51 with $\text{C51} \iff (mpart \geq 2^{-51})$.

The value in the mid part depends on whether $fsum \in [1,2[$ or $fsum \in [2,4[$. Therefore, we compute two different versions of (C51, L, R), namely $(\text{C51}_{[1,2[}, \text{L}_{[1,2[}, \text{R}_{[1,2[})$ under the assumption that $fsum \in [1,2[$ $(\text{COND}_{[2,4[} = 0)$ and $(\text{C51}_{[2,4[}, \text{L}_{[2,4[}, \text{R}_{[2,4[})$ under the assumption that $fsum \in [2,4[$ $(\text{COND}_{[2,4[} = 1)$:

$$<(\text{C51}_{[1,2[}, \text{L}_{[1,2[}, \text{R}_{[1,2[})>_{neg} = <\text{SFOSUM}[52:53]>_{neg} + <\text{INJ}_{[1,2[}[52:53]>_{neg} + \text{C53} \cdot 2^{-53} \tag{4.138}$$

$$<(\text{C51}_{[2,4[}, \text{L}_{[2,4[}, \text{R}_{[2,4[})>_{neg} = <\text{SFOSUM}[52:53]>_{neg} + <\text{INJ}_{[2,4[}[52:53]>_{neg} + \text{C53} \cdot 2^{-53} \tag{4.139}$$

Moreover, the upper part of $finj$ in positions $[-1:51]$ can only have either the value

$$usum = <\text{USUM}[-1:51]>_{neg} \tag{4.140}$$

$$= <\text{SFOSUM}[-1:51]>_{neg} + <\text{CFOSUM}[-1:51]>_{neg} \tag{4.141}$$

or the value $usumi = <\text{USUMI}[-1:51]>_{neg} = usum + 2^{-51}$, because of equation 4.137. Based on this and with the definition of the rounding increment condition

$$\text{RINC} \iff \left((\text{C51}_{[2,4[} \wedge \text{COND}_{[2,4[}) \text{ OR } (\text{C51}_{[1,2[} \wedge \overline{\text{COND}_{[2,4[}})\right), \tag{4.142}$$

the required bits of the injected significand $\text{FINJ}[-1:52]$ can be selected by

$$\text{FINJ}[-1:51] = \begin{cases} \text{USUMI}[-1:51] & \text{if RINC} \\ \text{USUM}[-1:51] & \text{otherwise} \end{cases} \tag{4.143}$$

$$\text{FINJ}[52] = \begin{cases} \text{L}_{[2,4[} & \text{if } \text{COND}_{[2,4[} \\ \text{L}_{[1,2[} & \text{otherwise} \end{cases} \tag{4.144}$$

to prepare the injection-based rounding mode reduction for the rounding modes $mode \in \{RZ, RNU, RI, RMI\}$.

To implement the IEEE rounding mode RNE instead of RNU, we have to consider the 'L-bit fix' for the case of a tie according to section 2.3.2, namely, the least significant bit of the rounded significand has to be pulled down for the case, that the rounding operand lies exactly between two consequtive rounding choices in rounding mode RNE. We denote the condition, that an 'L-bit fix' is required by $\text{LFIX}_{[2,4[}$ for $\text{COND}_{[2,4[} = 1$ and by $\text{LFIX}_{[1,2[}$ for $\text{COND}_{[2,4[} = 0$ with

$$\text{LFIX}_{[2,4[} \iff (\text{FSUM}[52:115] = (1,0^{63})) \text{ AND } srmode = RNE \quad (4.145)$$

$$\text{LFIX}_{[1,2[} \iff (\text{FSUM}[53:115] = (1,0^{62})) \text{ AND } srmode = RNE \quad (4.146)$$

Thus, we get (substitution of eq. 4.143-4.146 and eq. 4.128 in eq. 4.124-4.127)

$$rnd_{srmode,51}(fsum) = <(\text{FINJ}[-1:50], \text{FINJ}[51] \wedge \overline{\text{LFIX}_{[2,4[}}, 0)>_{neg} \quad (4.147)$$

$$rnd_{srmode,52}(fsum) = <(\text{FINJ}[-1:51], \text{L}_{[1,2[} \wedge \overline{\text{LFIX}_{[1,2[}}) >_{neg}. \quad (4.148)$$

According to definition 4.6 the rounded significand $rnd\_fsum = <\text{RND\_FSUM}[-1:52]>_{neg}$ can be written by

$$\text{RND\_FSUM}[-1:52] = \begin{cases} (\text{FINJ}[-1:50], \text{FINJ}[51] \wedge \overline{\text{LFIX}_{[2,4[}}, 0) & \text{if } \text{COND}_{[2,4[} \\ (\text{FINJ}[-1:51], \text{L}_{[1,2[} \wedge \overline{\text{LFIX}_{[1,2[}}) & \text{otherwise.} \end{cases} \quad (4.149)$$

The following lemma provides the missing details for the implementation of the rounding decision.

Lemma 4.14 *Based on the definition of the sticky bit:*

$$\text{STICKY} = ORtree(\text{FOSUM}[54:115] \oplus \text{SEFF})$$

the signals C53, $\text{FSUM}[51:53]$, $\text{LFIX}_{[1,2[}$, $\text{LFIX}_{[2,4[}$, R_TINX *and* R_TINC *can be computed by:*

$$\begin{aligned}
\text{C53} &= (\text{SEFF} \wedge \overline{\text{STICKY}}) \vee ((\text{STICKY} \vee \text{SEFF}) \wedge (srmode = RI)) \\
<\text{FSUM}[51:53]>_{neg} &\equiv <\text{SFOSUM}[51:53]>_{neg} + <\text{CFOSUM}[51]>_{neg} + \\
&\quad +(\text{SEFF} \wedge \overline{\text{STICKY}}) \cdot 2^{-53} \mod 2^{-50} \\
\text{LFIX}_{[1,2[} &= \text{FSUM}[53] \wedge \overline{\text{STICKY}} \wedge (srmode = RNE) \\
\text{LFIX}_{[2,4[} &= \text{FSUM}[52] \wedge \overline{\text{FSUM}[53]} \wedge \overline{\text{STICKY}} \wedge (srmode = RNE) \\
\text{R_TINX} &= \text{STICKY} \vee ((\text{COND}_{[2,4[} \wedge OR(\text{FSUM}[52:53])) \vee (\overline{\text{COND}_{[2,4[}} \wedge \text{FSUM}[53])) \\
\text{R_TINC} &= (((\text{FINJ}[51] \wedge \overline{\text{LFIX}_{[2,4[}})\overline{\oplus}\text{FSUM}[51]) \wedge \text{COND}_{[2,4[}) \vee \\
&\quad \vee (((\text{L}_{[1,2[} \wedge \overline{\text{LFIX}_{[1,2[}})\overline{\oplus}\text{FSUM}[52]) \wedge \overline{\text{COND}_{[2,4[}})
\end{aligned}$$

Proof: We first show, that the sticky-bit has the property:

$$\overline{\text{STICKY}} \iff (\text{FSUM}[54:115] = 0^{62}) \quad (4.150)$$

$$\iff (fsum \text{ is integral multiple of } 2^{-53}). \quad (4.151)$$

To prove this, we distinguish the two cases: (a) SEFF = 0; and (b) SEFF = 1. (a) For SEFF = 0, we get $fosum = fsum$, so that

$$(\text{FSUM}[54:115] = 0^{62}) \iff (\text{FOSUM}[54:115] = 0^{62}) \iff \overline{\text{STICKY}}.$$

(b) For SEFF $= 1$, we have $fosum = fsum - 2^{-115}$, so that in this case (FSUM$[54:115] = 0^{62}$) $\Longleftrightarrow$ (FOSUM$[54:115] = 1^{62}$) $\Longleftrightarrow \overline{\text{STICKY}}$, as required. Moreover, we can immediately conclude from equation 4.150 that STICKY $\Longleftrightarrow$ ($<$FSUM$[54{:}115]>_{neg} \geq 2^{-115}$).

The carry bit C53 signals the condition ($lpart \geq 2^{-53}$). By definition

$$lpart \quad = \quad <\text{FOSUM}[54{:}115]>_{neg} + <\text{INJ}[54{:}115]>_{neg} + \text{SEFF} \cdot 2^{-115}.$$

The injection bits INJ$[54:115]$ can only be either (i) 1^{62} for $srmode = RI$ or (ii) 0^{62} otherwise. (i) If INJ$[54{:}115] = 1^{62}$, then

$$(lpart \geq 2^{-53}) \Longleftrightarrow ((\text{FSUM}[54{:}115] \neq 0^{62}) \vee \text{SEFF}) \Longleftrightarrow (\text{STICKY} \vee \text{SEFF}).$$

(ii) If INJ$[54{:}115] = 0^{62}$, then

$$(lpart \geq 2^{-53}) \Longleftrightarrow ((\text{FOSUM}[54{:}115] = 1^{62}) \wedge \text{SEFF}) \Longleftrightarrow \overline{\text{STICKY}} \wedge \text{SEFF},$$

as required.

In the equation for $<$FSUM$[51{:}53]>_{neg}$, the carry from the low part into position [53] without considering an injection (INJ$[54{:}115] = 0^{62}$) has to be used, namely, in this case $\text{C53}' = (\overline{\text{STICKY}} \wedge \text{SEFF})$. Thus, we get as required

$$\begin{aligned} <\text{FSUM}[51{:}53]>_{neg} \quad = \quad & <\text{SFOSUM}[51{:}53]>_{neg} + <\text{CFOSUM}[51]>_{neg} + \\ & +(\overline{\text{STICKY}} \wedge \text{SEFF}) \cdot 2^{-53} \bmod 2^{-50}. \end{aligned}$$

The equations for the 'L-bit'-fix conditons are the straight-forward implementation of their definition from equations 4.145-4.146 using $\overline{\text{STICKY}} \Longleftrightarrow$ (FSUM$[54{:}115] = 0^{62}$).

The inexactness rounding tag R_TINX (equation 2.56) can be written as

$$\text{R\_TINX} \quad = \quad \begin{cases} \text{OR}(\text{FSUM}[52{:}115]) & \text{if } \text{COND}_{[2,4[} \\ \text{OR}(\text{FSUM}[53{:}115]) & \text{otherwise.} \end{cases}$$

By the substitution of STICKY $\Longleftrightarrow$ (FSUM$[54{:}115] \neq 0^{62}$) $\Longleftrightarrow$ OR(FSUM$[54{:}115]$) we get as required

$$\text{R\_TINX} = \text{STICKY} \vee ((\text{COND}_{[2,4[} \wedge OR(\text{FSUM}[52{:}53])) \vee (\overline{\text{COND}_{[2,4[}} \wedge \text{FSUM}[53])).$$

According to equation 2.57 the increment rounding tag R_TINC can be written as

$$\text{R\_TINX} \quad = \quad \begin{cases} (\text{RND\_FSUM}[51] \neq \text{FSUM}[51]) & \text{if } \text{COND}_{[2,4[} \\ (\text{RND\_FSUM}[52] \neq \text{FSUM}[52]) & \text{otherwise.} \end{cases}$$

We get the required form of this equation by the substitution of RND_FSUM$[51]$ and RND_FSUM$[52]$ according to equation 4.149. □

Lemma 4.15 *In the rounding computations, the condition on the range of the significand sum* $\text{COND}_{[2,4[} \Longleftrightarrow (fsum \in [2,4[)$ *can be substituted by* USUM$[-1]$*, so that the rounding increment decision* RINC *is given by*

$$\text{RINC} \quad \Longleftrightarrow \quad \left((\text{C51}_{[2,4[} \wedge \text{USUM}[-1])\ OR\ (\text{C51}_{[1,2[} \wedge \overline{\text{USUM}[-1]})\right), \qquad (4.152)$$

the rounded significand RND_FSUM$[-1{:}52]$ *can be selected by*

$$\text{RND\_FSUM}[-1:52] \quad = \quad \begin{cases} (\text{FINJ}[-1{:}50], \text{FINJ}[51] \wedge \overline{\text{LFIX}_{[2,4[}}, 0) & \textit{if } \text{USUM}[-1] \\ (\text{FINJ}[-1{:}51], \text{L}_{[1,2[} \wedge \overline{\text{LFIX}_{[1,2[}}) & \textit{otherwise.} \end{cases} \qquad (4.153)$$

and the rounding tags TINC *and* TINX *can be computed according to*

$$\text{R\_TINX} = \text{STICKY} \vee ((\text{USUM}[-1] \wedge OR(\text{FSUM}[52:53])) \vee (\overline{\text{USUM}[-1]} \wedge \text{FSUM}[53]) \quad (4.154)$$

$$\text{R\_TINC} = (((\text{FINJ}[51] \wedge \overline{\text{LFIX}_{[2,4[}})\overline{\oplus}\text{FSUM}[51]) \wedge \text{USUM}[-1]) \vee \quad (4.155)$$

$$\vee (((\text{L}_{[1,2[} \wedge \overline{\text{LFIX}_{[1,2[}})\overline{\oplus}\text{FSUM}[52]) \wedge \overline{\text{USUM}[-1]}). \quad (4.156)$$

Proof: Because $usum+ < \text{SFOSUM}[52:115] >_{neg} + \text{SEFF} \cdot 2^{-115} = fsum$ and because $< \text{SFOSUM}[52:115] >_{neg} + \text{SEFF} \cdot 2^{-115} \leq 2^{-51}$, the values of $usum$ and $fsum$ differ at most by 2^{-51} with $fsum \geq usum$. Thus, the values $\text{USUM}[-1]$ and $\text{COND}_{[2,4[}$ differ, iff $\text{SEFF} = 1$, $\text{FSUM}[-1:51] = (1{,}0^{52})$, $\text{USUM}[-1:51] = (0{,}1^{52})$, $\text{USUMI}[-1:51] = (1{,}0^{52})$, and $\text{SFOSUM}[52:115] = 1^{64}$. In this situation, we have $\text{USUM}[-1] = 0$ and $\text{COND}_{[2,4[} = 1$. Moreover, it follows, that $mpart \geq 2^{-51}$, so that $\text{C51}_{[2,4[} = \text{C51}_{[1,2[} = 1$, and the incremented upper sum $< \text{USUMI}[-1:51] >_{neg} = (1{,}0^{52})$ is selected for both range conditions: $\text{USUM}[-1]$ and $\text{COND}_{[2,4[}$. Thus, it also does not matter which of them is chosen for the selection of $\text{RND\_FSUM}[-1:50]$ for the case that $\text{USUM}[-1] \neq \text{COND}_{[2,4[}$.

We still consider the case $\text{USUM}[-1] \neq \text{COND}_{[2,4[}$ in the following. Because $\text{RINC} = 1$ we have $\text{FINJ}[51] = \text{USUMI}[51] = 0$ and because of $\text{SFOSUM}[52:53] = 1^2$, $\text{C53} = 1$ and $\text{INJ}_{[1,2[}[52] = 0$, we get from equation 4.138, that $\text{L}_{[1,2[} = 0$. Thus, it follows from equation 4.149 that also the selection of $\text{RND\_FSUM}[51:52] = 0^2$ is independent of the value of $\text{COND}_{[2,4[}$ and $\text{USUM}[-1]$.

We still have to show, that we also get the same rounding tags for both range detections. From the above we know, that in the case which we have to consider, $\text{FSUM}[51:53] = 0^3$. Thus, according to the equation fro R_TINX from lemma 4.14 we get in this case $\text{R\_TINX} = \text{STICKY}$ independent of the value of $\text{COND}_{[2,4[}$ and $\text{USUM}[-1]$.

Because $inj_{[2,4[} < 2^{-51}$, $inj_{[1,2[} < 2^{-52}$ and $finj = fsum + inj_X$, we have $\text{FINJ}[51] = 0 = \text{L}_{[1,2[}$, so that according to the equation from lemma 4.14 also the rounding tag $\text{R\_TINC} = 0$ does not depend on the value of $\text{COND}_{[2,4[}$ and $\text{USUM}[-1]$ in this case.

Thus, as required, the substitutions of $\text{COND}_{[2,4[}$ by $\text{USUM}[-1]$ in the equations of this lemma do not change the results of these equations. □

The following lemma integrates the rounding computations according to equations 4.143-4.144,4.153 with the generalized post-normalization shift according to equation 4.122 to compute the final results of the 'R'-path:

Lemma 4.16 *In the 'R'-path, the significand and exponent bits are given by*

$$\text{R\_F}[0:51] = \begin{cases} \text{RND\_FSUM}[-1:50] & \textit{if } \text{RND\_FSUM}[-1] \\ \text{RND\_FSUM}[0:51] & \textit{otherwise.} \end{cases}$$

$$\text{R\_F}[52] = \begin{cases} \text{L}'(inc) = \text{USUMI}[51] \wedge \overline{\text{LFIX}_{[2,4[}} & \textit{if } \text{RND\_FSUM}[-1] \wedge \text{RINC} \\ \text{L}'(ninc) = \text{USUM}[51] \wedge \overline{\text{LFIX}_{[2,4[}} & \textit{if } \text{RND\_FSUM}[-1] \wedge \overline{\text{RINC}} \\ \text{L} = \text{L}_{[1,2[} \wedge \overline{\text{LFIX}_{[1,2[}} & \textit{if } \overline{\text{RND\_FSUM}[-1]} \end{cases}$$

$$< \text{R\_E}[11:0] >_2 = \begin{cases} < \text{E}_1[11:0] >_2 + 1 & \textit{if } \text{RND\_FSUM}[-1] \\ < \text{E}_1[11:0] >_2 & \textit{otherwise.} \end{cases}$$

Proof: Because $rnd\_fsum \in [2, 4[$, iff $\text{RND\_FSUM}[-1]$, the equation for $\text{R\_F}[0:51]$ and $\text{R\_E}[11:0]$ are straight-forward implementations of definition 4.6 and definition 4.5 using the previous rounding description from equations 4.143-4.144 and equation 4.153. Because $rnd\_fsum \geq usum$, it follows from $\text{RND\_FSUM}[-1] = 0$ that also $\text{USUM}[-1] = 0$ and, thus,

for the case of RND_FSUM$[-1] = 0$ we have RND_FSUM$[52] = (\mathrm{L}_{[1,2[} \wedge \overline{\mathrm{LFIX}_{[1,2[}})$. Therefore, according to equation 4.15

$$\mathrm{R\_F}[52] \quad = \quad \begin{cases} \mathrm{FINJ}[51] \wedge \overline{\mathrm{LFIX}_{[2,4[}} & \text{if } \mathrm{RND\_FSUM}[-1] \\ \mathrm{L}_{[1,2[} \wedge \overline{\mathrm{LFIX}_{[1,2[}} & \text{if } \overline{\mathrm{RND\_FSUM}[-1]}, \end{cases}$$

so that by the substitution of FINJ$[51]$ with respect to the value of RINC according to equation 4.143, we get the equation for R_F$[52]$ from the lemma. □

In the following we summarize the computation steps in the 'R'-path:

1.-2. computation of the limited absolute exponent difference *deltalim*, the sign of the exponent difference SDELTA and the operand swapping like in the previous section.

3. significand one's complement negation of fs for effective subtractions (equation 4.109):

$$\mathrm{FSO}[-1:52] = \mathrm{FS}[-1:52] \oplus \mathrm{SEFF}.$$

4. alignment shift of FSO$[-1:52]$ by *deltalim* positions (equation 4.113):

$$\begin{aligned} \mathrm{FSOA}[-1:115] &= (\mathrm{SEFF}^{deltalim}, \mathrm{FSO}[-1:52], \mathrm{SEFF}^{63-deltalim}) && (4.157) \\ &= \mathrm{RSFT}(\mathrm{FSO}[-1:52], deltalim, \mathrm{SEFF}, \mathrm{SEFF}) && (4.158) \end{aligned}$$

5. significand addition: (a) compression of positions $[-1:52]$ by a halfadder line

$$<\mathrm{SFOSUM}[-1:115]>_{neg} + <\mathrm{CFOSUM}[-1:51]>_{neg} = <\mathrm{FL}[-1:52]>_{neg} + <\mathrm{FSOA}[-1:115]>_{neg}$$

and (b) computation of the upper sum USUM$[-1:51]$ and incremented upper sum USUMI$[-1:51]$ by a compound adder (equation 4.141):

$$\begin{aligned} <\mathrm{USUM}[-1:51]>_{neg} &= <\mathrm{SFOSUM}[-1:51]>_{neg} + <\mathrm{CFOSUM}[-1:51]>_{neg} \\ <\mathrm{USUMI}[-1:51]>_{neg} &= <\mathrm{USUM}[-1:51]>_{neg} + 2^{-51} \end{aligned}$$

8. rounding decisions: computation of RINC, R_TINX, R_TINC, $\mathrm{L}'(ninc) = \mathrm{USUM}[51] \wedge \overline{\mathrm{LFIX}_{[2,4[}}$, $\mathrm{L}'(inc) = \mathrm{USUMI}[51] \wedge \overline{\mathrm{LFIX}_{[2,4[}}$ and $\mathrm{L} = \mathrm{L}_{[1,2[} \wedge \overline{\mathrm{LFIX}_{[1,2[}}$ in the rounding decision circuit according to lemma 4.15, lemma 4.14 and equations 4.138-4.139 from the inputs SFOSUM$[51:53]$, CFOSUM$[51]$, SEFF, SL, STICKY, RMODE$[1:0]$, USUM$[-1]$, USUM$[51]$ and USUMI$[51]$. This 'rounding decisions' circuit is depicted in detail in figure 4.14, only for 3 small parts in it, some additional explanations have to be given:

   - the 'INJ generation' circuit implements the rounding mode reduction (according to equation 2.6-2.6) and the generation of the injection bits: $\mathrm{INJ}_{[2,4[}[52] = \mathrm{INJ}_{[1,2[}[53] = \mathrm{OR}(\mathrm{SR\_MODE}[1:0])$, $\mathrm{INJ}_{[2,4[}[53] = \mathrm{SR\_MODE}[1]$ and $\mathrm{INJ}_{[1,2[}[52] = 0$.
   - the 'Carry lower part' circuit computes the carries from the lower part, C53 according to lemma 4.14 with $(srmode = RI) \Longleftrightarrow \mathrm{SR\_MODE}[1]$, and $\mathrm{C53}' = (\overline{\mathrm{STICKY}} \wedge \mathrm{SEFF})$.
   - the '$\overline{\mathrm{LFIX}}$' circuit implements the equations for $\overline{\mathrm{LFIX}_{[2,4[}}$ and $\overline{\mathrm{LFIX}_{[1,2[}}$ according to lemma 4.14 with $(srmode = RNE) \Longleftrightarrow \mathrm{SR\_MODE}[0]$.

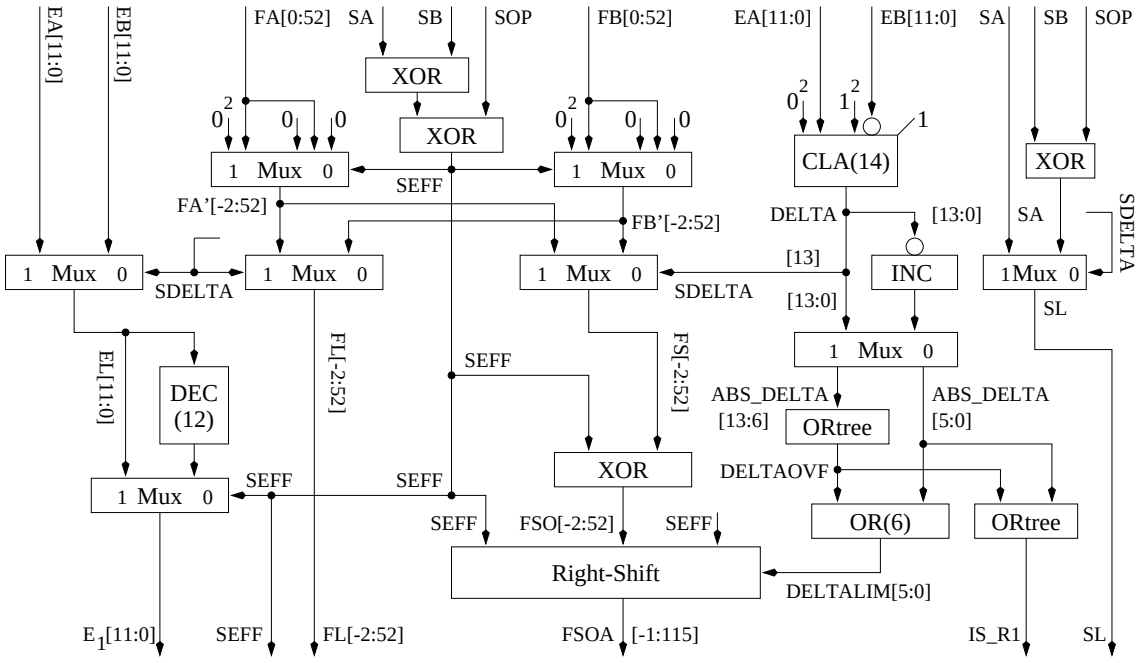

Figure 4.12: Structure of the 'R'-path (first part) of the addition/subtraction unit II.

rounding selections (equation 4.143 and lemma 4.16):

$$\text{FINJ}[-1:51] = \begin{cases} \text{USUMI}[-1:51] & \text{RINC} \\ \text{USUM}[-1:51] & \text{otherwise.} \end{cases}$$

$$\text{L}' = \begin{cases} \text{L}'(inc) & \text{RINC} \\ \text{L}'(ninc) & \text{otherwise.} \end{cases}$$

9. post-normalization shift of the rounded significand (lemma 4.16 and equation 4.153):

$$\text{R\_F}[0:52] = \begin{cases} (\text{FINJ}[-1:50], \text{L}') & \text{if RND\_FSUM}[-1] \\ (\text{FINJ}[0:51], \text{L}) & \text{otherwise.} \end{cases}$$

10. According to lemma 4.13, the sign of the 'R'-path is given by R_S = SL, which we compute like in the previous section. The exponent of the 'R'-path is based on $e_1 = <\text{E}_1[11:0]>_2$, which is computed like in the previous section, and on the selection according to lemma 4.16:

$$<\text{R\_E}[11:0]>_2 = \begin{cases} <\text{E}_1[11:0]>_2 + 1 & \text{if RND\_FSUM}[-1] \\ <\text{E}_1[11:0]>_2 & \text{otherwise.} \end{cases}$$

Moreover, during the computation of the exponent difference, we compute the condition IS_R1 $\iff (|ea - eb| > 1) \iff ORtree((\text{DELTAOVF}, \text{ABS\_DELTA}[5:1]))$. which will be used later for the selection of the valid path.

The implementation is depicted in two parts in figure 4.12 and 4.13. Additionaly, a more detailed block diagram of the 'Rounding decisions' circuit is shown in figure 4.14. This completes the description of the 'R'-path of the addition/subtractionII unit.

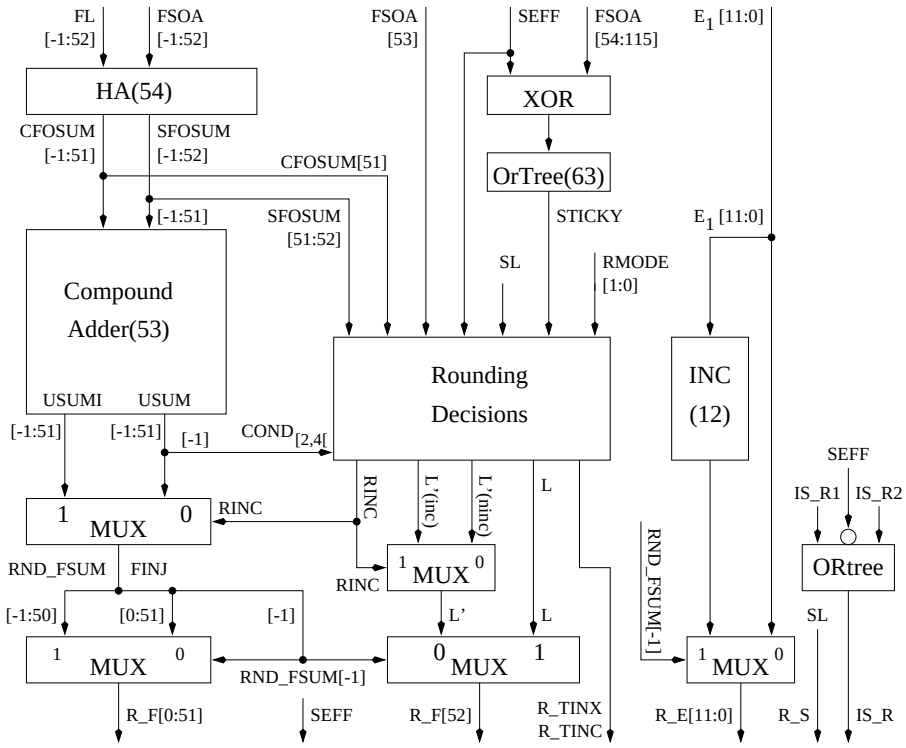

Figure 4.13: Structure of the 'R'-path (second part) of the addition/subtraction unit II.

'N'-path The computations in the 'N'-path are described on the basis of the adder implementation from the previous section, which is optimized regarding the specific properties of the 'N'-path. As discussed above (see lemma 4.12(b), (c)), we can use for the 'N'-path: SEFF $= 1$ and $\delta \in \{-1, 0, 1\}$. Additionaly, because of lemma 4.12(d) the gradual rounding has no effect, so that the required factoring in the 'N'-path according to equation 4.106 and definition 4.3 can be written as

$$((n\_s, n\_e, n\_f), \text{N\_TINC}, \text{N\_TINX}) \quad = \quad (\eta(\text{SL} \oplus \text{SFSUM}, e_1, abs\_fsum), 00). \qquad (4.159)$$

Because $\delta \in \{-1, 0, 1\}$, the exponent difference can be represented by the two bits DELTA[1:0] with $<\text{DELTA}[1:0]>_2 = \delta = ea - eb = <\text{EA}[1:0]>_2 + <\overline{\text{EB}[1:0]}>_2 + 1$, where the two bits DELTA[1:0] can already be interpreted as the sign-magnitude representation of δ, so that $abs\_delta = \text{DELTA}[0]$ and SDELTA $= \text{DELTA}[1]$.

Because for $\delta \in \{-1, 0, 1\}$, the bit combination DELTA[1:0] $= 10$ can not occur, the alignment shift can be integrated with the swapping and the unconditional pre-shift into the following selections:

$$\text{FSOA}[-2:52] \quad = \quad \begin{cases} (1, \overline{\text{FB}[0:52]}, 1) & \text{if } \overline{\text{DELTA}[1]} \text{ AND } \overline{\text{DELTA}[0]} \\ (11, \overline{\text{FB}[0:52]}) & \text{if } \overline{\text{DELTA}[1]} \text{ AND DELTA}[0] \\ (11, \overline{\text{FA}[0:52]}) & \text{if DELTA}[1]. \end{cases}$$

$$(\text{SL}, \text{EL}[11:0], \text{FL}[-2:52]) \quad = \quad \begin{cases} (\text{SB}, \text{EB}[11:0], 0, \text{FB}[0:52], 0) & \text{if DELTA}[1] \\ (\text{SA}, \text{EA}[11:0], 0, \text{FA}[0:52], 0) & \text{otherwise.} \end{cases}$$

Thus, $<\text{FSOA}[-2:52]>_{2neg} = <\text{FSA}[-2:52]>_{2neg} - 2^{-52}$, and $<\text{FOSUM}[-2:52]>_{2neg} = <\text{FSUM}[-2:52]>_{2neg} - 2^{-52}$.

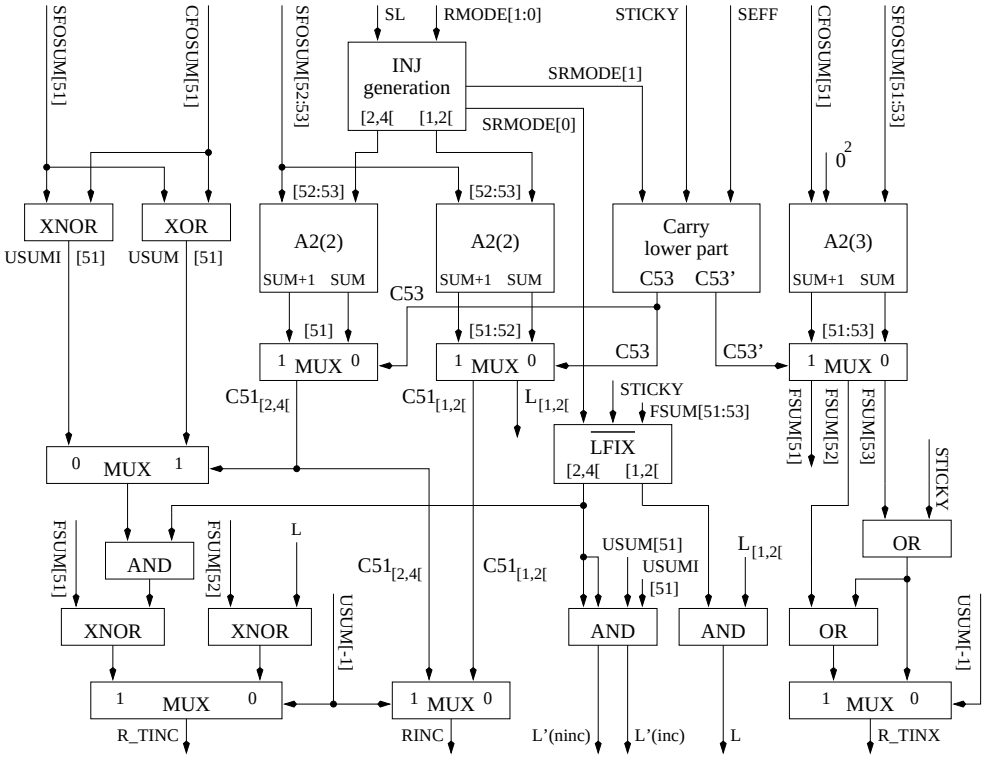

Figure 4.14: Implementations of the 'rounding decision circuit' in the 'R'-path of the addition/subtraction unit II.

Because of $<\overline{\text{FOSUM}[-2:52]}>_{2neg} = -<\text{FOSUM}[-2:52]>_{2neg} + 2^{-52} = -fsum$, we can get the binary representation of the absolute significand sum $abs\_fsum = <\text{ABS\_FSUM}[-1:52]>_{neg}$ with a compound adder, that computes the sum $fosum = <\text{FOSUM}[-2:52]>_{2neg}$ and the incremented sum $fosumi = <\text{FOSUMI}[-2:52]>_{2neg} = fosum + 2^{-52}$ by the selection

$$\text{ABS\_FSUM}[-1:52] \quad = \quad \begin{cases} \overline{\text{FOSUM}[-1:52]} & \text{if FOSUMI}[-2] \\ \text{FOSUMI}[-1:52] & \text{otherwise.} \end{cases}$$

In this way, we get the factoring $(\text{SL}\oplus\text{FOSUMI}[-2], <\text{EL}[11:0]>_2 -1, <\text{ABS\_FSUM}[-1:52]>_{neg})$, which already has the value of the 'N'-path-result, but according to equation 4.120, we still have to compute an unbounded normalization shift on this factoring.

Definition 4.7 *We define the term of an* imprecise normalized factoring *for factorings* $(s_{ipn}, e_{ipn}, f_{ipn})$, *whose significand fullfills the condition* $f_{ipn} \in [1, 4[$. *An operation ipnorm, that computes an imprecise normalized factoring* $(s_{ipn}, e_{ipn}, f_{ipn}) = ipnorm(s, e, f)$ *with* $val(s_{ipn}, e_{ipn}, f_{ipn}) = val(s, e, f)$ *for an arbitrary non-zero factoring* (s, e, f), *is called* imprecise normalization shift. *Note, that if* lz *is the shift distance of an unbounded normalization shift, then an imprecise normalization shift uses one of the shift distances* $\{lz, lz+1\}$.

Obviously, an unbounded normalization shift η *can be partitioned into an imprecise normalization shift followed by a generalized post-normalization shift:*

$$\eta(s, e, f) = gpost(ipnorm(s, e, f)).$$

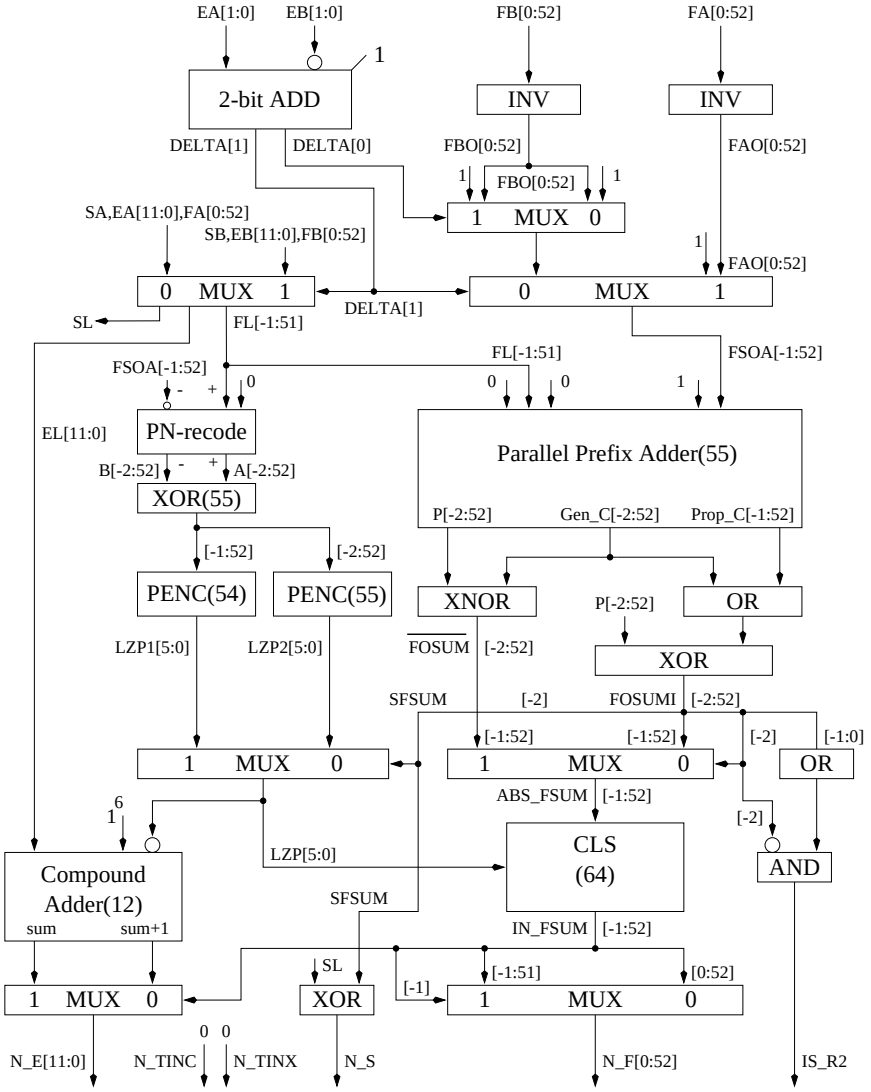

Figure 4.15: Structure of the 'N'-path in the addition/subtraction unit II.

Like suggested in the previous definition, we will partition the computation of the normalization shift into a first step of an imprecise normalization shift followed by a second step of a generalized post-normalization shift. The advantage of this approach is, that the shift-amout for the imprecise normalization shift can be already determined from the carry-save representation of the significand sum in parallel to the significand addition, so that we can save the delay of the slow serial leading-zero computation after the significand addition which was used in the previous section.

The generalized post-normalization shift is computed like in the 'R'-path. The only difference between the implementation of the imprecise normalization shift and the conventional normalization shift from the previous section is the computation of the shift-amout. Therefore, we will focus on the description of the shift-amount computation in the following. For this purpose we require some notations and techniques from [8]. We summarize them in the next definitions and lemmas in preparation for our leading-zero estimation.

Definition 4.8 *In a Borrow-Save representation, a number is represented by two binary strings: we call the tupel* $(\mathrm{A}[n_1:n_2], \mathrm{B}[n_1:n_2])$ *with a positively weigted bit string* $\mathrm{A}[n_1:n_2]$ *and a negatively weigted bit string* $\mathrm{B}[n_1:n_2]$ *a* Borrow-Save *representation of the number c, iff* $c = <\mathrm{A}[n_1:n_2]>_{neg} - <\mathrm{B}[n_1:n_2]>_{neg}$. *To annote that* A *is the positively weighted bit-string and* B *is the negatively weighted bit string, we also write* A^+ *and* B^-. *The digits* $\gamma[i] = \mathrm{A}[i] - \mathrm{B}[i] \in \{-1, 0, 1\}$ *are called* Borrow-Save digits *and we denote the value of a string of Borrow-Save digits* $\gamma[n_1:n_2] \in \{-1,0,1\}^{n_2-n_1}$ *by* $<\gamma[n_1:n_2]>_{bs} = c$. *We also write:*

$$c = <\gamma[n_1:n_2]>_{bs} = \left\langle \begin{array}{c} \mathrm{A}^+[n_1:n_2] \\ \mathrm{B}^+[n_1:n_2] \end{array} \right\rangle_{bs} = \left\langle \left(\begin{array}{c} \mathrm{A}[n_1], \mathrm{A}[n_1+1], \cdots, \mathrm{A}[n_2] \\ \mathrm{B}[n_1], \mathrm{B}[n_1+1], \cdots, \mathrm{B}[n_2] \end{array} \right) \right\rangle_{bs}.$$

For $\lambda \in [n_1 : n_2]$, *the* fraction *of a Borrow-Save representation* $\gamma[n_1:n_2] \in \{-1,0,1\}^{n_2-n_1}$ *at position* λ *is defined by:*

$$\begin{aligned} fract_\lambda(\gamma[n_1:n_2]) &= 2^\lambda \cdot \sum_{j=\lambda+1}^{n_2} \gamma[j] \cdot 2^{-j} \\ &= \gamma[\lambda{+}1] \cdot 2^{-1} + \gamma[\lambda{+}2] \cdot 2^{-2} + \cdots + \gamma[n_2] \cdot 2^{-n_2+\lambda}. \end{aligned}$$

The fraction range *of a Borrow-Save representation* $\gamma[n_1:n_2] \in \{-1,0,1\}^{n_2-n_1}$ *is defined by the interval* $FRANGE(\gamma[n_1:n_2]) = [a,b]$, *with* $a = min\{fract_\lambda(\gamma[n_1:n_2]) | \lambda \in [n_1 : n_2]\}$ *and* $b = max\{fract_\lambda(\gamma[n_1:n_2]) | \lambda \in [n_1 : n_2]\}$. *Obviously, for arbitrary Borrow-Save representations* $\gamma[n_1:n_2] \in \{-1,0,1\}^{n_2-n_1}$, *we have* $FRANGE(\gamma[n_1:n_2]) \subset]-1, 1[$.

In the following definition we introduce the 'P'-carry and the 'N'-carry-recoding that will be used for the compression of the fraction range in our leading-zero estimation.

Definition 4.9 *The 'P'-carry-recoding computes from a Borrow-Save representation* $B = (\mathrm{A}[n_1:n_2], \mathrm{B}[n_1:n_2])$, *the Borrow-Save representation* $B' = P(B) = (\mathrm{A}'[n_1-1:n_2], \mathrm{B}'[n_1-1:n_2])$, *where for all* $\lambda \in [n_1:n_2]$:

$$\textit{Carry: } \mathrm{A}'[\lambda{+}1] = \mathrm{A}[\lambda] \wedge \overline{\mathrm{B}[\lambda]} \qquad \textit{Residual: } \mathrm{B}'[\lambda] = \mathrm{A}[\lambda] \oplus \mathrm{B}[\lambda].$$

The 'N'-carry-recoding computes from a Borrow-Save representation $B = (\mathrm{A}[n_1:n_2], \mathrm{B}[n_1:n_2])$, *the Borrow-Save representation* $B' = N(B) = (\mathrm{A}'[n_1-1:n_2], \mathrm{B}'[n_1-1:n_2])$ *where for all* $\lambda \in [n_1:n_2]$:

$$\textit{Carry: } \mathrm{B}'[\lambda{+}1] = \overline{\mathrm{A}[\lambda]} \wedge \mathrm{B}[\lambda] \qquad \textit{Residual: } \mathrm{A}'[\lambda] = \mathrm{A}[\lambda] \oplus \mathrm{B}[\lambda].$$

The following lemma shows some properties of 'P'-carry- and 'N'-carry-recodings:

Lemma 4.17 *This lemma consists of 4 parts:*

(a) Both 'P'-carry- and 'N'-carry-recoding do not change the value of a Borrow-Save representation, namely: $< B >_{bs} = < P(B) >_{bs} = < N(B) >_{bs}$.

(b) The 'P'-carry-recoding compresses the fraction range $FRANGE(B) \subset]a, b[$ *of a Borrow-Save representation* B *to* $FRANGE(P(B)) \subset]-1/2 + a/2, b/2[$.

(c) The 'N'-carry-recoding compresses the fraction range $FRANGE(B) \subset]a, b[$ *of a Borrow-Save representation* B *to* $FRANGE(N(B)) \subset]a/2, 1/2 + b/2[$.

(d) 'PN'-recoding reduces the fraction range of an arbitrary Borrow-Save representation B *to* $FRANGE(N(P(B))) \subset]-3/4, 1/2[$.

Proof: (a) There are only 4 possible bit combinations for the Borrow-Save digit at position λ by the two bits $\mathrm{A}[\lambda]$ and $\mathrm{B}[\lambda]$. These bit combinations encode the 3 possible values of a Borrow-Save digit like summarized in table 4.1. After 'P'-carry-recoding, this Borrow-Save digit is represented by the carry $\mathrm{A}'[\lambda - 1]$ and the residual $\mathrm{B}'[\lambda]$, and we can read off from table 4.1, that the 'P'-recoding equations exactly fullfill the equation $\mathrm{A}[\lambda] - \mathrm{B}[\lambda] = 2\mathrm{A}'[\lambda - 1] - \mathrm{B}'[\lambda]$, so that $< B >_{bs} = < P(B) >_{bs}$. Accordingly, the 'N'-carry-recoding represents the two bits $\mathrm{A}[\lambda]$ and $\mathrm{B}[\lambda]$ by the carry $\mathrm{B}'[\lambda - 1]$ and the residual $\mathrm{A}'[\lambda]$ and implements the equation $\mathrm{A}[\lambda] - \mathrm{B}[\lambda] = \mathrm{A}'[\lambda] - 2\mathrm{B}'[\lambda - 1]$ (see table 4.1), so that also $< B >_{bs} = < N(B) >_{bs}$.

(b) With the BS representations B and $P(B)$:

$$B = \begin{pmatrix} \mathrm{A}[n_1], \mathrm{A}[n_1+1], \cdots, \mathrm{A}[n_2] \\ \mathrm{B}[n_1], \mathrm{B}[n_1+1], \cdots, \mathrm{B}[n_2] \end{pmatrix} \quad \text{and} \quad P(B) = \begin{pmatrix} \mathrm{A}'[n_1-1], & \mathrm{A}'[n_1], & \cdots, & \mathrm{A}'[n_2] \\ 0, & \mathrm{B}'[n_1], & \cdots, & \mathrm{B}'[n_2] \end{pmatrix},$$

we obtain by extracting a term from the radix polynomial of $P(B)$ for $\lambda \in [n_1 : n_2]$:

$$\begin{aligned} fract_\lambda(P(B)) &= fract_\lambda \begin{pmatrix} \mathrm{A}'[\lambda+1], & \mathrm{A}'[\lambda+2], & \cdots, & \mathrm{A}'[n_2] \\ \mathrm{B}'[\lambda+1], & \mathrm{B}'[\lambda+2], & \cdots, & \mathrm{B}'[n_2] \end{pmatrix} \\ &= -\mathrm{B}'[\lambda+1] \cdot 2^{-1} + fract_\lambda \begin{pmatrix} \mathrm{A}'[\lambda+1], & \mathrm{A}'[\lambda+2], & \cdots, & \mathrm{A}'[n_2] \\ 0, & \mathrm{B}'[\lambda+2], & \cdots, & \mathrm{B}'[n_2] \end{pmatrix} \\ &= -\mathrm{B}'[\lambda+1] \cdot 2^{-1} + fract_\lambda \begin{pmatrix} 0, & \mathrm{A}[\lambda+2], & \cdots, & \mathrm{A}[n_2] \\ 0, & \mathrm{B}[\lambda+2], & \cdots, & \mathrm{B}[n_2] \end{pmatrix} \\ &= -\mathrm{B}'[\lambda+1] \cdot 2^{-1} + \frac{1}{2} fract_{\lambda+1} \begin{pmatrix} \mathrm{A}[\lambda+2], & \mathrm{A}[\lambda+3], & \cdots, & \mathrm{A}[n_2] \\ \mathrm{B}[\lambda+2], & \mathrm{B}[\lambda+3], & \cdots, & \mathrm{B}[n_2] \end{pmatrix}. \\ &= -\mathrm{B}'[\lambda+1] \cdot 2^{-1} + \frac{1}{2} fract_{\lambda+1}(B) \end{aligned}$$

Since $-\mathrm{B}'[\lambda + 1] \cdot 2^{-1} \in \{-\frac{1}{2}, 0\}$ and $fract_{\lambda+1}(B) \subset]a, b[$, we obtain $fract_\lambda(P(B)) \in]-1/2 + a/2, b/2[$ for all $\lambda \in [n_1 : n_2]$, so that $FRANGE(P(B)) \subset]-1/2 + a/2, b/2[$, as required. Part (c) can be proven in analogy to part (b).

(d) Starting from the fraction range $FRANGE(B) \subset]-1, 1[$ of an arbitrary Borrow-Save-representation B, the use of part (b) and part (c) of this lemma directly yields $FRANGE(P(N(B))) \subset]-3/4, 1/2[$, as required. □

The following lemma describes the application of 'PN'-recoding for the imprecize normalization shift of the 'N'-path.

Borrow-Save representation			'P'-carry-recoding		'N'-carry-recoding	
$A^+[\lambda]$	$B^-[\lambda]$	$A^+[\lambda] - B^-[\lambda] = \gamma[\lambda]$	$A'[\lambda - 1]$	$B'[\lambda]$	$B'[\lambda - 1]$	$A'[\lambda]$
0	1	-1	0	1	1	1
0	0	0	0	0	0	0
1	1	0	0	0	0	0
1	0	1	1	1	0	1

Table 4.1: Summary of the cases in the 'P'-carry and the 'N'-carry-recoding.

Lemma 4.18 *With the computation of*

$$\begin{aligned}
\gamma_{fsum}[-4{:}52] = \begin{pmatrix} A^+_{fsum}[-4{:}52] \\ B^-_{fsum}[-4{:}52] \end{pmatrix} &= P\left(N\left(\begin{array}{c} \mathrm{FL}[-2{:}52] \\ NOT(\mathrm{FSOA}[-2{:}52]) \end{array}\right)\right) \\
\mathrm{LZP1}[5:0] &= \mathrm{PENC}\left(A^+_{fsum}[-1{:}52] \oplus B^-_{fsum}[-1{:}52]\right) \\
\mathrm{LZP2}[5:0] &= \mathrm{PENC}\left(A^+_{fsum}[-2{:}52] \oplus B^-_{fsum}[-2{:}52]\right) \\
\mathrm{LZP}[5:0] &= \begin{cases} \mathrm{LZP1}[5:0] & \textit{if } \mathrm{SFSUM} \\ \mathrm{LZP2}[5:0] & \textit{otherwise.} \end{cases} \\
\mathrm{IN_FSUM}[-1{:}52] &= \mathrm{CLS}(\mathrm{ABS_FSUM}[-1{:}52], <\mathrm{LZP}[5{:}0]>_2) \\
<\mathrm{IN_E}[11{:}0]>_2 &= <\mathrm{EL}[11{:}0]>_2 + <(111111, \overline{\mathrm{LZP}[5{:}0]}>_2
\end{aligned}$$

we get the imprecisely normalized factoring

$$(n\_s, in\_e, in\_fsum) = (\mathrm{N}\_s, <\mathrm{IN\_E}[11{:}0]>_2, <\mathrm{IN\_FSUM}[-1{:}52]>_{neg}),$$

so that $val(n\_s, in\_e, in\_fsum) = val(n\_s, <\mathrm{EL}[11:0]>_2 - 1, <\mathrm{ABS\_FSUM}[-1{:}52]>_{neg})$ *and* $in\_fsum \in [1, 4[$.

Proof: Because $fsum = <\mathrm{FL}[-2{:}52]>_{neg} - <\overline{\mathrm{FSOA}[-2{:}52]}>_{neg}$ and because of lemma 4.17 (a), we get $<\gamma_{fsum}>_{bs} = fsum$. For all non-zero $fsum \neq 0$, the borrow-save representation $\gamma_{fsum}[-4{:}52]$ includes at least one non-zero digit, so that for a $k \in [-4{:}52]$, it has the form $\gamma_{fsum}[-4{:}52] = (0^{k+4}, \gamma_{fsum}[k{:}52])$ with $\gamma_{fsum}[k] \in \{-1, 1\}$.

By lemma 4.17(d) we obtain the fraction range $FRANGE(\gamma_{fsum}[-4{:}52]) \subset]-3/4, 1/2[$. For the determination of the range of ABS_FSUM from $\gamma_{fsum}[k{:}52]$ and the fraction range, we differ between the two cases: (a) $\gamma_{fsum}[k] = 1$; and (b) $\gamma_{fsum}[k] = -1$.

(a) From $\gamma_{fsum}[k] = 1$ and $fract_k(\gamma_{fsum}[k+1:52]) \in]-3/4, 1/2[$, it follows, that $fract_{k-1}(\gamma_{fsum}[k{:}52]) \in]1/2 - 3/8, 1/2 + 1/4[=]1/8, 3/4[$. Moreover, the fraction range is also valid for the fraction at position $k-1$, so that $fract_{k-1}(\gamma_{fsum}[k{:}52]) \in]1/8, 3/4[\cap]-3/4, 1/2[=]1/8, 1/2[$. Thus, $2^{k+2} \cdot abs\_fsum \in]1, 4[$. According to lemma 4.12, we assume in the 'N'-path, that $fsum < 1$. Thus, we define $lzp2 = (k+2)$ and get for case (a), $lzp2 = (k+2) \geq 0$.

(b) Correspondingly, for $\gamma_{fsum}[k] = -1$, we get $fract_{k-1}(\gamma_{fsum}[k{:}52]) \in]-\frac{1}{2} - \frac{3}{8}, -\frac{1}{2} + \frac{1}{4}[\cap]-\frac{3}{4}, \frac{1}{2}[=]-\frac{3}{4}, -\frac{1}{4}[$. Thus, $2^{k+1} \cdot abs\_fsum \in]1, 3[$. For this case we define $lzp1 = (k+1)$, so that we can derive from $abs\_fsum < 2$, that also this leading zero prediction has to be non-negative $lzp1 = (k+1) \geq 0$.

Because the representation of a Borrow-Save digit by $\mathrm{A}^+[\lambda]$ and $\mathrm{B}^-[\lambda]$ is non-zero, iff $\mathrm{A}^+[\lambda] \oplus \mathrm{B}^-[\lambda]$ and $lzp1, lzp2 \geq 0$, the number $lzp2 = k + 2$ can be interpreted as the number of leading zeros in the string $\gamma_{fsum}[-2{:}52]$, so that

$$lzp2 = <\mathrm{LZP2}[5:0]> = <\mathrm{PENC}(\mathrm{A}^+[-2{:}52] \oplus \mathrm{B}^-[-2{:}52])>.$$

Accordingly, the number $lzp1 = lzp2 - 1 = k + 1$ can be recognized as the number of leading zeros in the string $\gamma_{fsum}[-1{:}52]$, so that

$$lzp1 = <\mathrm{LZP1}[5:0]> = k + 1 = <\mathrm{PENC}(\mathrm{A}^+[-1{:}52] \oplus \mathrm{B}^-[-1{:}52])>.$$

Obviously, the above case (a) occurs, iff $fsum > 0$ and the above case (b) occurs, iff $fsum < 0$. Because in case (a), $2^{lzp2} \cdot abs\_fsum \in]1, 4[$, $lzp2 \geq 0$ and $abs\_fsum = <\mathrm{ABS\_FSUM}[-1{:}52]>_{neg}$, the significand $in\_fsum = 2^{lzp2} \cdot abs\_fsum$ is imprecisely normalized and can be represented by $\mathrm{IN\_FSUM}[-1:52]$ and the multiplication of abs_fsum by 2^{lzp2} can be implemented by a left-shift of $\mathrm{ABS\_FSUM}[-1:52]$ by $lzp2$ positions. Accordingly, in case (b) the significand $in\_fsum = 2^{lzp1} \cdot abs\_fsum$ is imprecisely normalized and can be represented by $\mathrm{IN\_FSUM}[-1:52]$ and the multiplication of abs_fsum by 2^{lzp1} can be implemented by a left-shift of $\mathrm{ABS\_FSUM}[-1{:}52]$ by $lzp1$ positions.

The definition of the leading zero prediction lzp selects either the $lzp2$ for case (a) and $lzp1$ for case (b), so that the left-shift of $\mathrm{ABS\_FSUM}[-1:52]$ by lzp positions exactly computes the binary representation of the imprecisely normalized significand $in\_fsum = <\mathrm{IN\_FSUM}[-1{:}52]>_{neg}$. In this way $in\_fsum = abs\_fsum \cdot 2^{lzp}$.

The term lzp is adjusted in the exponent by $in\_e = <\mathrm{IN\_E}[11{:}0]>_2 = el - 1 - lzp$, so that $in\_fsum \cdot 2^{in\_e} = abs\_fsum \cdot 2^{el-1}$, as required by the lemma. □

Based on the results of this lemma, the 'N'-path result is computed from SL, SFSUM, $\mathrm{IN\_E}[11{:}0]$, and $\mathrm{IN\_FSUM}[-1{:}52]$ by the final generalized post-normalization shift:

$$\begin{aligned}(n\_s, n\_e, n\_f) &= post\_norm(\mathrm{SL} \oplus \mathrm{SFSUM}, <\mathrm{IN\_E}[11{:}0]>_2, <\mathrm{IN\_FSUM}[-1{:}52]>_{neg})) \\ &= \begin{cases} (\mathrm{SL} \oplus \mathrm{SFSUM}, <\mathrm{IN\_E}[11{:}0]>_2 + 1, <\mathrm{IN\_FSUM}[-1{:}51]>_{neg}/2) & \text{if } \mathrm{IN\_FSUM}[-1] \\ (\mathrm{SL} \oplus \mathrm{SFSUM}, <\mathrm{IN\_E}[11{:}0]>_2, <\mathrm{IN\_FSUM}[0{:}52]>_{neg}) & \text{otherwise.} \end{cases}\end{aligned}$$

The incremented exponent is already precomputed with a compound adder during the exponent adjustment from lemma 4.18, so that the post-normalization shift can be realized by a simple selection depending on the value of $\mathrm{IN\_FSUM}[-1]$. Additionaly, we compute the condition $\mathrm{IS\_R2} = \overline{\mathrm{FOSUMI}[-2]} \wedge (\mathrm{FOSUMI}[-1] \vee \mathrm{FOSUMI}[0])$, that will be used for the path selection. This completes the description of the 'N'-path, a block diagram of which is depicted in figure 4.15.

Path selection In the following we explain how the general path selection condition IS_R is computed from the signals IS_R1, IS_R2 and SEFF. We start from the definition of the path selection condition according to lemma 4.3:

$$\begin{aligned}\mathrm{IS\_R} &\iff \overline{\mathrm{SEFF}} \vee (fsum \in [1, 4)) \\ &\iff \overline{\mathrm{SEFF}} \vee (fsum \in [1, 4) \text{ under the assumption } \mathrm{SEFF} = 1)\end{aligned}$$

Because for $\mathrm{IS\_R} = 0$, we have $abs\_delta \leq 1$ according to lemma 4.12, which is signaled by $\overline{\mathrm{IS\_R1}}$, the above equation can be further extended to:

$$\mathrm{IS\_R} \iff \overline{\mathrm{SEFF}} \vee \mathrm{IS\_R1} \vee \left(fsum \in [1, 4) \text{ under the assumption } \overline{\mathrm{SEFF}} \text{ and } \overline{\mathrm{IS\_R1}}\right)$$

Because the assumptions SEFF = 1 and $\overline{\text{IS\_R1}}$ are exactly the assumptions, that we use during the computation of $fsum$ in the 'N'-path, the condition IS_R2 exactly implements the expression $\left(fsum \in [1,4)\right.$ under the assumption SEFF = 1 and $\left.\overline{\text{IS\_R1}}\right)$, so that

$$\text{IS\_R} \quad = \quad \overline{\text{SEFF}} \vee \text{IS\_R1} \vee \text{IS\_R2} \tag{4.160}$$

The condition IS_R1 and the signal SEFF are computed in the 'R'-path and the condition IS_R2 is computed in the 'N'-path. These three parts of the path selection condition are combined in the 'R'-path like depicted in figure 4.13, so that the result of the valid path can be selected by the combined path selection condition IS_R like depicted in figure 4.11. This completes the description of the addition/subtractionII unit.

4.2.3 Addition/Subtraction III (normalized $\longrightarrow$ normalized format)

Like in the two previous sections also in this section the FP addition/subtraction is computed from the inputs of the normalized representations $BUSa_{NF}[69:0]$ and $BUSb_{NF}[69:0]$ (section 2.6.3), the rounding mode represented by RMODE[1:0] and the bit SOP that signals the case of addition or subtraction. But in contrast to the previous implementations, where a representative of the exact operation result or a gradual rounded result had to be delivered, in this case the addition unit III already has to compute the IEEE factoring representation of the rounded result in the normalized format.

Formally, with the notation from equation 2.16 and lemma 2.8 and with

$$(s_{nrc}, e_{nrc}, f_{nrc}) = nround_{mode}(s_{rc}, e_{rc} + wec, f_{rc}) \tag{4.161}$$

$$= exp\_rnd_{mode \star s_{rc}}(\eta(n\_sig\_rnd_{mode \star s_{rc}}(\eta(s_{rc}, e_{rc} + wec, f_{rc})))) \tag{4.162}$$

the required addition result is based on the following NF factoring

$$(s_{NF}, e_{NF}, f_{NF}) = \begin{cases} (0, e_{qNaN}, f_{qNaN}) & \text{if SCQNAN} \\ (s_{inf}, e_{\infty}, f_{\infty}) & \text{if SCINF} \\ (sa, ea, fa) & \text{if SCX} \\ (sb, eb, fb) & \text{if SCY} \\ (s_0, e_0, 0) & \text{if SCZERO} \\ (s_{nrc}, e_{nrc}, f_{nrc}) & \text{otherwise.} \end{cases} \tag{4.163}$$

so that the sum output of the addition/subtraction unit III is specified by the corresponding representation in the normalized format $BUS_{NF}[70:0] = \text{NF}(s_{NF}, e_{NF}, f_{NF})$. To compute the exponent wrapping, the inputs of the trap handler enable bits UNF_EN and OVF_EN are required. Moreover, the occurance of an invalid, inexact, overflow and underflow exception should be signaled by the bit INV, INX, OVF and UNF, respectively.

The computation of the special value results according to equation 4.163 is implemented like in the previous section. The only difference is that in this section an infinity result might also be generated in the regular case because of the exponent rounding. Thus, the special case condition for an infinity result from the special cases circuit SCINF is only valid for SPCA = 1. We denote this condition by INF_{sc} = SCINF in this section. Accordingly, we define the condition INF_{nrc} that signals the case of an infinity result for the regular case $\text{INF}_{nrc} \Longleftrightarrow (val(s_{nrc}, e_{nrc}, f_{nrc}) = \pm\infty)$, so that we get the final infinity flag by

$$\text{INF}_{NF} \quad = \quad \begin{cases} \text{INF}_{sc} & \text{if SPCA} \\ \text{INF}_{nrc} & \text{otherwise.} \end{cases} \tag{4.164}$$

For the regular case the computations of $(s_{nrc}, e_{nrc}, f_{nrc})$ have to be modified in comparison to the computations of $(s_{grc}, e_{grc}, f_{grc})$ from the previous section. The difference in these computations is that in this section we already have to consider single step rounding at the final rounding position vp while integrating the cases for single precision and double precision. Moreover, we also have to consider the exponent wrapping and the exponent rounding.

We base the computation of $(s_{nrc}, e_{nrc}, f_{nrc})$ on the two-path addition algorithm from the previous section with the path selection condition IS_R. In this section we denote the factoring output of the R-path (IS_R $= 1$) by (r_sn, r_en, r_fn) and the factoring output of the N-path (IS_R $= 0$) by (n_sn, n_en, n_fn), so that the factoring for the regular case is selected by

$$(s_{nrc}, e_{nrc}, f_{nrc}) = \begin{cases} (r\_sn, r\_en, r\_fn) & \text{if IS\_R} \\ (n\_sn, n\_en, n\_fn) & \text{otherwise.} \end{cases}$$

Moreover, the exceptions are detected seperately for the two paths by (N_INX, N_UNF, N_OVF) for IS_R $= 0$ and by (R_INX, R_UNF, R_OVF) for IS_R $= 1$, so that in general the occurance of the inexact, the underflow and the overflow exception are signaled by

$$(\text{INX, UNF, OVF}) = \begin{cases} (\text{R\_INX, R\_UNF, R\_OVF}) & \text{if IS\_R} \\ (\text{N\_INX, N\_UNF, N\_OVF}) & \text{otherwise.} \end{cases}$$

In this way the main structure of the implementation in this section (see figure 4.16) is very similar to that from the previous section (see figure 4.11). In the following we describe the details of the implementation of the R-path and the implementation of the N-path separately.

N-path In the N-path we have to compute the representation of the factoring

$$(n\_sn, n\_en, n\_fn) = exp\_rnd_{mode \star \mathrm{S}_{rc}}(\eta(n\_sig\_rnd_{mode \star s_{rc}}(\eta(s_{rc}, e_{rc}+wec, f_{rc})))),$$

where we can use the condition that IS_R $= 0$. We partition the discussion of these computations into two steps: the first step with the computation of

$$(n\_sn1, n\_en1, n\_fn1) \quad = \quad \eta(n\_sig\_rnd_{mode \star s_{rc}}(\eta(s_{rc}, e_{rc}, f_{rc}))), \tag{4.165}$$

so that we get the final result of the N-path by the second computation step of

$$(n\_sn, n\_en, n\_fn) \quad = \quad exp\_rnd_{mode \star \mathrm{S}_{rc}}(n\_sn1, n\_en1 + wec, n\_fn1). \tag{4.166}$$

In the first computation step, the rounding function $n\_sig\_rnd_{mode \star s_{rc}}$ differs from the rounding function $sgrnd1_{mode \star s_{rc}}$ and the second unbounded normalization shift differs from the post-normalization shift in the previous section. The two rounding functions only differ by the rounding position, which is $vp = (p-1) - max\{0, e_{min} - e'_{rc}\}$ in this case. In the previous section, the rounding position was 52 and it was shown, that this rounding function does not have any effect in the N-path. We will show in the following lemma, that also the significand rounding at the variable rounding position vp can be neglected in the N-path. In this way the rounding output is still normalized from the first normalization shift, so that also the second normalization shift is not required. Thus, all computations for the first step (equation 4.165) are already considered by the N-path implementation from the previous section:

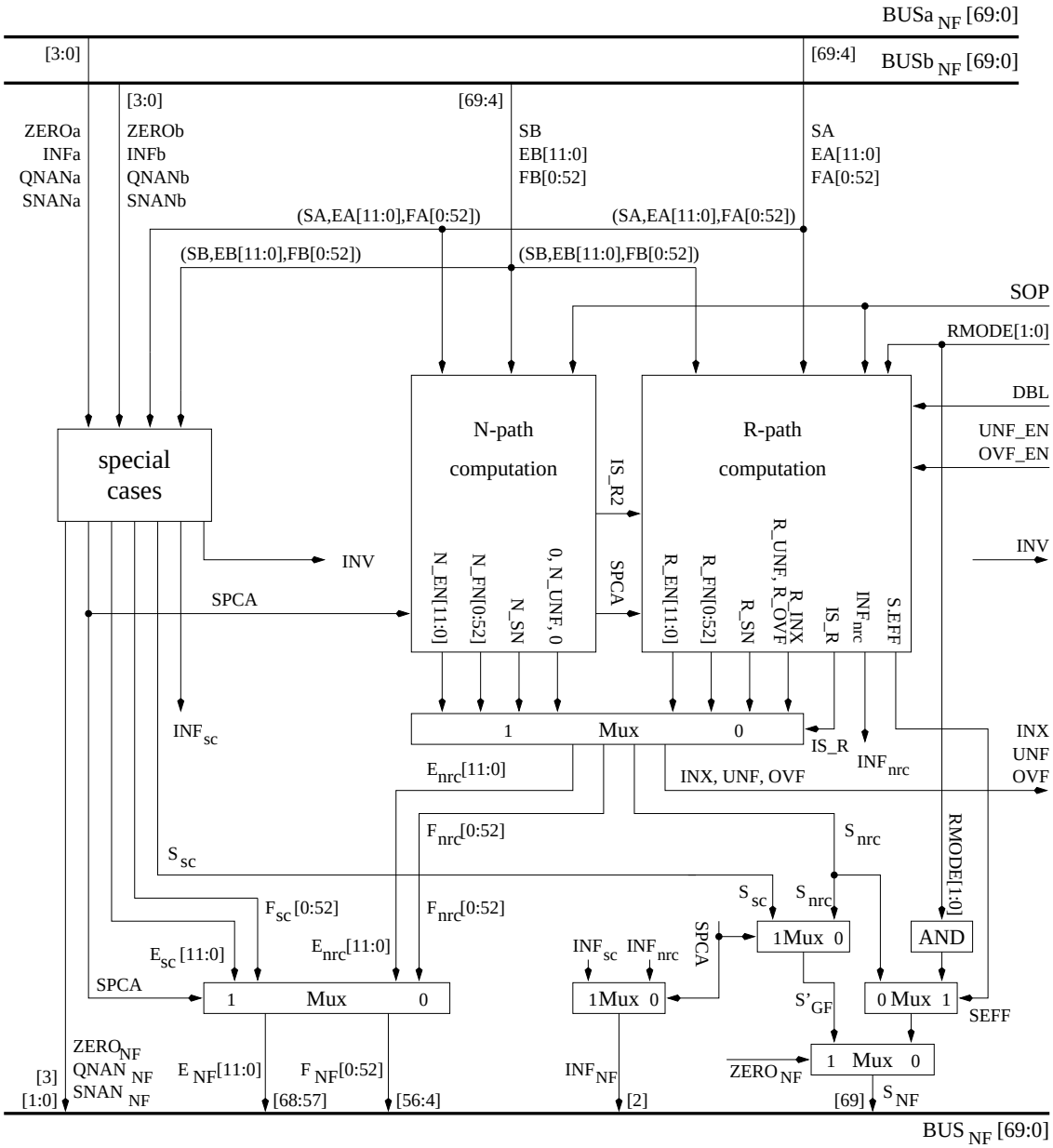

Figure 4.16: Structure of the addition/subtraction unit III.

Lemma 4.19 *(a) For a fixed precision with fixed p and e_{min}, all exact addition/subtraction results are integral multiples of $2^{e_{min}-p+1}$.*

(b) If the significand rounding position is variable, namely $vp \neq p-1$, then no rounding computation is required in the addition/subtraction implementation.

(c) In the N-path of the adder III, no implementation of rounding or the second normalization shift is required, so that for $\text{IS\_R} = 0$ *we have*

$$(n\_sn1, n\_en1, n\_fn1) = (n\_s, n\_e, n\_f) = \eta(s_{rc}, e_{rc}, f_{rc}) = \eta(\text{SL} \oplus \text{SFSUM}, e_1, abs\_fsum).$$

Proof: (a) Because for a fixed p and e_{min}, all operands are integral multiples of $2^{e_{min}-p+1}$, also the exact sum and the exact difference of these operands are integral multiples of $2^{e_{min}-p+1}$.

(b) The rounding position vp satisfies $vp \neq p-1$, iff the exponent e'_{rc} of the rounding operand is smaller than e_{min}. In this case, the weight of the rounding position vp is $2^{e'_{rc}-p+1} < 2^{e_{min}-p+1}$. Because from (a) it follows, that all exact addition/subtraction results are integral multiples of this rounding position weight, the rounding computation has no effect for the case that $vp \neq p-1$.

(c) We first show, that there is no rounding computation required in the N-path. For this proof we distinguish between the cases of single precision and double precision and the cases that the rounding position fulfills either $vp = p-1$ or $vp \neq p-1$: For $vp \neq p-1$ the proof already follows from part (b). For double precision and $vp = p-1$, we have $vp = 52$, so that the setting is like in the previous section and it follows from lemma 4.12(d) that no rounding computation is required in the N-path in this case. For single precision with $p = 24$ and $vp = p-1 = 23$, the proof of lemma 4.12(d) could be adopted accordingly, so that also for this case, no rounding computations in the N-path are required. Thus, the rounding computations can be neglected in the N-path for all cases. In this way the input of the second normalization shift in equation 4.165 is still normalized, so that even the second normalization shift can be neglected in equation 4.165. In this way we have exactly the same situation like in the N-path of the previous section, so that we get as required $(n\_sn1, n\_en1, n\_fn1) = \eta(s_{rc}, e_{rc}, f_{rc}) = \eta(\text{SL} \oplus \text{SFSUM}, e_1, abs\_fsum) = (n\_s, n\_e, n\_f)$. □

Thus, for the computation of $(n\_sn1, n\_en1, n\_fn1) = (n\_s, n\_e, n\_f)$ (equation 4.165), the N-path implementation from the previous section is used. The computation of the second part according to equation 4.166, requires the detection of the overflow and the underflow exception. In the following lemma we consider both, the exception detections and the implementation of the second step (equation 4.166) to compute the factoring (n_sn, n_en, n_fn).

Lemma 4.20 *In the N-path:*

(a) no overflow can occur: $\text{N\_OVF} = 0$.

(b) exponent rounding has no effect., so that $(n\_sn, n\_en, n\_fn) = (n\_s, n\_e + wec, n\_f)$.

(c) all results are exact and an inexact exception can not occur: $\text{N\_INX} = 0$.

(d) with the computation of $<\text{N\_TT}[11:0]>_2 = <\text{N\_E}[11:0]>_2 + <(0, \text{DBL}^3, 1^6, 0)>_2$, *the underflow exception can be detected by:* $\text{N\_UNF} \Longleftrightarrow (\text{N\_TT}[11] \wedge \text{UNF\_EN} \wedge \overline{\text{SPCA}})$.

(e) $wec = \begin{cases} +\alpha = <\text{ALPHA}[11:7]>_2 = <(0, \text{DBL}^2, 0, \overline{\text{DBL}}^2)>_2 & \textit{if } \text{N\_UNF} \\ 0 & \textit{otherwise.} \end{cases}$

Proof: (a) Because we consider non-zero representable input operands in the N-path, the exponent of the "larger" operand el is smaller than or equal to e_{max}. Because in the N-path we have $fsum < 1$, all results in the N-path have values smaller than $2^{e_{max}}$, so that no overflow can occur in the N-path and $\text{N\_OVF} = 0$.

(b) Because all results in the N-path have values smaller than $2^{e_{max}}$, the exponent rounding in equation 4.166 becomes the identity function, so that we get the result of the N-path by $(n\_sn, n\_en, n\_fn) = (n\_s, n\_e + wec, n\_f)$, as required.

(c) Because both the significand rounding and the exponent rounding do not change the value of the result, all results in the N-path are exact, so that $\text{N\_INX} = 0$.

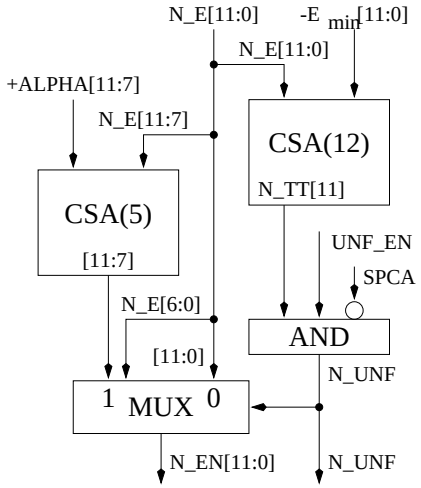

Figure 4.17: Additional circuits for the N-path of the addition/subtraction unit III.

(d) Because the factoring (n_s, n_e, n_f) is normalized with $n\_f \in [1, 2[$, the value of the result $val(n\_s, n\_e, n\_f)$ is tiny, iff $n\_e < e_{min}$. From (c) and the definition of the underflow exception in section 2.4.1 it then follows, that $\text{N\_UNF} \Longleftrightarrow ((n\_e < e_{min}) \wedge \text{UNF\_EN} \wedge \overline{\text{SPCA}})$. Because $-e_{min} = <(0, \text{DBL}^3, 1^6, 0)>_2$, we have $(n\_e < e_{min}) \Longleftrightarrow \text{N\_TT}[11]$, so that the above equation for N_UNF can be written as $\text{N\_UNF} \Longleftrightarrow (\text{N\_TT}[11] \wedge \text{UNF\_EN} \wedge \overline{\text{SPCA}})$.

(e) From (c) it follows, that $\text{N\_UNF} \Longleftrightarrow \text{N\_UNF} \wedge \text{UNF\_EN}$. Because no overflow can occur in the N-path according to (a), the definition of the exponent wrapping constant from equation 2.14 becomes

$$wec = \begin{cases} +\alpha = <\text{ALPHA}[11:7]>_2 = <(0, \text{DBL}^2, 0, \overline{\text{DBL}}^2)>_2 & \text{if N\_UNF} \\ 0 & \text{otherwise.} \end{cases}$$

□

Thus, in the second computation step of the N-path only the underflow exception has to be detected according to part (b) and the exponent wrapping $n\_en = n\_e + wec$ has to be computed according to part (b) and (e). The implementation of these extensions for the N-path are depicted in figure 4.17. This completes the description of the N-path for the adder III unit.

R-path In the R-path we have to compute the representation of the NF-factoring

$$(r\_sn, r\_en, r\_fn) = exp\_rnd_{mode \star \mathrm{S}_{rc}}(\eta(n\_sig\_rnd_{mode \star s_{rc}}(\eta(s_{rc}, e_{rc}+wec, f_{rc})))),$$

where we can use the condition that IS_R = 1. Like in the description of the N-path, we also partition the discussion of the R-path computations into two steps: the first step with the computation of

$$(r\_sn1, r\_en1, r\_fn1) \quad = \quad \eta(n\_sig\_rnd_{mode \star s_{rc}}(\eta(s_{rc}, e_{rc}, f_{rc}))), \tag{4.167}$$

so that we get the final R-path result by the second computation step with

$$(r\_sn, r\_en, r\_fn) \quad = \quad exp\_rnd_{mode \star \mathrm{S}_{rc}}(r\_sn1, r\_en1 + wec, r\_fn1). \tag{4.168}$$

First, we deal with the computations for equation 4.167. This formula for the computation of (r_sn1, r_en1, r_fn1) and the formula for (r_s, r_e, r_f) in the R-path of the previous section (equation 4.120) differ for the significand rounding functions and the second normalization shift. Let $(s'_{rc}, e'_{rc}, f'_{rc}) = \eta(s_{rc}, e_{rc}, f_{rc})$. With the rounding position $vp = (p-1) - max\{0, e_{min} - e'_{rc}\}$, the above factorings can be written as (see rounding function definitions 2.16 and 2.9):

$$(r\_sn1, r\_en1, r\_fn1) \quad = \quad \eta(s'_{rc}, e'_{rc}, rnd_{mode \star \mathrm{SL}, vp}(f'_{rc})) \tag{4.169}$$

$$(r\_s, r\_e, r\_f) \quad = \quad post\_norm(s'_{rc}, e'_{rc}, rnd_{mode \star \mathrm{SL}, 52}(f'_{rc})) \tag{4.170}$$

Thus, the only difference of the significand rounding functions are the rounding positions: in this section significand rounding at the variable significand position vp is considered, while in the previous section significand rounding at the fixed rounding position 52 was computed.

The following lemma shows, that for the addition/subtraction implementation, the variable rounding position vp can be substituted by vp', a fixed rounding position for single precision and a fixed rounding position for double precision, so that the rounding implementation from the previous section could be adopted either for the single precision or for the double precision case. As we know from the previous section, that the post-normalization shift in equation 4.170 already normalizes the rounded factoring and we have a similar rounding computation in equation 4.169, also in equation 4.169 a post-normalization shift will be sufficient to normalize the result instead of an unbounded normalization shift:

Lemma 4.21 *In the addition/subtraction implementation, the variable rounding position* $vp = (p-1) - max\{0, e_{min} - e'_{rc}\}$ *can be substituted by* $vp' = \begin{cases} 52 & \textit{if DBL} \\ 23 & \textit{otherwise} \end{cases}$ *without changing the rounded result, so that*

$$(r\_sn1, r\_en1, r\_fn1) \quad = \quad \eta(s'_{rc}, e'_{rc}, rnd_{mode \star \mathrm{SL}, vp'}(f'_{rc})) \tag{4.171}$$

$$= \quad \begin{cases} post\_norm(s'_{rc}, e'_{rc}, rnd_{mode \star \mathrm{SL}, 52}(f'_{rc})) & \textit{if DBL} \\ post\_norm(s'_{rc}, e'_{rc}, rnd_{mode \star \mathrm{SL}, 23}(f'_{rc})) & \textit{otherwise.} \end{cases} \tag{4.172}$$

Proof: According to lemma 4.19(b), no rounding computation is required in the adder implementation for $vp \neq p-1$. Therefore, and because $vp \leq p-1$, we always could set the rounding position to $p-1$ without changing the rounding result. The integration of the case for single precision ($p-1 = 23$) and double precision ($p-1 = 52$) exactly

yields the rounding position vp'. With the rounding computation at the fixed position vp', which is the least significant bit position for single and double precision, also the rounding result $rnd_{mode\star\mathrm{SL},vp'}(f'_{rc})$ is in the range $[1,2]$, so that a post-normalization shift and an unbounded normalization shift have the same effect on the rounded factoring, and we can replace η by $post\_norm$ in the lemma. □

Based on this lemma we could apply the implementation of the R-path from the previous section for the computation of $(r\_sn1, r\_en1, r\_fn1) = (r\_s, r\_e, r\_f)$ in the double precision case. Thus, we get for double precision according to lemma 4.13:

$$(r\_sn1, r\_en1, r\_fn1) = \begin{cases} gpost\_norm(\mathrm{SL}, e_1, rnd_{mode\star\mathrm{SL},52}(fsum)) & \text{if } fsum \in [1,2[\wedge \mathrm{DBL} \\ gpost\_norm(\mathrm{SL}, e_1, rnd_{mode\star\mathrm{SL},51}(fsum)) & \text{if } fsum \in [2,4[\wedge \mathrm{DBL} \end{cases}$$

Accordingly, lemma 4.13 could also be adopted for $p-1 = 23$, so that the single precision case could be integrated and we get

$$(r\_sn1, r\_en1, r\_fn1) = \begin{cases} gpost\_norm(\mathrm{SL}, e_1, rnd_{mode\star\mathrm{SL},52}(fsum)) & \text{if } fsum \in [1,2[\wedge \mathrm{DBL} \\ gpost\_norm(\mathrm{SL}, e_1, rnd_{mode\star\mathrm{SL},51}(fsum)) & \text{if } fsum \in [2,4[\wedge \mathrm{DBL} \\ gpost\_norm(\mathrm{SL}, e_1, rnd_{mode\star\mathrm{SL},23}(fsum)) & \text{if } fsum \in [1,2[\wedge \overline{\mathrm{DBL}} \\ gpost\_norm(\mathrm{SL}, e_1, rnd_{mode\star\mathrm{SL},22}(fsum)) & \text{if } fsum \in [2,4[\wedge \overline{\mathrm{DBL}} \end{cases}$$

The sign and the exponent are constant in the four choices, so that for the following discussion we isolate the significand computation by

$$r\_frnd = \begin{cases} rnd_{mode\star\mathrm{SL},52}(fsum) & \text{if } fsum \in [1,2[\wedge \mathrm{DBL} \\ rnd_{mode\star\mathrm{SL},51}(fsum) & \text{if } fsum \in [2,4[\wedge \mathrm{DBL} \\ rnd_{mode\star\mathrm{SL},23}(fsum) & \text{if } fsum \in [1,2[\wedge \overline{\mathrm{DBL}} \\ rnd_{mode\star\mathrm{SL},22}(fsum) & \text{if } fsum \in [2,4[\wedge \overline{\mathrm{DBL}} \end{cases}$$

and have then to compute $(r\_sn1, r\_en1, r\_fn1) = gpost\_norm(\mathrm{SL}, e_1, r\_frnd)$.

Because the injection based rounding reduction only depends on the rounding position and not on the value of the rounding operand, we align the rounding positions for single precision and double precision by

$$r\_frnd \;=\; \begin{cases} rnd_{mode\star\mathrm{SL},52}(fsum) & \text{if } fsum \in [1,2[\wedge \mathrm{DBL} \\ rnd_{mode\star\mathrm{SL},51}(fsum) & \text{if } fsum \in [2,4[\wedge \mathrm{DBL} \\ 2^{29} \cdot rnd_{mode\star\mathrm{SL},52}(2^{-29} \cdot fsum) & \text{if } fsum \in [1,2[\wedge \overline{\mathrm{DBL}} \\ 2^{29} \cdot rnd_{mode\star\mathrm{SL},51}(2^{-29} \cdot fsum) & \text{if } fsum \in [2,4[\wedge \overline{\mathrm{DBL}} \end{cases} \tag{4.173}$$

In the implementation, the multiplication of the rounding operand by 2^{-29} in the case of single precision is achieved by a conditional left-shift of the representations of both input significands by 29 positions for $\mathrm{DBL} = 0$. We denote these aligned operands by

$$faq =< \mathrm{FAQ}[0{:}52] >_{neg} \;=\; \begin{cases} fa =< \mathrm{FA}[0{:}52] >_{neg} & \text{if DBL} \\ 2^{-29} \cdot fa =< (0^{29}, \mathrm{FA}[0{:}23]) >_{neg} & \text{otherwise.} \end{cases} \tag{4.174}$$

$$fbq =< \mathrm{FBQ}[0{:}52] >_{neg} \;=\; \begin{cases} fb =< \mathrm{FB}[0{:}52] >_{neg} & \text{if DBL} \\ 2^{-29} \cdot fb =< (0^{29}, \mathrm{FB}[0{:}23]) >_{neg} & \text{otherwise.} \end{cases} \tag{4.175}$$

Accordingly, we indicate all corresponding values that are computed from faq and fbq instead of fa and fb by appending a 'q' to their name. With this notation and with the

inputs of FAQ[0:52] and FBQ[0:52], the R-path implementation from the previous section computes

$$fsumq = <\text{FSUMQ}[-1:115]>_{neg} \tag{4.176}$$

$$= \begin{cases} fsum = <\text{FSUM}[-1:115]>_{neg} & \text{if DBL} \\ 2^{-29} \cdot fsum = <(0^{29}, \text{FSUM}[-1:86])>_{neg} & \text{otherwise.} \end{cases} \tag{4.177}$$

$$usumq = <\text{USUMQ}[-1:51]>_{neg} \tag{4.178}$$

$$= \begin{cases} usum = <\text{USUM}[-1:51]>_{neg} & \text{if DBL} \\ 2^{-29} \cdot usum = <(0^{29}, \text{USUM}[-1:22])>_{neg} & \text{otherwise.} \end{cases} \tag{4.179}$$

In equation 4.179 the signal USUM[−1], which substitutes the condition $fsum \in [2,4[$ according to lemma 4.15, is shifted to position USUMQ[28] for single precision. Thus, the signal USUM[−1] can be selected by

$$\text{USUM}[-1] = \text{CONDQ}_{[2,4[} = \begin{cases} \text{USUMQ}[-1] & \text{if DBL} \\ \text{USUMQ}[28] & \text{otherwise.} \end{cases} \tag{4.180}$$

With the substitution of $\text{CONDQ}_{[2,4[}$ for the condition $fsum \in [2,4[$, equation 4.173 can be written as

$$r\_frnd = \begin{cases} rnd_{mode\star \text{SL},52}(fsumq) & \text{if } \overline{\text{CONDQ}_{[2,4[}} \wedge \text{DBL} \\ rnd_{mode\star \text{SL},51}(fsumq) & \text{if } \text{CONDQ}_{[2,4[} \wedge \text{DBL} \\ 2^{29} \cdot rnd_{mode\star \text{SL},52}(fsumq) & \text{if } \overline{\text{CONDQ}_{[2,4[}} \wedge \overline{\text{DBL}} \\ 2^{29} \cdot rnd_{mode\star \text{SL},51}(fsumq) & \text{if } \text{CONDQ}_{[2,4[} \wedge \overline{\text{DBL}} \end{cases} \tag{4.181}$$

If the condition $\text{COND}_{[2,4[} \iff fsum \in [2,4[$ is also substituted by the bit $\text{CONDQ}_{[2,4[}$ in the modified rounding computations from the previous section, then this R-path implementation computes

$$rnd\_fsumq = <\text{RND\_FSUMQ}[-1:52]>_{neg} \tag{4.182}$$

$$= \begin{cases} rnd_{mode\star \text{SL},52}(fsumq) & \text{if } \overline{\text{CONDQ}_{[2,4[}} \\ rnd_{mode\star \text{SL},51}(fsumq) & \text{if } \text{CONDQ}_{[2,4[}. \end{cases} \tag{4.183}$$

Obviously, the rounded significand r_frnd can then be computed by a conditional left shift of RND_FSUMQ[−1:52] by 29 positions for the case of single precision (see eq. 4.181)

$$r\_frnd = <\text{R\_FRND}[-1:52]>_{neg} \tag{4.184}$$

$$= \begin{cases} rnd\_fsumq = <\text{RND\_FSUMQ}[-1:52]>_{neg} & \text{if DBL} \\ 2^{29} \cdot rnd\_fsumq = <(\text{RND\_FSUMQ}[28:52], 0^{29})>_{neg} & \text{otherwise.} \end{cases} \tag{4.185}$$

Although, this is already an equation for the required significand r_frnd, we would like to postpone the re-alignment-shift by 29 positions in this equation after the computation of the generalized post-normalization shift. One can easily read off from equation 4.185, that the rounded significand r_frnd is in the range $[2,4[$ for the *post-normalization* condition

$$\text{PSCOND} \iff (r\_frnd \in [2,4[) \tag{4.186}$$

$$\iff (\text{RND\_FSUMQ}[-1] \wedge \text{DBL}) \vee (\text{RND\_FSUMQ}[28] \wedge \overline{\text{DBL}}). \tag{4.187}$$

Thus, we get for the generalized post-normalization shift,

$$(r\_sn1, r\_en1, r\_fn1) = gpost\_norm(\text{SL}, e_1, r\_frnd) \tag{4.188}$$

$$= \begin{cases} (\text{SL}, e_1 + 1, r\_frnd/2) & \text{if PSCOND} \\ (\text{SL}, e_1, r\_frnd) & \text{otherwise.} \end{cases} \tag{4.189}$$

With the preliminary significand r_fq

$$r\_fq = <\text{R\_FQ}[0:52]>_{neg} \tag{4.190}$$

$$= \begin{cases} rnd\_fsumq = <\text{RND\_FSUMQ}[0:52]>_{neg} & \text{if PSCOND} \\ rnd\_fsumq/2 = <\text{RND\_FSUMQ}[-1:51]>_{neg} & \text{otherwise,} \end{cases} \tag{4.191}$$

the significand for the factoring (r_sn1, r_en1, r_fn1) can be computed by the following re-alignment selection

$$r\_fn1 = <\text{R\_FN1}[0:52]>_{neg} \tag{4.192}$$

$$= \begin{cases} r\_fq = <\text{R\_FQ}[0:52]>_{neg} & \text{if DBL} \\ 2^{29} \cdot r\_fq = <(\text{R\_FQ}[29:52], 0^{29})>_{neg} & \text{otherwise,} \end{cases} \tag{4.193}$$

Note, that equation 4.191 describes the generalized post-normalization shift of the modified R-path implementation from the previous section, where only the control signal RND_FSUM$[-1]$ is substituted by the post-normalization condition PSCOND. In this way the computation of the factoring (r_sn1, r_en1, r_fn1) on the basis of the modified R-path implementation from the previous section requires only the five additional circuits according to the equations 4.174, 4.175, 4.180, 4.187 and 4.193. The integration of these additional circuits around the R-path implementation from the previous section is depicted in figure 4.18, where the additional circuits are represented by shaded boxes. This completes the description of first step in the R-path computations according to equation 4.167.

In the following we consider the second step of the R-path computations according to equation 4.168. This includes the detection of the exceptions, the exponent rounding and the exponent wrapping.

Lemma 4.22 *With the computation of*

$$<\text{R\_TT}[11:0]>_2 = <\text{E}_1[11:0]>_2 + <(0, \text{DBL}^3, 1^6, 0)>_2$$
$$<\text{R\_TTI}[11:0]>_2 = <\text{R\_TT}[11:0]>_2 + 1$$

the exceptions in the R-path can be detected by:

$$\text{R\_OVF} \iff (\text{ZEROTEST}(\text{E}_1[11:0] \oplus (0, \text{DBL}^3, 1^7)) \wedge \overline{\text{SPCA}} \wedge \text{PSCOND})$$
$$\text{R\_INX} \iff (\text{R\_TINXQ} \vee \text{R\_OVF})$$
$$\text{R\_UNF} \iff \begin{cases} \text{R\_TTI}[11] \wedge (\text{UNF\_EN} \vee \text{R\_INX}) \wedge \overline{\text{SPCA}} & \textit{if PSCOND} \\ \text{R\_TT}[11] \wedge (\text{UNF\_EN} \vee \text{R\_INX}) \wedge \overline{\text{SPCA}} & \textit{otherwise} \end{cases}$$

Proof: The condition for an overflow in the R-path is given by

$$\text{R\_OVF} \iff (|val(r\_sn1, r\_en1, r\_fn1)| \geq 2^{e_{max}+1}).$$

Because of the normalized significand $r\_fn1 \in [1, 2[$, this overflow condition can be written as R_OVF $\iff (r\_en1 \geq e_{max} + 1)$. Because in the R-path we only have to consider non-zero representable operands, the exponent of the 'larger' operand el and also e_1 are smaller than or equal to e_{max}. Thus, according to equation 4.189, the exponent of the normalized factoring r_en1 can only become larger than e_{max}, if $r\_en1 = ei_1 = e_{max} + 1$. Thus, R_OVF $\iff (ei_1 = e_{max} + 1) \wedge$ PSCOND $\wedge \overline{\text{SPCA}}$. Because $(ei_1 = e_{max} + 1) \iff (e_1 = e_{max})$ and $e_{max} =< (0, \text{DBL}^3, 1^7) >_2$, the equation for R_OVF can be written as R_OVF $\iff$ (ZEROTEST(E$_1[11:0] \oplus (0, \text{DBL}^3, 1^7)) \wedge$ PSCOND $\wedge \overline{\text{SPCA}}$, as required.

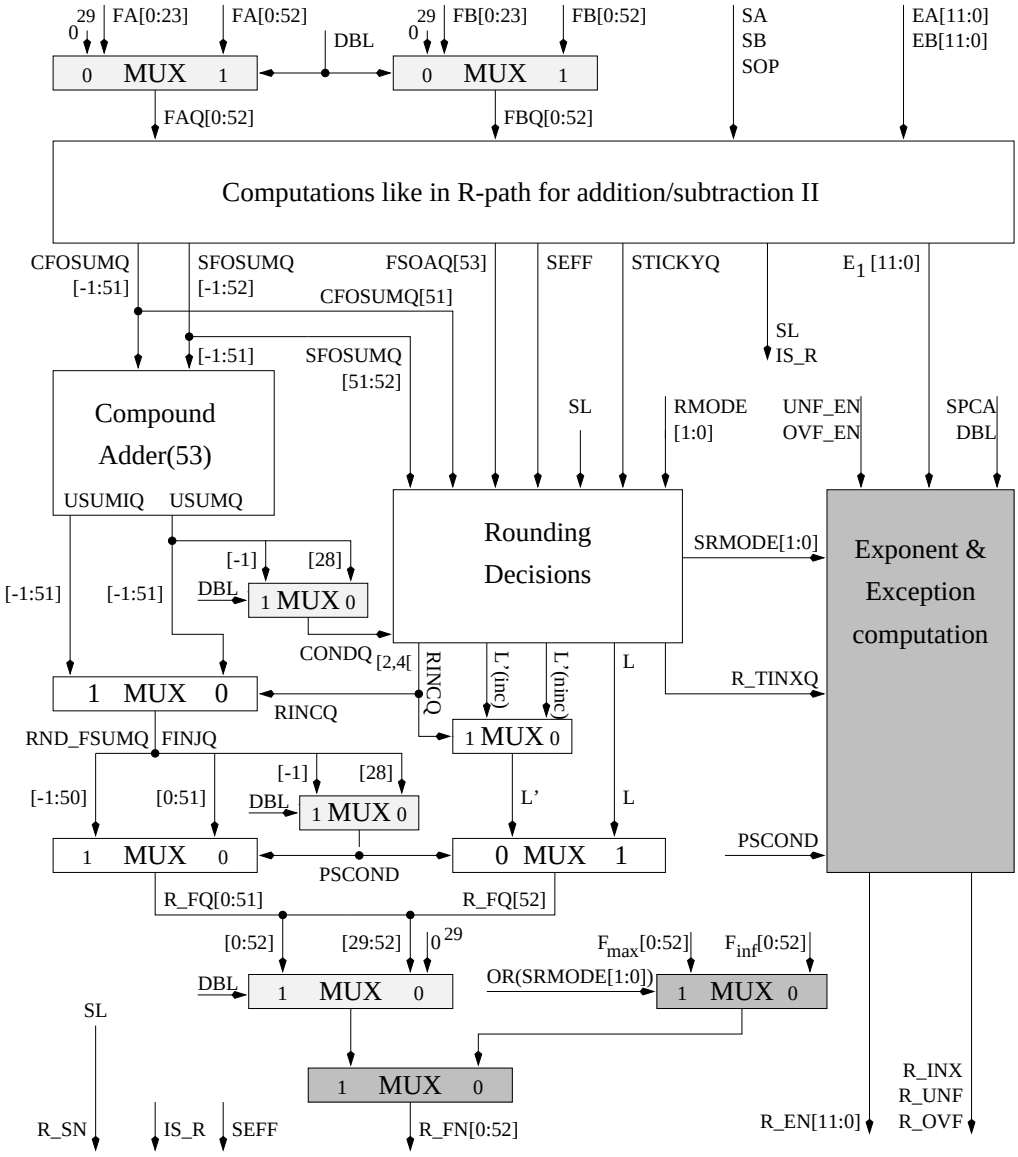

Figure 4.18: R-path implementation for the addition/subtraction unit III. Shaded boxes had to be added to the R-path implementation of the addition/subtraction unit II.

The inexactness of an result can have two reasons, either the significand rounding or the case of an overflow. The significand rounding inexactness was computed in the previous section by R_TINX, The rounding position and rounding computation has not changed in this section and only the range detection condition $\text{COND}_{[2,4[}$ was substituted by $\text{CONDQ}_{[2,4[}$. Thus, the significand rounding inexactness from lemma 4.14 becomes

$$\text{R\_TINXQ} = \text{STICKYQ} \vee ((\text{CONDQ}_{[2,4[} \wedge OR(\text{FSUMQ}[52:53])) \vee (\overline{\text{CONDQ}_{[2,4[}} \wedge \text{FSUMQ}[53]))$$

and we get the R-path inexactness condition by $\text{R\_INX} = \text{R\_TINXQ} \vee \text{R\_OVF}$.

With $\text{R\_TINY} \iff |val(r\_sn1, r\_en1, r\_fn1)| < 2^{e_{min}}$, the underflow exception for the

R-path is given by

$$\text{R\_UNF} \Longleftrightarrow \text{R\_TINY} \wedge (\text{UNF\_EN} \vee \text{R\_INX}) \wedge \overline{\text{SPCA}}.$$

The normalized factoring (r_sn1, r_en1, r_fn1) is tiny, namely R_TINY $= 1$, iff $r\_en1 < e_{min}$. Because of the exponent selection of r_en1 from e_1 and ei_1 according to equation 4.189 and $-e_{min} = <(0, \text{DBL}^3, 1^6, 0)>_2$, so that $<\text{R\_TT}[11:0]>_2 = e_1 - e_{min}$ and $<\text{R\_TTI}[11:0]>_2 = ei_1 - e_{min}$, we get

$$\text{R\_TINY} = \begin{cases} \text{R\_TTI}[11] & \text{if PSCOND} \\ \text{R\_TT}[11] & \text{otherwise.} \end{cases}$$

Thus, the underflow condition for the R-path can be written as

$$\text{R\_UNF} \Longleftrightarrow \begin{cases} \text{R\_TTI}[11] \wedge (\text{UNF\_EN} \vee \text{R\_INX}) \wedge \overline{\text{SPCA}} & \text{if PSCOND} \\ \text{R\_TT}[11] \wedge (\text{UNF\_EN} \vee \text{R\_INX}) \wedge \overline{\text{SPCA}} & \text{otherwise.} \end{cases}$$

□

Because the signal PSCOND is valid rather late, we compute each exception in two parallel paths: one path under the assumption that PSCOND $= 1$ with the signals R_OVF[1], R_INX[1] and R_UNF[1] and the other path under the assumption that PSCOND $= 0$ with the signals R_OVF[0], R_INX[0] and R_UNF[0]. Obviously, the exception flags can then be selected by:

$$(\text{R\_OVF}, \text{R\_INX}, \text{R\_UNF}) = \begin{cases} (\text{R\_OVF}[1], \text{R\_INX}[1], \text{R\_UNF}[1]) & \text{if PSCOND} \\ (\text{R\_OVF}[0], \text{R\_INX}[0], \text{R\_UNF}[0]) & \text{otherwise.} \end{cases} \tag{4.194}$$

From lemma 4.22 one can easily read off the following equations for the two paths:

$$\text{R\_OVF}[1] = (\text{ZEROTEST}(\text{E}_1[11:0] \oplus (0, \text{DBL}^3, 1^7)) \wedge \overline{\text{SPCA}} \tag{4.195}$$
$$\text{R\_OVF}[0] = 0 \tag{4.196}$$
$$\text{R\_INX}[1] = \text{R\_TINXQ} \vee \text{R\_OVF}[1] \tag{4.197}$$
$$\text{R\_INX}[0] = \text{R\_TINXQ} \tag{4.198}$$
$$\text{R\_UNF}[1] = \text{R\_TTI}[11] \wedge (\text{UNF\_EN} \vee \text{R\_INX}[1]) \wedge \overline{\text{SPCA}} \tag{4.199}$$
$$\text{R\_UNF}[0] = \text{R\_TT}[11] \wedge (\text{UNF\_EN} \vee \text{R\_INX}[0]) \wedge \overline{\text{SPCA}} \tag{4.200}$$

In the following we describe the computations of the exponent wrapping and the exponent rounding, which are required in the second step according to equation 4.168.

We split the computations for the sign, the exponent and the significand. The sign is given by R_SN1 = SL. With the preselection of the significand result for an untrapped overflow based on the rounding mode by (Note, that $srmode \neq RZ$ for $OR(\text{SR\_MODE}[1:0]) = 1$):

$$r\_fovf = <\text{R\_FOVF}[0:52]>_{neg} = \begin{cases} f_\infty = <(1, 0^{52})>_{neg} & \text{if } OR(\text{SR\_MODE}[1:0]) \\ f_{max} = <(1^{24}, \text{DBL}^{29})>_{neg} & \text{otherwise.} \end{cases}$$

the final significand can be selected according to 2.12 by

$$r\_fn = \begin{cases} r\_fovf & \text{if R\_OVF} \wedge \overline{\text{OVF\_EN}} \\ r\_fn1 & \text{otherwise.} \end{cases}$$

For the exponent computations we predict the wrapping exponent constant based on the sign of the exponent e_1.

$$pwec = \begin{cases} +\alpha = <+\text{ALPHA}[11:7]>_2 = <(0, \text{DBL}^2, 0, \overline{\text{DBL}}^2, 0^6)>_2 & \text{if } \text{E}_1[11] \\ -\alpha = <-\text{ALPHA}[11:7]>_2 = <(1, \overline{\text{DBL}}, 1, \overline{\text{DBL}}, 0, \overline{\text{DBL}}, 0^6)>_2 & \text{otherwise.} \end{cases}$$

This prediction can be done due to the fact, that for a positive exponent $e_1 \geq 0$, also $r\_en1 \geq 0$, so that no underflow can occur and for a negative exponent $e_1 < 0$, we have $r\_en1 \leq 0$, so that no overflow can occur. The exponent wrapping constant can then be selected by:

$$wec = \begin{cases} pwec & \text{if } ((\text{R\_OVF} \wedge \text{OVF\_EN}) \vee (\text{R\_UNF} \wedge \text{UNF\_EN})) \\ 0 & \text{otherwise.} \end{cases}$$

The final exponent selection including the exponent wrapping and rounding is given by

$$r\_en = \begin{cases} e_{max}+1 & \text{if R\_OVF} \wedge \overline{\text{OVF\_EN}} \wedge OR(\text{SR\_MODE}[1:0]) \\ e_{max} & \text{if R\_OVF} \wedge \overline{\text{OVF\_EN}} \wedge NOR(\text{SR\_MODE}[1:0]) \\ r\_e + pwec & \text{if R\_OVF} \wedge \text{OVF\_EN} \vee \text{R\_UNF} \wedge \text{UNF\_EN} \\ r\_e & \text{otherwise.} \end{cases} \quad (4.201)$$

By the definition of

$$r\_erp = \begin{cases} e_{max}+1 & \text{if } OR(\text{SR\_MODE}[1:0]) \\ e_{max} & \text{otherwise} \end{cases} \quad (4.202)$$

$$r\_eop = \begin{cases} r\_erp & \text{if } \overline{\text{OVF\_EN}} \wedge \overline{\text{E}_1[11]} \\ r\_en1 + pwec & \text{otherwise} \end{cases} \quad (4.203)$$

the exponent selection according to equation 4.201 can be written as

$$r\_en = \begin{cases} r\_eop & \text{if R\_OVF} \vee (\text{R\_UNF} \wedge \text{UNF\_EN}) \\ r\_en1 & \text{otherwise} \end{cases} \quad (4.204)$$

Because the computation of r_en1 is selected from e_1 and ei_1 depending on the post-normalization shift condition PSCOND, we also compute the selections of the exponents in two parallel paths for PSCOND = 0 and for PSCOND = 1 like in the computation of the exception conditions. With the convention that the appendix of the letter 'i' to a variable name indicates the incremented version of this variable, we get the final exponent by the following selections

$$ew = e_1 + pwec \qquad ewi = ei_1 + pwec$$

$$eop = \begin{cases} r\_erp & \text{if } \overline{\text{OVF\_EN}} \wedge \overline{\text{E}_1[11]} \\ ew & \text{otherwise} \end{cases} \qquad eopi = \begin{cases} r\_erp & \text{if } \overline{\text{OVF\_EN}} \wedge \overline{\text{E}_1[11]} \\ ewi & \text{otherwise} \end{cases}$$

$$en = \begin{cases} eop & \text{if R\_OVF}[0] \vee \\ & (\text{R\_UNF}[0] \wedge \text{UNF\_EN}) \\ e_1 & \text{otherwise} \end{cases} \qquad eni = \begin{cases} eopi & \text{if R\_OVF}[1] \vee \\ & (\text{R\_UNF}[1] \wedge \text{UNF\_EN}) \\ ei_1 & \text{otherwise} \end{cases}$$

$$r\_en = \begin{cases} eni & \text{if PSCOND} \\ en & \text{otherwise} \end{cases}$$

This completes the description of the second computation step according to equation 4.168. Additionaly, we have to compute the signal INF_{reg}, that indicates the case of an infinity result in the regular case:

$$\text{INF}_{reg} \Longleftrightarrow \text{R\_OVF} \wedge \overline{\text{OVF\_EN}} \wedge OR(\text{SR\_MODE}[1:0]).$$

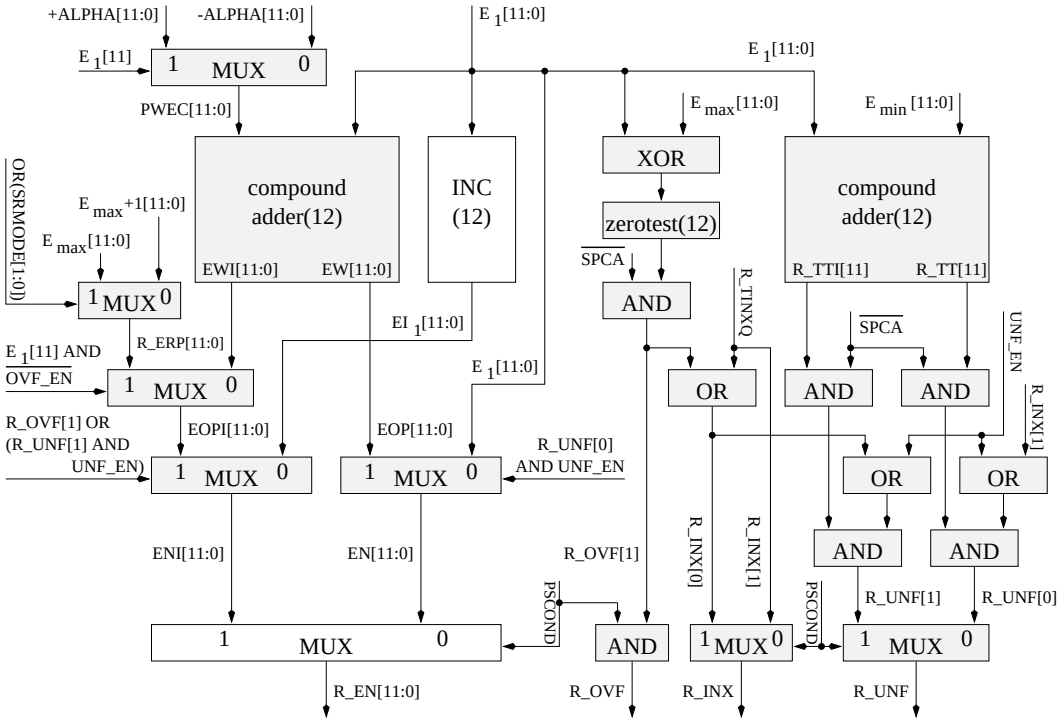

Figure 4.19: Exponent and exceptions circuit for the R-path of the addition/subtraction unit III. Shaded boxes had to be added to the R-path implementation of the addition/subtraction unit II.

The extensions and changes for the R-path of the addition/subtraction unit III based on the R-path implementation of the previous section are depicted in figure 4.18. A more detailed block diagram of the exponent and exceptions circuit is given in figure 4.19. Also in this figure the shaded circuits are required in addition to the R-path implementation from the previous section. The path selection condition is not changed in this section, so that we can use the same implementation for IS_R like in the previous section. This completes the description of the addition/subtraction III unit.

4.3 Multiplication

4.3.1 Multiplication I (normalized $\longrightarrow$ representative format)

Specification. This section describes a FP multiplication unit, that is able to multiply two FP numbers given in the normalized representations (section 2.6.3):

$$BUSa_{NF}[69:0] = (\text{SA}, \text{EA}[11:0], \text{FA}[0:52], \text{ZEROA}, \text{INFA}, \text{QNANA}, \text{SNANA}) \quad (4.205)$$

$$BUSb_{NF}[69:0] = (\text{SB}, \text{EB}[11:0], \text{FB}[0:52], \text{ZEROB}, \text{INFB}, \text{QNANB}, \text{SNANB}), \quad (4.206)$$

which represent the factorings $(sa, ea, fa) = fact_{NF}(BUSa_{NF}[69:0])$ and $(sb, eb, fb) = fact_{NF}(BUSb_{NF}[69:0])$.

In the case, that both operands have representable values, the exact product $exact_{mult}$ is defined by (section 2.2.4):

$$exact_{mult} = (-1)^{\text{SA}\oplus\text{SB}} \cdot 2^{ea+eb} \cdot fa \cdot fb. \quad (4.207)$$

If (s_{rc}, e_{rc}, f_{rc}) is a RF factoring of this exact product $exact_{mult}$ for non-zero representable inputs, then for the general case of arbitrary input values, a RF factoring of the required product is given by (see equation 2.17):

$$(s_{RF}, e_{RF}, f_{RF}) = \begin{cases} (0, e_{qNaN}, f_{qNaN}) & \text{if SCQNAN} \\ (s_{inf}, e_{\infty}, f_{\infty}) & \text{if SCINF} \\ (sa, ea, fa) & \text{if SCX} \\ (sb, eb, fb) & \text{if SCY} \\ (s_0, e_0, 0) & \text{if SCZERO} \\ (s_{rc}, e_{rc}, f_{rc}) & \text{otherwise.} \end{cases} \quad (4.208)$$

The product output of the multiplication I unit is then specified by the corresponding representation in the representative format $BUS_{RF}[73:0] = \text{RF}(s_{RF}, e_{RF}, f_{RF})$. Moreover, in the multiplication I unit the invalid flag INV should be signaled according to the occurance of an invalid exception.

Implementation. The computations of the special conditions in equation 4.208 are already summarized in section 2.4.4 by equations 2.27-2.33. Like in the addition implementations, we select the result from equation 4.61 in two steps by the definition of the sign s_{sc}, the exponent e_{sc} and the significand f_{sc} for the special case:

$$(s_{sc}, e_{sc}, f_{sc}) = \begin{cases} (0, e_{qNaN}, f_{qNaN}) & \text{if SCQNAN} \\ (s_{inf}, e_{\infty}, f_{\infty}) & \text{if SCINF} \\ (sa, ea, fa) & \text{if SCX} \\ (sb, eb, fb) & \text{if SCY} \\ (s_0, e_0, 0) & \text{otherwise.} \end{cases} \quad (4.209)$$

These computations are implemented in the special cases circuit in figure 4.20. Obviously, according to tables2.6-2.7, an invalid exception occurs for a multiplication, iff the result is a quiet NaN, in which case we have SCQNAN = 1. Thus, we already get the invalid flag by INV $\Longleftrightarrow$ SCQNAN. With the definition of the special case condition SPCA by

$$\text{SPCA} \iff \text{SCQNAN} \vee \text{SCINF} \vee \text{SCX} \vee \text{SCY} \vee \text{SCZERO} \quad (4.210)$$

like in the addition implementations, the final multiplication result can be selected by

$$(s_{RF}, e_{RF}, f_{RF}) = \begin{cases} (s_{sc}, e_{sc}, f_{sc}) & \text{if SPCA} \\ (s_{rc}, e_{rc}, f_{rc}) & \text{otherwise.} \end{cases} \quad (4.211)$$

This completes the description of the computations for the special cases and the exception recognition.

In the following the computation of the RF factoring (s_{rc}, e_{rc}, f_{rc}) for the regular case is described. For this computation we can assume non-zero representable operands.

For non-zero operands the significands are normalized with $fa, fb \in [1, 2[$, so that the product of the significands is in the range $fpr = fa \cdot fb \in [1, 4[$. Thus, according to definition 2.21, the factoring $(s_{rc}, e_{rc}, f_{rc}) = (\text{SA} \oplus \text{SB}, ea + eb, rep_{53}(fpr))$ is a RF factoring of $exact_{mult}$ for representable operands. In this way the sign s_{rc} and the exponent e_{rc} for the regular case can be computed by:

$$\text{S}_{rc} = \text{SA} \oplus \text{SB} \tag{4.212}$$

$$e_{rc} = <\text{E}_{rc}[12:0]>_2 = <(0, \text{EA}[11:0]>_2 + <(0, \text{EB}[11:0]>_2. \tag{4.213}$$

We deal with the computation of the significand $f_{rc} = rep_{53}(fpr)$ in the following. Because the significands fa and fb are both integral multiples of 2^{-52}, the product $fpr = fa \cdot fb \in [1, 4[$ is an integral multiple of 2^{-104} and can be represented by $fpr = <\text{FPR}[-1:104]>_{neg}$. From this representation of the significand product, the 53-representative $f_{rc} = rep_{53}(fpr)$ can then easily be generated following lemma 2.11:

$$\text{F}_{rc}[-1:54] = (\text{FPR}[-1:53], \text{ORtree}\,(\text{FPR}[54:104]))\,. \tag{4.214}$$

The computation of $\text{FPR}[-1:104]$ is partitioned into two steps:

(A) the computation of a carry-save representation of the product fpr by

$$<\text{FPRS}[-1:104]>_{neg} + <\text{FPRC}[-1:104]>_{neg} = <\text{FA}[0:52]>_{neg} \cdot <\text{FB}[0:52]>_{neg} \tag{4.215}$$

(B) the compression from the carry-save representation of the product to the binary product representation $\text{FPR}[-1:104]$ with

$$<\text{FPR}[-1:104]>_{neg} = <\text{FPRS}[-1:104]>_{neg} + <\text{FPRC}[-1:104]>_{neg}.$$

This equation is implemented by a 106-bit carry-lookahead adder.

The computation for step (A) consists of the partial product generation and reduction of the significand multiplication and has to be further specified. We consider two different implementations using a Booth encoded adder tree:

In the first, full-sized implementation we directly use a 53-bit by 53-bit Booth2 encoded partial product generation and reduction implementation, which we denote by the function *btree*, to implement equation 4.215 by

$$(\text{FPRS}[-1:104], \text{FPRC}[-1:104]) = \text{BTREE}_{53,53}(\text{FA}[0:52], \text{FB}[0:52]).$$

This implementation is depicted in figure 4.21.

In the second implementation of the partial product generation and reduction for step (A), we use a half-sized 53-bit by 27-bit Booth2 encoded adder tree, that is able to consider two additional constants which we denote by the function *boothtreepp*. In this "half-sized" implementation, the computations of step (A) are implemented in two iterations for double precisio and in one iteration for single precision. For the double precision computation in two iterations, we require the signal ITER2, that indicates the case that we are in the second iteration. The following lemma describes the underlying partitioning of the partial partial product formula for the significand product fpr:

Lemma 4.23 *With the selection of the significand half* $fbsel = <\text{FBSEL}[0:26]>_{neg}$ *and the definition of the sums* $pp1$*, for that we use* ITER2 $= 0$*, and* $pp2$*, for that we use* ITER2 $= 1$*, according to*

$$\text{FBSEL}[0:26] = \begin{cases} (\text{FB}[27:52], 0) & \textit{if } \text{DBL } \textit{AND } \overline{\text{ITER2}} \\ \text{FB}[0:26] & \textit{otherwise.} \end{cases} \tag{4.216}$$

$$pp1 = \sum_{i=0}^{26} fa \wedge \text{FBSEL}[i] \cdot 2^{-i} \tag{4.217}$$

$$pp2 = \sum_{i=0}^{26} fa \wedge \text{FBSEL}[i] \cdot 2^{-i} \quad + \quad 2^{-27} \cdot pp1 \tag{4.218}$$

the significand product fpr *can be selected by*

$$fpr = \begin{cases} pp2 & \textit{if } \text{DBL} \\ pp1 & \textit{otherwise} \end{cases}$$

Proof: The partial product formula for the significand product fpr can be written as

$$fpr = fa \cdot fb = fa \cdot <\text{FB}[0:52]>_{neg} \tag{4.219}$$

$$= \sum_{i=0}^{52} fa \wedge \text{FB}[i] \cdot 2^{-i} \tag{4.220}$$

$$= \sum_{i=0}^{26} fa \wedge \text{FB}[i] \cdot 2^{-i} + \sum_{i=27}^{52} fa \wedge \text{FB}[i] \cdot 2^{-i} \tag{4.221}$$

$$= \sum_{i=0}^{26} fa \wedge \text{FB}[i] \cdot 2^{-i} \quad + \quad 2^{-27} \cdot \sum_{i=0}^{26} fa \wedge \text{FB}[i+27] \cdot 2^{-i}. \tag{4.222}$$

For double precision we have in the first iteration $\text{FBSEL}[0:26] = (\text{FB}[27:52], 0)$, so that $pp1 = \sum_{i=0}^{26} fa \wedge \text{FB}[i+27] \cdot 2^{-i}$. Thus, because in the second iteration for double precision we have $\text{FBSEL}[0:26] = \text{FB}[0:26]$, it follows directly from equation 4.222, that $fprod = pp2$, as required for double precision.

Because for single precision $\text{FB}[27:52] = 0^{27}$ and $\text{FBSEL}[0:26] = \text{FB}[0:26]$, we get $fprod = \sum_{i=0}^{26} fa \wedge \text{FB}[i] \cdot 2^{-i} = pp1$, as required for single precision. □

With the definition of the feedback operand

$$fdb = \begin{cases} 2^{-27} \cdot pp1 & \text{if DBL AND ITER2} \\ 0 & \text{otherwise} \end{cases} \tag{4.223}$$

the equations from lemma 4.23 for $pp1$ with ITER2 $= 0$ and for $pp2$ with ITER2 $= 1$ can be written as

$$pp1 = fa \cdot fbsel + fdb \tag{4.224}$$

$$pp2 = fa \cdot fbsel + fdb \tag{4.225}$$

Because $fa \cdot fbsel = <\text{FA}[0:52]>_{neg} \cdot <\text{FBSEL}[0:26]>_{neg}$ is an integral multiple of 2^{-78}, the lower part of the binary representation of $pp2 = <\text{PP2}[-1:105]>_{neg}$ could be directly copied from the lower part of $fdb = <\text{FDB}[-1:105]>_{neg}$ by

$$\text{PP2}[79:105] = \text{FDB}[79:105].$$

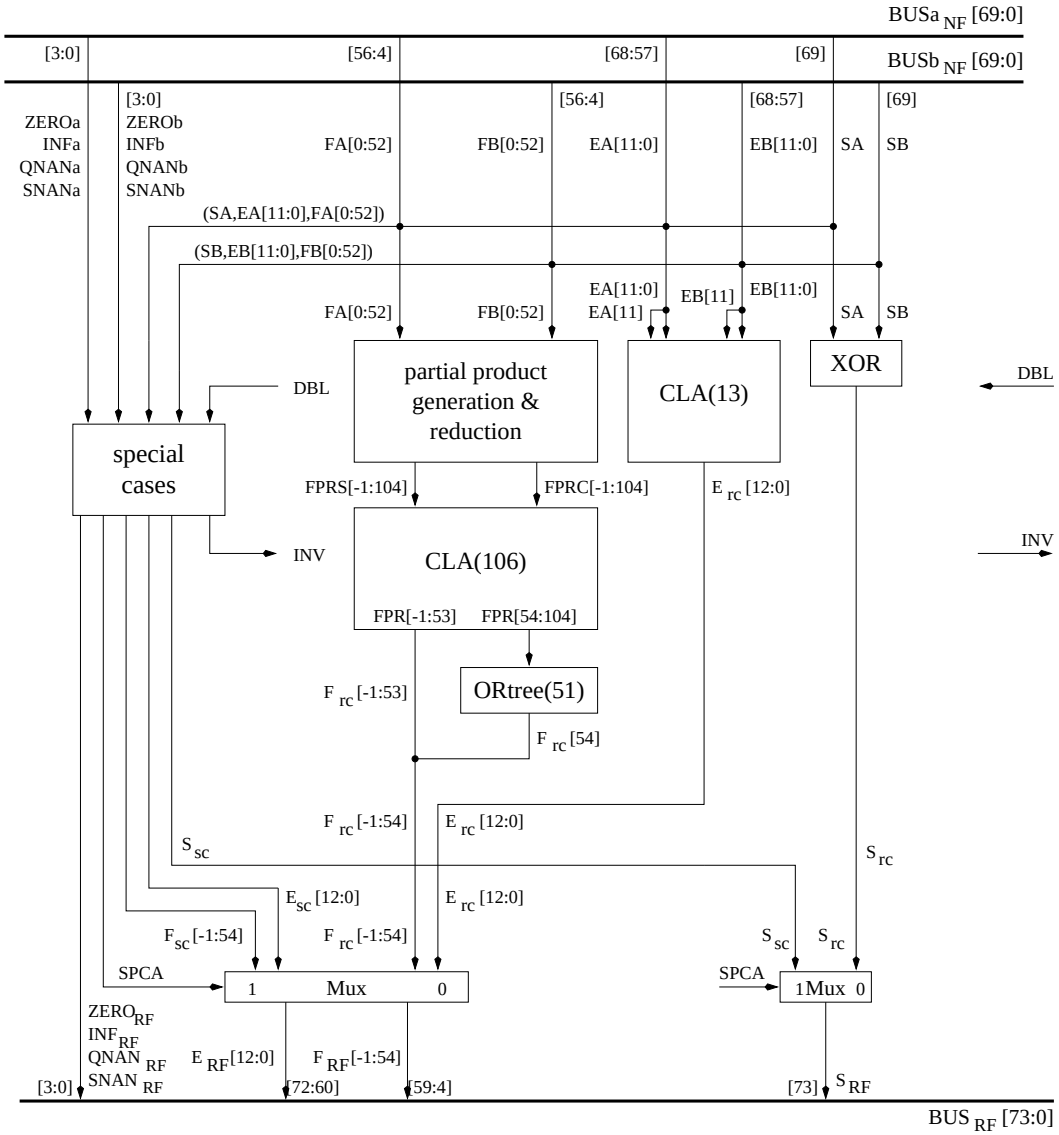

Figure 4.20: Block digram of the multiplication unit I.

During the computations of step (A), we use carry-save representations to represent $pp1$, fdb and $pp2$. These carry-save representations are denoted according to the equations

$$pp1 = <\text{PPS1}[-1:78]>_{neg} + <\text{PPC1}[-1:78]>_{neg} \quad (4.226)$$

$$fdb = <\text{FDBS}[-1:105]>_{neg} + <\text{FDBC}[-1:105]>_{neg} \quad (4.227)$$

$$pp2 = <\text{PPS2}[-1:105]>_{neg} + <\text{PPC2}[-1:105]>_{neg} \quad (4.228)$$

Following equations 4.224-4.225, the carry-save representations of $pp1$ and of $pp2$ can then be computed with the "half-sized" adder tree by the function $\text{BTREEPP}_{53,27}$:

$$(\text{PPS1}[-1:78], \text{PPC1}[-1:78]) = \text{BTREEPP}_{53,27}(\text{FA}[0:52], \text{FBSEL}[0:26], \text{FDBS}[-1:78], \text{FDBC}[-1:78])$$

$$(\text{FDBS}[-1:78], \text{FDBC}[-1:78]) = (0^{27}, \text{PPS1}[-1:51], 0^{27}, \text{PPC1}[-1:51]) \text{ AND } (\text{DBL} \wedge \text{ITER2})$$

$$(\text{PPS2}[-1:78], \text{PPC2}[-1:78]) = \text{BTREEPP}_{53,27}(\text{FA}[0:52], \text{FBSEL}[0:26], \text{FDBS}[-1:78], \text{FDBC}[-1:78])$$

$$(\text{PPS2}[79:105], \text{PPC2}[79:105]) = (\text{PPS1}[52:78], \text{PPC1}[52:78]) \text{ AND } (\text{DBL} \wedge \text{ITER2}),$$

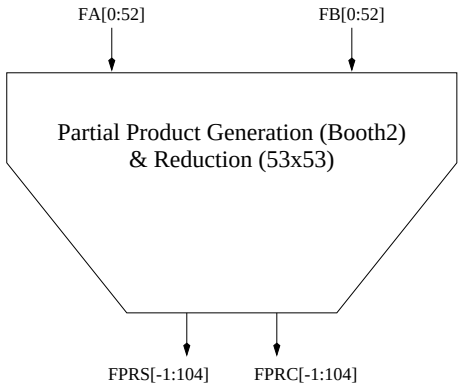

Figure 4.21: Full-sized implementation of the partial product generation and reduction for multiplication unit I.

so that we get the carry-save representation of the significand product fpr with the carry-string FPRS$[-1:104]$ and the sum-string FPRC$[-1:104]$ by

$$\text{FPRS}[-1:\,104] \;=\; \begin{cases} (\text{PPS2}[-1:78], \text{PPS1}[52:77]) & \text{if DBL} \\ (\text{PPS1}[-1:78], 0^{26}) & \text{otherwise.} \end{cases} \tag{4.229}$$

$$\text{FPRC}[-1:104] \;=\; \begin{cases} (\text{PPC2}[-1:78], \text{PPC1}[52:77]) & \text{if DBL} \\ (\text{PPC1}[-1:78], 0^{26}) & \text{otherwise.} \end{cases} \tag{4.230}$$

after one iteration for single precision and after two iterations for double precision. Based on this formula, equation 4.215 for the computation of step (A) is implemented based on the 'half-sized' adder-tree like depicted in figure 4.22. In this implementation, the results of the adder-tree are saved in the carry-save registers PPREGS$[-1:78]$ and PPREGC$[-1:78]$ after each iteration. The feedback to the adder tree is split into an upper part considering positions $[-1:78]$ and a lower part considering positions $[79:104]$. The upper part of the feedback operand FDB$[-1:78]$ is directly input into the adder tree including the right-shift by 27 positions for double precision. Because the lower part of the feedback in FDBS$[79:104]$ and FDBC$[79:104]$ is not changed by the adder tree, it can be directly saved into registers PPREGS$[79:104]$ and PPREGC$[79:104]$. In this way, the registers PPREGS$[-1:104]$ and PPREGC$[-1:104]$ contain the carry-save representation of the significand product fpr by FPRS$[-1:104]$ and FPRS$[-1:104]$ after two iterations for double precision and after one iteration for single precision.

This completes the description of the half-sized implementation for step (A), so that the descriptions of both implementations of the multiplication I unit are completed.

4.3.2 Multiplication II (normalized $\longrightarrow$ gradual result format)

Specification. Like in the previous section also in this section the FP multiplication is computed from the inputs of the normalized representations (section 2.6.3) $BUSa_{NF}[69:0]$ and $BUSb_{NF}[69:0]$. Because some rounding computations have to be considered in this section, also the input of the rounding mode, represented by RMODE$[1:0]$, is required.

In this section, the exact multiplication result according to equation 4.207 has to be rounded by the general rounding function $ground1$. After this gradual rounding step the

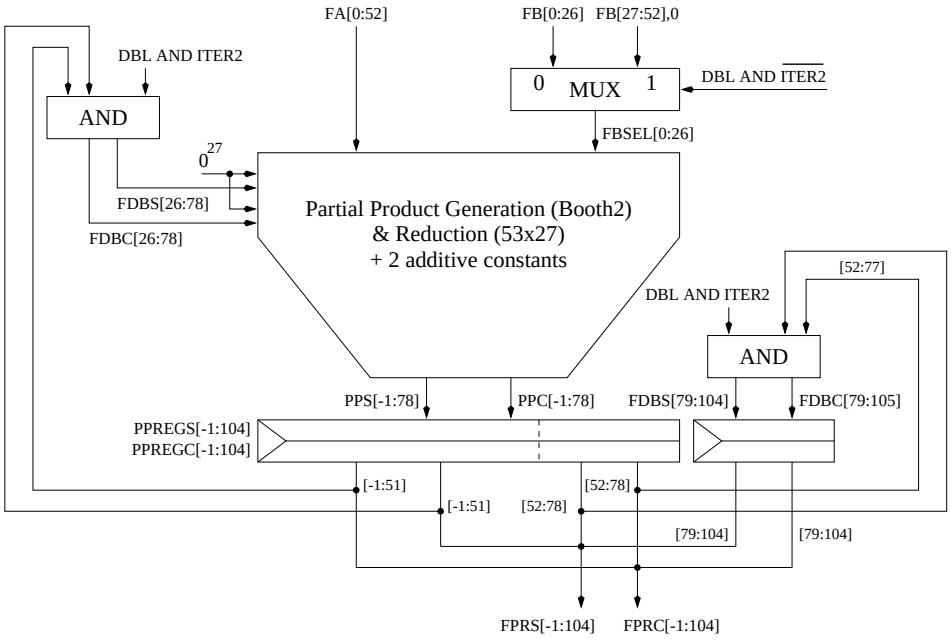

Figure 4.22: Half-sized implementation of the partial product generation and reduction for multiplication unit I.

product should be output in the gradual result format $BUS_{GF}[73:0]$ (section 2.6.5). According to equation 4.207, a factoring of the exact product is given by $(s_{ex}, e_{ex}, fpr) = (\text{SA} \oplus \text{SB}, ea + eb, fa \cdot fb)$ for non-zero representable operands. With the gradual rounded product $((s_{grc}, e_{grc}, f_{grc}), \text{TINC}, \text{TINX}) = ground1(s_{ex}, e_{ex}, fpr)$ and the following GF factoring of the result for the case of arbitrary IEEE operands

$$((s_{GF}, e_{GF}, f_{GF}), \text{TINC}_{GF}, \text{TINX}_{GF}) = \begin{cases} ((0, e_{qNaN}, f_{qNaN}), 0, 0) & \text{if SCQNAN} \\ ((s_{inf}, e_{\infty}, f_{\infty}), 0, 0) & \text{if SCINF} \\ ((sa, ea, fa), 0, 0) & \text{if SCX} \\ ((sb, eb, fb), 0, 0) & \text{if SCY} \\ ((s_0, e_0, 0), 0, 0) & \text{if SCZERO} \\ ((s_{grc}, e_{grc}, f_{grc}), \text{TINC}, \text{TINX}) & \text{otherwise,} \end{cases} \tag{4.231}$$

the product output of the multiplication unit II is specified by the gradual result representation $BUS_{GF}[73:0] = \text{GF}((s_{GF}, e_{GF}, f_{GF}), \text{TINC}_{GF}, \text{TINX}_{GF})$. The occurance of an invalid exception should be signaled by the bit INV also in this section.

Implementation. The special cases conditions and values in equation 4.231 are identical to that in the specification of the previous section. In the implementation of this special cases selection, the only difference to the previous section is that a representation in the gradual result format has 3 bits less in the significand, which have been filled with zeros in the representative format. Moreover, the gradual result format requires two additional rounding tags, which have to be zero for special value results. For the special cases selections, these small adjustments are integrated in the implementation depicted in figure 4.23. Also in the equations, that are implemented in the special cases circuit, the selections for bit positions $[-1]$ and $[53:54]$ have to be neglected.

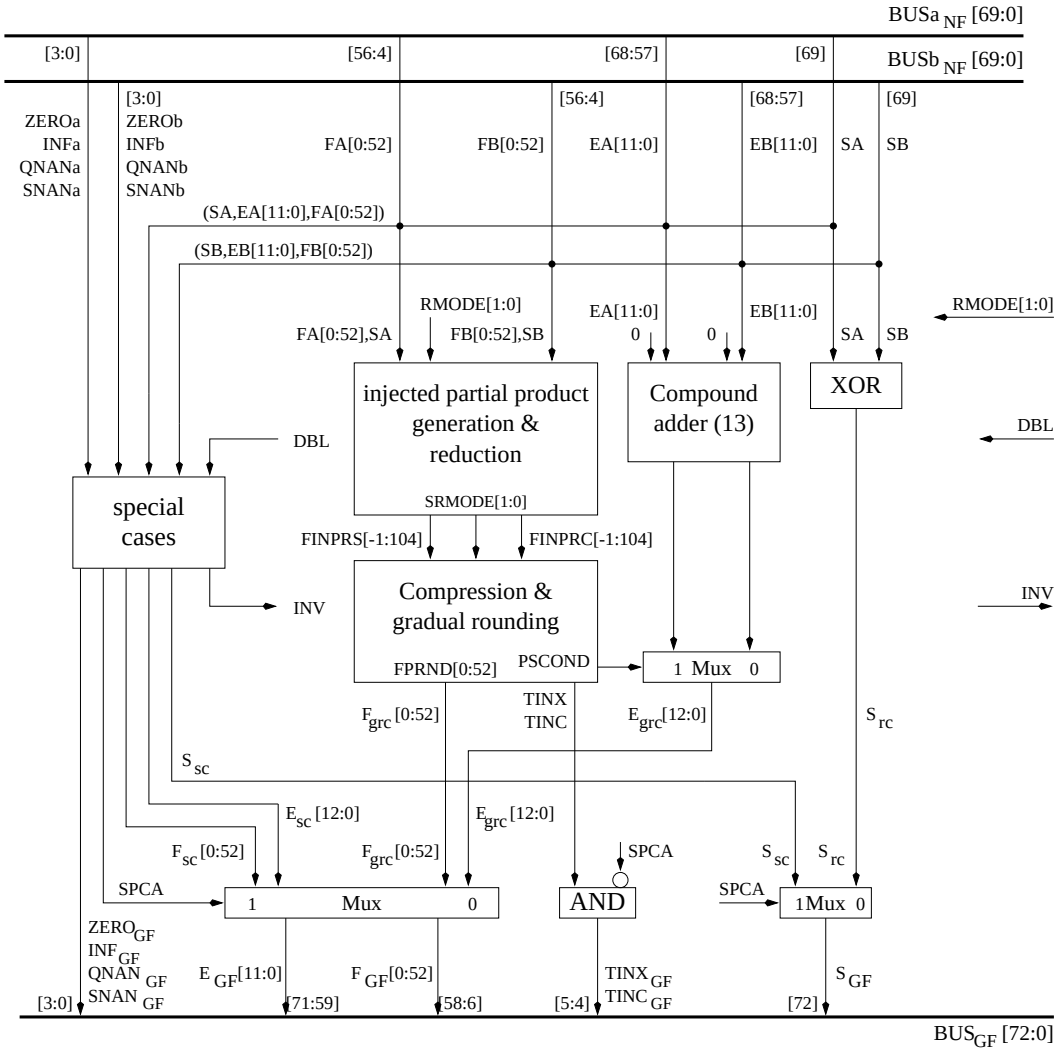

Figure 4.23: Block digram of the multiplication unit II.

This already completes the description of the special cases computation and we only have to describe the computation of $((s_{grc}, e_{grc}, f_{grc}), \text{TINC}, \text{TINX})$ for non-zero representable operands in the following.

According to definition 2.17, the gradual rounding of the product can be composed of the three steps of an unbounded normalization shift, gradual significand rounding1 and a post-normalization shift:

$$\begin{aligned} ((s_{grc}, e_{grc}, f_{grc}), \text{TINC}, \text{TINX}) &= ground1(s_{ex}, e_{ex}, f_{ex}) && (4.232) \\ &= post\_norm(sgrnd1_{mode \star s}(\eta(s_{ex}, e_{ex}, f_{ex}))) && (4.233) \end{aligned}$$

Because for both, single and double precision, $fa, fb \in [1, 2-2^{-52}]$, the exact significand product is in the range $fpr = fa \cdot fb \in [1, 4-2^{-51}]$. Thus, we can get in the same way like for the R-path of the addition II unit in lemma 4.13:

$$(s_{grc}, e_{grc}, f_{grc}) = \begin{cases} gpost\_norm(s_{ex}, e_{ex}, rnd_{mode \star \mathrm{S}_{ex},51}(fpr)) & \text{if } fpr \in [2, 4[\\ gpost\_norm(s_{ex}, e_{ex}, rnd_{mode \star \mathrm{S}_{ex},52}(fpr)) & \text{if } fpr \in [1, 2[. \end{cases} \quad (4.234)$$

With $srmode = mode \star \mathrm{S}_{ex}$ and the definition of the rounded significand product

$$fprnd = \begin{cases} rnd_{srmode,51}(fpr) & \text{if } fpr \in [2,4[\\ rnd_{srmode,52}(fpr) & \text{if } fpr \in [1,2[\end{cases} \tag{4.235}$$

the regular case factoring can be written as $(s_{grc}, e_{grc}, f_{grc}) = gpost\_norm(s_{ex}, e_{ex}, fprnd)$.

From $fpr \in [1, 4-2^{-51}]$ it follows, that also the rounded significand product is in the range $fprnd \in [1, 4-2^{-51}]$ and can be represented by $fprnd = <\text{FPRND}[-1:104]>_{neg}$. With the definition of the post-normalization condition PSCOND

$$\text{PSCOND} \iff (fprnd \in [2,4[) \iff \text{FPRND}[-1], \tag{4.236}$$

the generalized post-normalization shift (definition 4.121) can be written as

$$(s_{grc}, e_{grc}, f_{grc}) = \begin{cases} (s_{ex}, e_{ex}+1, fprnd/2) & \text{if PSCOND} \\ (s_{ex}, e_{ex}, fprnd) & \text{otherwise.} \end{cases} \tag{4.237}$$

The exponents $e_{ex} = <\mathrm{E}_{ex}[12:0]>_2 = <(0, \mathrm{EA}[11:0])>_2 + <(0, \mathrm{EB}[11:0])>_2$ and $e_{ex}+1 = <\mathrm{EI}_{ex}[12:0]>_2$ are computed by a 13-bit compound adder, so that according to equation 4.237, we get the exponent for the regular case $e_{grc} = <\mathrm{E}_{grc}[12:0]>_2$ by a selection depending on PSCOND. This exponent computation and the computations of the sign $\mathrm{S}_{grc} = \mathrm{SA} \oplus \mathrm{SB}$ are included in the block diagram in figure 4.23.

In the following we deal with the computation of the rounded significand product $fprnd$. The rounding computations are based on the injection-based rounding reduction (see section []) like it is used in the R-path of the addition units II and III. With the use of the rounding injections from equations 4.124-4.124, we get for $srmode \in \{RZ, RNU, RI\}$

$$fprnd' = \begin{cases} rnd_{srmode,51}(fpr) = rnd_{RZ,51}(fpr + inj_{[2,4[}) & \text{if } fpr \in [2,4[\\ rnd_{srmode,52}(fpr) = rnd_{RZ,52}(fpr + inj_{[1,2[}) & \text{otherwise.} \end{cases} \tag{4.238}$$

Finally, to compute $fprnd$ from $fprnd'$ by implementing RNE instead of RNU, we will have to consider the L-bit fix.

Definition 4.10 *We define the injected significand product $finpr = fpr + inj_{[1,2[}$, that already contains the rounding injection for the case that $fpr \in [1,2[$. The injection correction $injcor$ is defined by*

$$injcor = inj_{[2,4[} - inj_{[1,2[} \tag{4.239}$$

$$= \begin{cases} 2^{-52} & \textit{if } srmode = RI \\ 2^{-53} & \textit{if } srmode = RN \\ 0 & \textit{otherwise.} \end{cases} \tag{4.240}$$

We define the corrected significand product by $fcorpr = finpr + injcor = fpr + inj_{[2,4[}$.

With these definitions, equation 4.238 can be written as

$$fprnd' = \begin{cases} rnd_{srmode,51}(fpr) = rnd_{RZ,51}(finpr + injcor) & \text{if } fpr \in [2,4[\\ rnd_{srmode,52}(fpr) = rnd_{RZ,52}(finpr) & \text{otherwise.} \end{cases} \tag{4.241}$$

Like in the previous section, the computations for the significand product are partitioned into two steps:

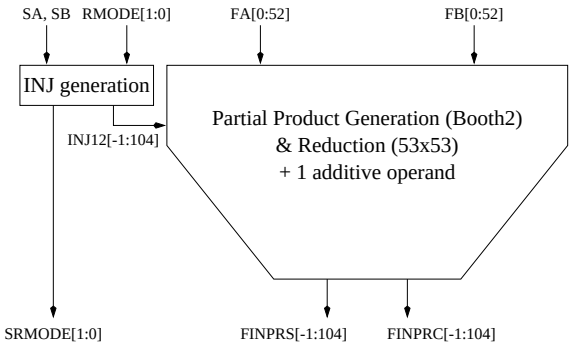

Figure 4.24: Full-sized implementation of the partial product generation and reduction with rounding injection for multiplication unit II.

(A) computation of a carry-save representation of the injected significand product $finpr = fa \cdot fb + inj_{[1,2[}$ with sum-string FINPRS$[-1:104]$ and carry-string FINPRC$[-1:104]$. This computation corresponds to the 'injected partial product generation & reduction' circuit in figure 4.23.

(B) compression and gradual rounding from the carry-save representation of the injected significand product with the bit-strings FINPRS$[-1:104]$ and FINPRC$[-1:104]$ to the rounded significand product $fprnd = <$FPRND$[-1:52]>_{neg}$ and the rounding tags TINX and TINC. In combination with the generalized post-normalization shift for the significand according to equation 4.237, these computations correspond to the 'compression & gradual rounding circuit' in figure 4.23.

(A) The only difference in the implementations of step (A) in this section from the partial product generation and reduction implementations in the previous section is the addition of the rounding injection constant $inj_{[1,2[}$.

For the full-sized adder tree implementation, we use the binary 106-bit representation of the injection $inj_{[1,2[}$. (Note, that $srmode = RNE$ for SR_MODE$[0] = 1$ and $srmode = RI$ for SR_MODE$[1] = 1$.)

$$inj_{[1,2[} = <\text{INJ12}[-1:104]>_{neg} \quad (4.242)$$

$$= <(0^{54}, \text{SR\_MODE}[0] \vee \text{SR\_MODE}[1], \text{SR\_MODE}[1]^{51})>_{neg} \quad (4.243)$$

We replace the function BTREE$_{53,53}$ from the previous section by BTREEP$_{53,53}$ to add the injection to the significand product according to $finpr = fpr + inj_{[1,2[}$ by

$$(\text{FINPRS}[-1:104], \text{FINPRC}[-1:104]) = \text{BTREEP}_{53,53}(\text{FA}[0:52], \text{FB}[0:52], \text{INJ12}[-1:104]).$$

This full-sized implementation of step (A) is depicted in figure 4.24. In this figure, the 'INJ generation' circuit contains the implementation of equation 4.243 and the rounding mode reduction according to equation 2.6-2.7.

For the half-sized adder tree implementation of step (A), we modify the feedback operand fdb from the previous section to add the rounding injection $inj_{[1,2[}$ in the first iteration for both, single and double precision. This can easily be done, because in the first iteration we have $fdb = 0$. Note, that because the result of the first iteration is added

in the second iteration weighted by 2^{-27}, the injection $2^{27} \cdot inj_{[1,2[}$ has to be added in the first iteration for double precision. In this way, we define the injection feedback by

$$fdbinj = <\text{FDBINJ}[-1:78]>_{neg} \tag{4.244}$$

$$= \begin{cases} 2^{27} \cdot inj_{[1,2[} & \text{if DBL} \\ inj_{[1,2[} & \text{otherwise.} \end{cases} \tag{4.245}$$

$$= \begin{cases} <(0^{27}, \text{SR\_MODE}[0] \vee \text{SR\_MODE}[1], \text{SR\_MODE}[1]^{52})>_{neg} & \text{if DBL} \\ <(0^{54}, \text{SR\_MODE}[0] \vee \text{SR\_MODE}[1], \text{SR\_MODE}[1]^{25})>_{neg} & \text{otherwise.} \end{cases} \tag{4.246}$$

Integrated with the previous feedback operand fdb, we define the modified feedback operand fdb' and the modified partial sums $pp1'$ and $pp2'$ by

$$fdb' = \begin{cases} fdbinj & \text{if } \overline{\text{ITER2}} \\ 2^{-27} \cdot pp1' & \text{if DBL AND ITER2} \\ 0 & \text{otherwise} \end{cases}$$

$$pp1' = fa \cdot fbsel + fdb'$$

$$pp2' = fa \cdot fbsel + fdb'$$

Lemma 4.24 *Based on the previous definitions, we get the injected significand product by*

$$finpr = fpr + inj_{[1,2[} = \begin{cases} pp2' & \textit{if DBL} \\ pp1' & \textit{otherwise.} \end{cases} \tag{4.247}$$

after one iteration for single precision and after two iterations for double precision.

Proof: For single precision, $fdbinj = inj_{[1,2[}$, so that $pp1' = pp1 + inj_{[1,2[} = fpr + inj_{[1,2[}$, as required. For double precision, we have $fdbinj = 2^{27} \cdot inj_{[1,2[}$, so that

$$\begin{aligned} pp2' &= fa \cdot fbsel + 2^{-27} \cdot pp1' \\ &= fa \cdot fbsel + 2^{-27} \cdot pp1 + 2^{-27} \cdot 2^{27} \cdot inj_{[1,2[} \\ &= fpr + inj_{[1,2[} \end{aligned}$$

and the proof of the lemma is completed. □

Thus, starting from the half-sized adder tree implementation from the previous section, only the feedback operand fdb has to be changed to fdb' and the carry-save representations of fdb' and $finpr$ have to be considered for the implementation of step (A). This half-sized implementation of step (A) is depicted in figure 4.25. The injection feedback according to equation 4.246 is generated in the 'INJ generation' circuit, which also includes the rounding mode reduction according to equations 2.6-2.7. This completes the description of the two implementations (full-sized and half-sized) for step (A).

(B) For the computation of step (B) of the significand multiplication rounding, we first consider the computation of $fprnd'$, so that with the additional computation of the L-bit fix, we will then get the rounded significand product $fprnd$.

According to equation 4.241, the computation of $fprnd' = <\text{FPRND}'[-1:52]>_{neg}$ depends on $finpr = <\text{FINPR}[-1:104]>_{neg}$ and $fcorpr = <\text{FCORPR}[-1:104]>_{neg}$. Since

$$fprnd' = \begin{cases} rnd_{RZ,51}(fcorpr) = <\text{FCORPR}[-1:51]>_{neg} & \text{if } fpr \in [2,4[\\ rnd_{RZ,52}(finpr) = <\text{FINPR}[-1:52]>_{neg} & \text{otherwise.} \end{cases}$$

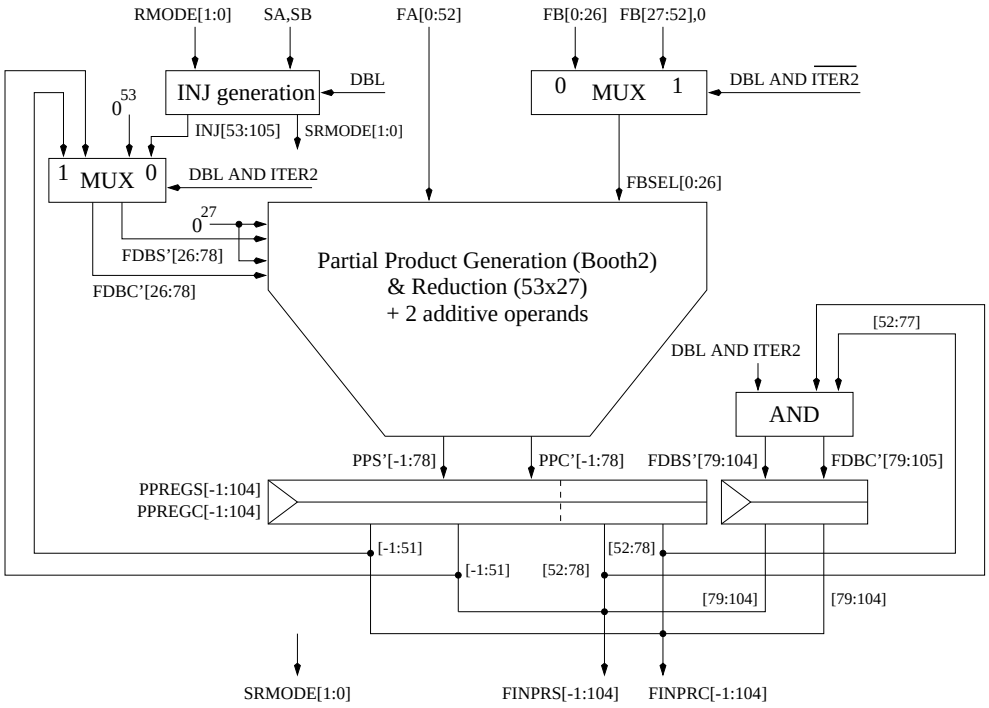

Figure 4.25: Half-sized implementation of the partial product generation and reduction with rounding injection for multiplication unit II.

only the bit strings FINPR$[-1:52]$ and FCORPR$[-1:51]$ have to be considered.

As input for the computations, we get a carry-save representation of the injected significand product $finpr$ from step (A) with the sum string FINPRS$[-1:104]$ and the carry-string FINPRC$[-1:104]$. We compress the the bit positions $[-1:52]$ of this carry-save representation by a half-adder line to the carry-save representation with sum-string XSUM$[-1:52]$ and carry-string XCARRY$[-1:51]$, so that

$$<\text{XSUM}[-1:52]>_{neg} + <\text{XCARRY}[-1:51]>_{neg} = <\text{FINPRS}[-1:52]>_{neg} + <\text{FINPRC}[-1:52]>_{neg}.$$

Moreover, the bit positions $[53:104]$ of the carry-save representation of $finpr$ are compressed by a carry-look-ahead adder to the binary representation (C52, FINPRS$[53:104]$) according to

$$<(\text{C52}, \text{FINPR}[53:104])>_{neg} = <\text{FINPRS}[53:104]>_{neg} + <\text{FINPRC}[53:104]>_{neg}$$

In this sum, the bit C52 is generated as a carry bit into position [52].

Based on these compressed representations, we partition the computations for $finpr$ and $fcorpr$ into an upper part considering bit positions $[-1:51]$ and a lower part considering bit positions $[52:104]$.

For the computation of $finpr$, the lower part consists of

$$lpart12 = <\text{XSUM}[52]>_{neg} + <(\text{C52}, \text{FINPR}[53:104])>_{neg} \quad (4.248)$$

$$= <(\text{RC12}, \text{FINPR}[52:104])>_{neg} . \quad (4.249)$$

Thus, the bit FINPR[52] and the carry bit RC12 (rounding carry for $fpr \in [1,2[$) into position [51] can be computed by a half-adder according to

$$<(\text{RC12}, \text{FINPR}[52])> \quad = \quad <\text{XSUM}[52]> + <\text{C52}> . \tag{4.250}$$

For the computation of $fcorpr = finpr + injcor$, we additionaly have to consider the injection correction $injcor$. According to equation 4.240, we have for the injection correction $injcor \leq 2^{-52}$, and $injcor$ can be represented by $injcor = <\text{INJCOR}[52:53]>_{neg}$. Thus, the lower part of $fcorpr$ consists of

$$\begin{aligned} lpart24 &= <xsum[52]>_{neg} + <(\text{C52}, finpr[53:104])>_{neg} + <\text{INJCOR}[52:53]>_{neg} \\ &= <(\text{RC24}, fcorpr[52:104])>_{neg} \end{aligned}$$

Because $injcor \leq 2^{-52}$, we can add the tail $<(\text{FINPR}[53:104])>_{neg}$ with the injection correction by

$$<(\text{CC52}, \text{FCORPR}[53:104])>_{neg} = <\text{FINPR}[53:104]>_{neg} + <\text{INJCOR}[52:53]>_{neg}$$

In this addition, the carry bit CC52 (correction carry into position [52]) is generated. Using the definition of the injection correction and the encoding for the reduced rounding modes according to table 2.3, the condition for the correction carry CC52 can be written as

$$\text{CC52} \quad = \quad \text{SR\_MODE}[1] \vee \text{SR\_MODE}[0] \wedge \text{FINPR}[53]. \tag{4.251}$$

Thus, the bits FCORPR[52] and RC24, which is the carry from the lower part of $fcorpr$ into bit position [51], can be computed by a full-adder according to

$$<(\text{RC24}, \text{FCORPR}[52])> \quad = \quad <\text{XSUM}[52]> + <\text{C52}> + <\text{CC52}>. \tag{4.252}$$

The upper part of $finpr$ and $fcorpr$ consists of

$$\begin{aligned} <\text{FINPR}[-1:51]>_{neg} &= <\text{XSUM}[-1:51]>_{neg} + <\text{XCARRY}[-1:51]>_{neg} + \text{RC12} \cdot 2^{-51} \\ <\text{FCORPR}[-1:51]>_{neg} &= <\text{XSUM}[-1:51]>_{neg} + <\text{XCARRY}[-1:51]>_{neg} + \text{RC24} \cdot 2^{-51} \end{aligned}$$

With the definition of the upper product upr and the incremented upper product $upri$ by

$$\begin{aligned} upr &= <\text{UPR}[-1:51]>_{neg} && (4.253) \\ &= <\text{XSUM}[-1:51]>_{neg} + <\text{XCARRY}[-1:51]>_{neg} && (4.254) \\ upri &= <\text{UPRI}[-1:51]>_{neg} && (4.255) \\ &= upr + 2^{-51}, && (4.256) \end{aligned}$$

the upper parts of $finpr$ and $fcorpr$ both can only have either the value upr or the value $upri$. Obviously, only the carry, which is generated from the lower part into position [51], is differing in the upper parts for $finpr$ and $fcorpr$. Thus, if we select the proper carry bit into position [51] depending on whether $fpr \in [1,2[$ or $fpr \in [2,4[$ by

$$\text{RCARRY51} = \begin{cases} \text{RC24} & \text{if } fpr \in [2,4[\\ \text{RC12} & \text{otherwise,} \end{cases} \tag{4.257}$$

the upper part of the significand product $fprnd'$ can by selected by

$$<\text{FPRND}'[-1:51]>_{neg} \quad = \quad \begin{cases} upri = <\text{UPRI}[-1:51]>_{neg} & \text{if RCARRY51} \\ upr = <\text{UPR}[-1:51]>_{neg} & \text{otherwise} \end{cases} \tag{4.258}$$

Additionaly, the bit FPRND′[52] is required for $fpr \in [1,2[$. In this case, we have FPRND′[52] = FINPR[52], which we already computed before, so that

$$\text{FPRND}'[52] = \begin{cases} 0 & \text{if } fpr \in [2,4[\\ \text{FINPR}[52] & \text{otherwise.} \end{cases} \tag{4.259}$$

This completes the computation of $fprnd'$. To get the rounded significand product $fprnd$, we additionaly have to implement the L-bit fix. In contrast to the L-bit fix implementation for the addition II unit, we have to consider that the injected significand product $fprnd = fpr + inj_{[1,2[}$ contains the rounding injection $inj_{[1,2[} = 2^{-53}$ for $srmode = RN$. In this way, the conditions for the L-bit fix are given by

$$\begin{aligned} \text{LFIX}_{[1,2[} &= \text{SR\_MODE}[0] \wedge \overline{\text{FINPR}[53]} \wedge \overline{OR(\text{FINPR}[54\!:\!104])} \\ \text{LFIX}_{[2,4[} &= \text{SR\_MODE}[0] \wedge \text{FINPR}[52] \wedge \text{FINPR}[53] \wedge \overline{OR(\text{FINPR}[54\!:\!104])}, \end{aligned}$$

so that the rounded significand product $fprnd = <\text{FPRND}[-1\!:\!52]>_{neg}$ can be computed by

$$\text{FPRND}[-1\!:\!52] = \begin{cases} (\text{FPRND}'[-1\!:\!50], \text{FPRND}'[51] \wedge \overline{\text{LFIX}_{[2,4[}}, 0) & \text{if } fpr \in [2,4[\\ (\text{FPRND}'[-1\!:\!51], \text{FPRND}'[52] \wedge \overline{\text{LFIX}_{[1,2[}}) & \text{otherwise.} \end{cases}$$

In this selection, only the bits in positions $[51:52]$ are differing and have to be selected. We denote the least significant bit of the significand by L24 for the case that $fpr \in [2,4[$ and by L12 for the case that $fpr \in [1,2[$ according to (Note, that for $fpr \in [1,2[$, we have FPRND′[52] = FINPR[52])

$$\text{L24} = \text{FPRND}'[51] \wedge \overline{\text{LFIX}_{[2,4[}} \tag{4.260}$$

$$\text{L12} = \text{FINPR}[52] \wedge \overline{\text{LFIX}_{[1,2[}}, \tag{4.261}$$

so that equation 4.260 can be written as

$$\text{FPRND}[-1\!:\!52] = \begin{cases} (\text{FPRND}'[-1\!:\!50], \text{L24}, 0) & \text{if } fpr \in [2,4[\\ (\text{FPRND}'[-1\!:\!51], \text{L12}) & \text{otherwise.} \end{cases} \tag{4.262}$$

In the description of the rounding computations, the condition $fpr \in [2,4[$ is used to choose the proper rounding injection and to choose either $rnd_{RZ,51}(fcorpr)$ or $rnd_{RZ,52}(finpr)$ as the rounded result $fprnd'$. Because we only deal with the injected significand products, we do not have a signal, that exactly implements the condition $fpr \in [2,4[$. The following lemma shows, that the bit UPR[−1] can be used to substitute the condition $fpr \in [2,4[$. The bit UPR[−1] does not always agree with the condition $fpr \in [2,4[$, but it will be shown, that in every case, where UPR[−1] fails to predict the condition $fpr \in [2,4[$ correctly, it does not matter which rounding injection is chosen, because in these cases $rnd_{RZ,51}(fcorpr) = rnd_{RZ,52}(finpr)$.

Lemma 4.25 *For the rounding computation according to equation 4.238, the condition* $fpr \in [2,4[$ *can be substituted by the signal* UPR[−1]*, so that*

$$fprnd' = \begin{cases} rnd_{RZ,51}(fcorpr) & \textit{if } \text{UPR}[-1] \\ rnd_{RZ,52}(finpr) & \textit{otherwise.} \end{cases}$$

Proof: We only have to consider the cases where $\text{UPR}[-1] \neq (fpr \in [2,4[)$. In the following, we distinguish between: (a) $\text{UPR}[-1] = 0$; and (b) $\text{UPR}[-1] = 1$.

(a) For $\text{UPR}[-1] = 0$, we have to consider the case, that $(fpr \in [2,4[)$. Because $fpr \geq 2$ and $finpr = fpr + inj_{[1,2[} = upr + lpart12$, we have

$$finpr \in [2, upr + lpart12], \tag{4.263}$$

where $lpart12 < 3 \cdot 2^{-52}$ according to equation 4.248. Since $\text{UPR}[-1] = 0$, it follows that $upr = <\text{UPR}[-1:51]>_{neg} \leq 2 - 2^{-51}$. Since $upr + lpart12 \geq exact \geq 2$, we have $upr > 2 - 3 \cdot 2^{-52}$. Thus, $upr = <\text{UPR}[-1:51]>_{neg} = 2 - 2^{-51}$ and equation 4.263 yields

$$finpr \in [2, 2 + 2^{-52}[.$$

The injection correction satisfies $0 \leq injcor \leq 2^{-52}$, therefore

$$fcorpr = finpr + injcor \in [2, 2 + 2^{-51}[.$$

For these ranges of $finpr$ and $fcorpr$, it follows, that

$$rnd_{RZ,51}(fcorpr) = rnd_{RZ,52}(finpr) = 2.$$

Thus, it does not matter which rounding injection is chosen in this case and $fprnd' = 2$ independent of the selection value.

(b) For $\text{UPR}[-1] = 1$, we only have to consider the case, that $fpr < 2$. Since $inj_{[1,2[} \in [0, 2^{-52}[$, it follows that $finpr = fpr + inj_{[1,2[} < 2 + 2^{-52}$. Since $\text{UPR}[-1] = 1$ and $finpr = upr + lpart12$, we have $finpr \geq 2$, so that

$$finpr \in [2, 2 + 2^{-52}[.$$

The proof now follows the proof of case (a). □

By the use of this lemma, equations 4.257 and 4.259 are implemented by

$$\text{RCARRY51} = \begin{cases} \text{RC24} & \text{if UPR}[-1] \\ \text{RC12} & \text{otherwise,} \end{cases}$$

$$\text{FPRND}'[52] = \begin{cases} 0 & \text{if UPR}[-1] \\ \text{FINPR}[52] & \text{otherwise.} \end{cases}$$

This completes the description of the rounded significand product $fprnd$. Additionaly, for step (B) we have to compute the rounding tags for the rounding inexactness TINX and for the rounding increment TINC.

The conditions for the rounding tags TINX and TINC are given by:

$$\text{TINX} = \begin{cases} \text{FPR}[52] \vee ORtree(\text{FPR}[53:104]) & \text{if UPR}[-1] \\ ORtree(\text{FPR}[53:104]) & \text{otherwise.} \end{cases}$$

$$\text{TINC} = \begin{cases} \text{FPRND}[51] \overline{\oplus} \text{FPR}[51] & \text{if UPR}[-1] \\ \text{FPRND}[52] \overline{\oplus} \text{FPR}[52] & \text{otherwise.} \end{cases}$$

The following lemma provides the equations for the implementation of the rounding tags based on the injected significand product $finpr$. Moreover, this lemma proposes how the computation of the LFIX-bits can be based on signals from the rounding tag computation to share hardware.

Lemma 4.26 *With the definition of*

$$\begin{aligned}
\text{FPR}_{RN}[51\!:\!53] &= <(\text{XSUM}[51] \oplus \text{XCARRY}[51] \oplus \text{RC12}, \text{FINPR}[52\!:\!53])>_{neg} - 2^{-53} \bmod 2^{-50} \\
\text{TINC}_{RN} &= \begin{cases} \text{FPR}_{RN}[51] \oplus (\text{FPRND}'[51] \wedge \overline{\text{LFIX}_{[2,4[}} & \textit{if } \text{UPR}[-1] \\ \text{FPR}_{RN}[52] \oplus \text{FPRND}'[52] \wedge \overline{\text{LFIX}_{[1,2[}} & \textit{otherwise.} \end{cases} \\
\text{STICKY2} &= \mathit{ORtree}(\text{FINPR}[53\!:\!104] \oplus \text{SR_MODE}[1])
\end{aligned}$$

the rounding tags can be computed by

$$\begin{aligned}
\text{TINX} &= ((\text{SR_MODE}[0] \oplus \text{FPR}[52]) \wedge \text{UPR}[-1]) \vee \text{STICKY2} \\
\text{TINC} &= (\text{SR_MODE}[0] \overline{\wedge} \text{TINC}_{RN}) \overline{\wedge} (\text{SR_MODE}[1] \overline{\wedge} \text{TINX}).
\end{aligned}$$

Moreover, based on the signal STICKY2*, the* LFIX*-bits can be written as:*

$$\begin{aligned}
\text{LFIX}_{[1,2[} &= \text{SR_MODE}[0] \wedge \overline{\text{FINPR}[53]} \wedge \overline{\text{STICKY2}} \\
\text{LFIX}_{[2,4[} &= \text{SR_MODE}[0] \wedge \text{FINPR}[52] \wedge \text{FINPR}[53] \wedge \overline{\text{STICKY2}},
\end{aligned}$$

Proof: In order to proof the equations for the rounding tags and the LFIX-bits, we first show some properties of the signals $\text{FPR}_{RN}[51:53]$ and STICKY2, namely, that (a) $\text{STICKY2} = \mathit{ORtree}(\text{FPR}[53\!:\!104])$ and that (b) in the rounding mode $srmode = RNE$, we have $\text{FPR}_{RN}[51\!:\!53] = \text{FPR}[51\!:\!53]$:

(a) Keeping in mind, that $finpr = fpr + inj_{[1,2[}$, we distinguish for the proof between the two cases: (i) the rounding mode $srmode \in \{RZ, RNE\}$; and (ii) the rounding mode $srmode = RI$.

(i) For $srmode \in \{RZ, RNE\}$, we have $\text{SR\_MODE}[1] = 0$ and $\text{INJ}_{[1,2[}[53\!:\!104] = 0^{52}$, so that $\text{FPR}[53\!:\!104] = \text{FINPR}[53\!:\!104] \oplus \text{SR\_MODE}[1]$ and (a) follows immediately.

(ii) For $srmode = RI$, we have $\text{SR\_MODE}[1] = 1$ and $\text{INJ}_{[1,2[}[53\!:\!104] = 1^{52}$. Thus,

$$\begin{aligned}
(\text{FPR}[53\!:\!104] = 0^{52}) &\iff (\text{FINPR}[53\!:\!104] = 1^{52}) \\
(\text{FPR}[53\!:\!104] = 0^{52}) &\iff ((\text{FINPR}[53\!:\!104] \oplus \text{SR_MODE}[1]) = 0^{52}) \\
\mathit{ORtree}(\text{FPR}[53\!:\!104]) &\iff \mathit{ORtree}(\text{FINPR}[53\!:\!104] \oplus \text{SR_MODE}[1]),
\end{aligned}$$

as required.

(b) In the rounding mode $srmode = RNE$, we have for the rounding injection constant $\text{INJ}_{[1,2[} = 2^{-53}$. Thus, $<\text{FPR}[-1\!:\!53]>_{neg} = <\text{FINPR}[-1\!:\!53]>_{neg} - 2^{-53}$, and therefore

$$<\text{FPR}[51\!:\!53]>_{neg} = <\text{FINPR}[51\!:\!53]>_{neg} - 2^{-53} \bmod 2^{-50}.$$

Property (b) follows then from $\text{FINPR}[51] = \text{XSUM}[52] \oplus \text{XCARRY}[52] \oplus \text{RC12}$.

The equations for TINX and the LFIX-bits follow immediately from property (a). In the proof of the equation for TINC, we distinguish between the three cases: (i) $srmode = RZ$; (ii) $srmode = RNE$; and (iii) $srmode = RI$.

(i) In the rounding mode $srmode = RZ$, a rounding increment never occurs.

(iii) It follows from the IEEE rounding definition, that in the rounding mode $srmode = RI$, a rounding increment occurs, iff the result is inexact, where $(\text{TINX} = 1)$.

(ii) In the rounding mode $srmode = RNE$, we implement equation 4.264 for TINC. Using property (b) and the equations for $\text{FPRND}[51] = \text{FPRND}'[51] \wedge \overline{\text{LFIX}_{[2,4[}}$ and $\text{FPRND}[52] = \text{FPRND}'[52] \wedge \overline{\text{LFIX}_{[1,2[}}$, we get in the rounding mode $srmode = RNE$, that $\text{TINC} = \text{TINC}_{RN}$.

We join the three cases (i)-(iii) for the three rounding modes to the following general equation for TINC:

$$\text{TINC} = \begin{cases} \text{TINX} & \text{if } srmode = RI \\ \text{TINC}_{RN} & \text{if } srmode = RNE \end{cases}$$

Because the rounding mode $srmode = RI$ is signaled by $\text{SR\_MODE}[1] = 1$ and the rounding mode $srmode = RNE$ by $\text{SR\_MODE}[0] = 1$, it is obvious, that this is equivalent to the equation, that we have to prove for TINC. □

In this way, the description of the implementation of part (B) with the rounded significand product $fprnd$ and the rounding tags TINX and TINC is completed. Based on this, the significand $f_{grc} = <\text{F}_{grc}[0:52]>_{neg}$ for the regular case can be selected from $fprnd = <\text{FPRND}[-1:52]>_{neg}$ by the generalized post-normalization shift for the significand according to equation 4.237. In the following, we summarize the computation steps for the computation of part (B) of the significand multiplication and rounding and the generalized post-normalization shift:

1. compression of positions $[-1:52]$ of the carry-save representation of $fprnd$ by a half-adder line according to

$$<\text{XSUM}[-1:52]>_{neg} + <\text{XCARRY}[-1:51]>_{neg} = <\text{FINPRS}[-1:52]>_{neg} + <\text{FINPRC}[-1:52]>_{neg}.$$

   addition of bit positions $[53:104]$ of the carry-save representation of $fprnd$ by a 52-bit carry-lookahead adder according to:

$$<(\text{C52}, \text{FINPR}[53:104])>_{neg} = <\text{FINPRS}[53:104]>_{neg} + <\text{FINPRC}[53:104]>_{neg}.$$

2. computation of the upper product $upr = <\text{UPR}[-1:51]>_{neg}$ (equation 4.254) and the incremented upper product $upri = <\text{UPRI}[-1:51]>_{neg}$ (equation 4.256) by a 53-bit compound adder that computes

$$\begin{aligned} <\text{UPR}[-1:51]>_{neg} &= <\text{XSUM}[-1:51]>_{neg} + <\text{XCARRY}[-1:51]>_{neg} \\ <\text{UPRI}[-1:51]>_{neg} &= <\text{UPR}[-1:51]>_{neg} + 2^{-51}, \end{aligned}$$

3. After the computation of the correction carry CC52 into position [52] (equation 4.251), the rounding carries into position 51 are computed: RC12 for the case $fpr \in [1,2[$ by a half-adder according to equation 4.250 and RC24 for the case $fpr \in [2,4[$ by a full-adder according to equation 4.252. The proper rounding carry RCARRY51 into position [51] is then selected according to equation 4.264:

$$\begin{aligned} \text{CC52} &= \text{SR\_MODE}[1] \vee \text{SR\_MODE}[0] \wedge \text{FINPR}[53] \\ <(\text{RC12}, \text{FINPR}[52])> &= <\text{XSUM}[52]> + <\text{C52}> \\ <(\text{RC24}, \text{FCORPR}[52])> &= <\text{XSUM}[52]> + <\text{C52}> + <\text{CC52}>. \\ \text{RCARRY51} &= \begin{cases} \text{RC24} & \text{if UPR}[-1] \\ \text{RC12} & \text{otherwise,} \end{cases} \end{aligned}$$

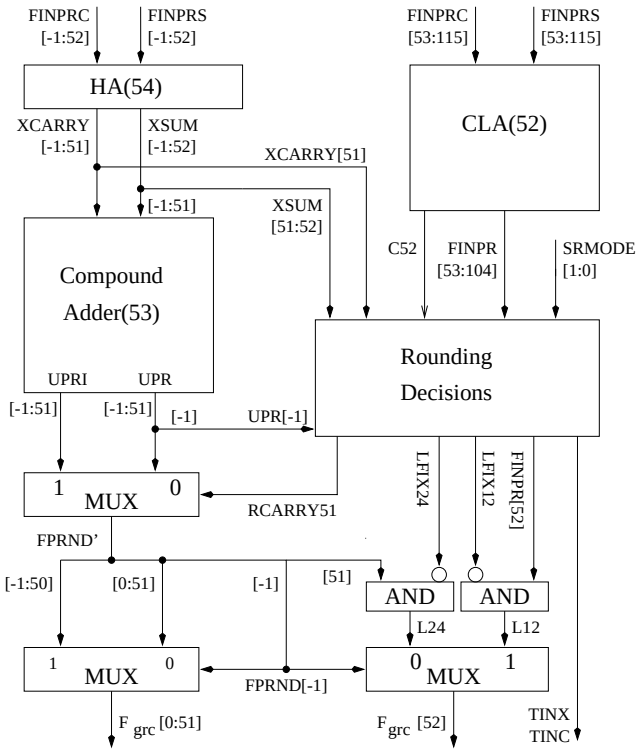

Figure 4.26: Block diagram of part (B) of the significand multiplication including the gradual rounding computations and the generalized post-normalization shift for multiplication unit II.

4. Depending on the value of the rounding carry RCARRY51, the upper part (positions $[-1{:}51]$) of the rounded significand product $fprnd'$ is selected by (equation 4.258):

$$\text{FPRND}'[-1{:}51] \quad = \quad \begin{cases} \text{UPRI}[-1{:}51] & \text{if RCARRY51} \\ \text{UPR}[-1{:}51] & \text{otherwise.} \end{cases}$$

5. The rounding tags and the LFIX-bits are computed according to the equations from lemma 4.26.

6. The L-bits of the rounded significand product L24 and L12 are computed according to equations 4.260-4.261 considering the L-bit fix:

$$\begin{aligned} \text{L24} &= \text{FPRND}'[51] \wedge \overline{\text{LFIX}_{[2,4[}} \\ \text{L12} &= \text{FINPR}[52] \wedge \overline{\text{LFIX}_{[1,2[}}, \end{aligned}$$

7. Finally, the generalized post-normalization shift of the significand is computed according to equation 4.262 and 4.237. This selection is computed separately for positions $[0{:}51]$ by

$$\text{F}_{grc}[0:51] = \begin{cases} \text{FPRND}'[-1{:}50] & \text{if FPRND}'[-1] \\ \text{FPRND}'[0{:}51] & \text{otherwise} \end{cases}$$

and for position [52] by

$$\mathrm{F}_{grc}[0:51] = \begin{cases} \mathrm{L24} & \text{if } \mathrm{FPRND}'[-1] \\ \mathrm{L12} & \text{otherwise} \end{cases}$$

The implementation of these steps is depicted in the block diagram in figure 4.26. In this figure, the 'rounding decisions' circuit contains the implementation of steps 3 and 5.

In this way, the description of the multiplication unit II is completed.

4.3.3 Multiplication III (normalized $\longrightarrow$ normalized format)

Specification. Like in the previous section also in this section, the FP multiplication is computed from the inputs of the normalized representations $BUSa_{NF}[69:0]$ and $BUSb_{NF}[69:0]$ (section 2.6.3). Because IEEE rounding has to be considered in this section, also the bit DBL, that signals the case of single precision (DBL = 0) or double precision (DBL = 1), the input of the rounding mode, represented by RMODE[1:0], and the underflow and overflow enable bits UNF_EN and OVF_EN are required.

In this section, the exact multiplication result according to equation 4.207 has to be rounded by the rounding function *nround*, that computes the NF factoring of the IEEE rounded result. After this rounding computation the product should be output in the normalized format $BUS_{NF}[69:0]$ (section 2.6.3). According to equation 4.207, a factoring of the exact product is given by $(s_{pr}, e_{pr}, fpr) = (\mathrm{SA} \oplus \mathrm{SB}, ea + eb, fa \cdot fb)$ for non-zero representable operands. With the NF factoring of the IEEE result for non-zero representable operands $(s_{nrc}, e_{nrc}, f_{nrc}) = nround(s_{ex}, e_{ex} + wec, fpr)$ including the exponent wrapping constant *wec* according to equation 2.14 and the following NF factoring of the result for the general case of arbitrary operands according to equation 2.16:

$$(s_{NF}, e_{NF}, f_{NF}) = \begin{cases} (0, e_{qNaN}, f_{qNaN}) & \text{if SCQNAN} \\ (s_{inf}, e_{\infty}, f_{\infty}) & \text{if SCINF} \\ (sa, ea, fa) & \text{if SCX} \\ (sb, eb, fb) & \text{if SCY} \\ (s_0, e_0, 0) & \text{if SCZERO} \\ (s_{nrc}, e_{nrc}, f_{nrc}) & \text{otherwise,} \end{cases} \tag{4.264}$$

the product output of the multiplication unit III is specified by the corresponding representation in the normalized format $BUS_{NF}[69:0] = \mathrm{NF}(s_{NF}, e_{NF}, f_{NF})$. In this section, the occurance of an invalid, inexact, overflow and underflow exception should be signaled by the bits INV, INX, OVF and UNF, respectively.

Implementation. The special cases conditions and values in equation 4.264 are identical to that in the specification of the two previous sections. In the implementation of this special cases selection, the only difference is that in this case a representation in the normalized format is required. Because all special cases results are exact, just the two rounding tags have to be neglected from the special cases implementation of the previous section. For the special cases selections, these small adjustments are integrated in the implementation depicted in figure 4.27. This already completes the description of the special cases computation and we only have to describe the computation of $(s_{nrc}, e_{nrc}, f_{nrc})$ for non-zero representable operands in the following.

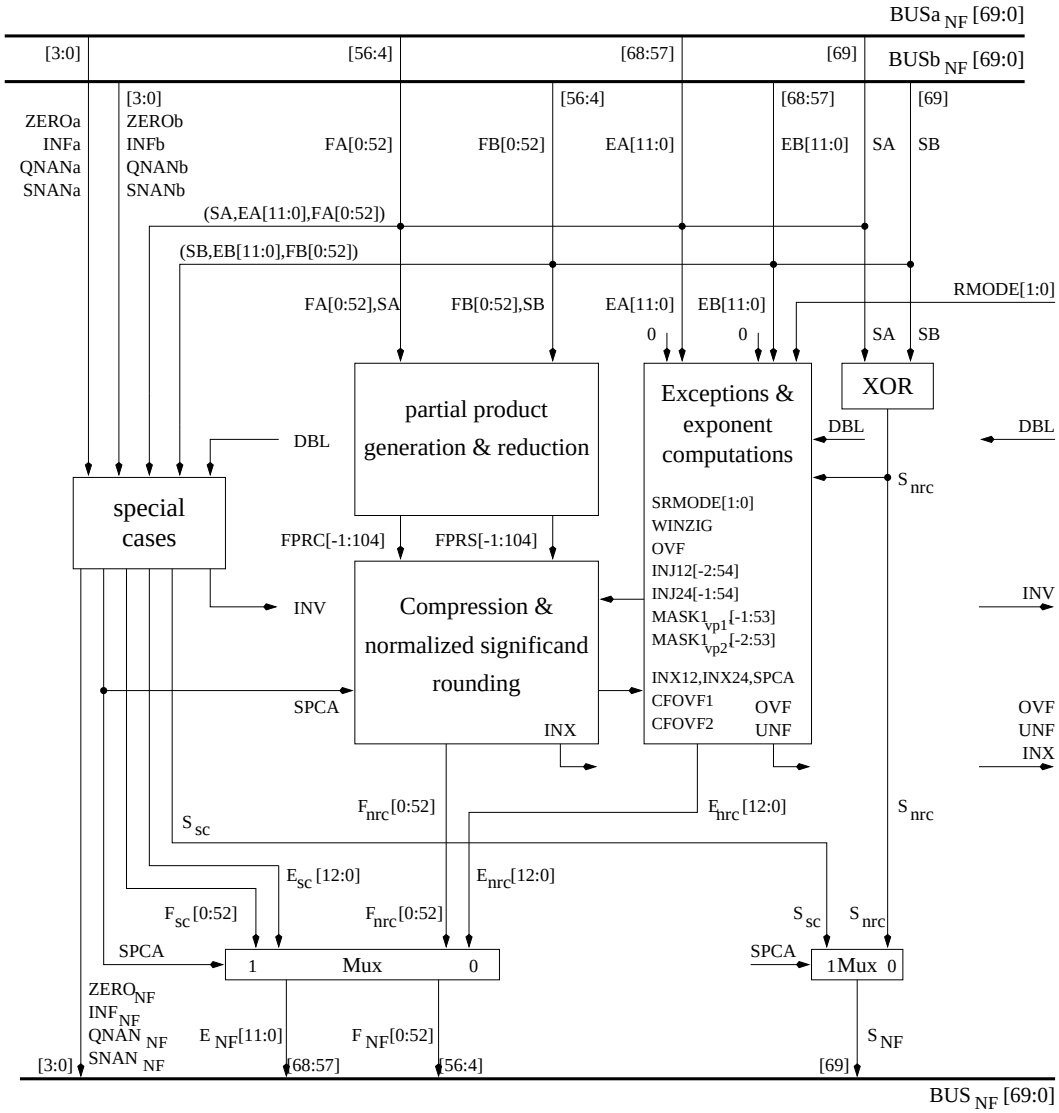

Figure 4.27: Block digram of the multiplication unit III.

According to lemma 2.8, the rounding function $nround$ can be composed of the four steps of an unbounded normalization shift, normalized significand rounding, another unbounded normalization shift and exponent rounding:

$$\begin{aligned}(s_{nrc}, e_{nrc}, f_{nrc}) &= nround(s_{pr}, e_{pr} + wec, fpr) && (4.265)\\ &= exp\_rnd_{mode \star s_{pr}}(\eta(n\_sig\_rnd_{mode \star s_{pr}}(\eta(s_{pr}, e_{pr} + wec, fpr))))\end{aligned} \quad (4.266)$$

Like in the addition computations II and III, we partition the discussion of the rounding computations into two steps. After the computation of a first step according to

$$(s_{nr1}, e_{nr1} + wec, f_{nr1}) = \eta(n\_sig\_rnd_{mode \star s_{pr}}(\eta(s_{pr}, e_{pr} + wec, fpr))), \quad (4.267)$$

the final result can obviously be computed by the exponent rounding

$$(s_{nrc}, e_{nrc}, f_{nrc}) = exp\_rnd_{mode \star s_{pr}}(s_{nr1}, e_{nr1} + wec, f_{nr1}). \quad (4.268)$$

Because for both, single and double precision, $fa, fb \in [1, 2-2^{-52}]$, the exact significand product is in the range $fpr = fa \cdot fb \in [1, 4-2^{-51}]$. Thus, with the definition of normalized significand rounding by $n\_sig\_rnd_{srmode}(s, e, f) = (s, e, rnd_{srmode,vp}(f))$ and the variable rounding position $vp = p - 1 - max\{0, e_{min} - e\}$ according to definition 2.9, the normalization shift can be simplified and combined with the rounding separately for the two cases: $fpr \in [1, 2[$ and $fpr \in [2, 4[$ by

$$(s_{nr1}, e_{nr1}, f_{nr1}) = \eta(n\_sig\_rnd_{mode \star s_{pr}}(\eta(s_{pr}, e_{pr}, fpr))) \tag{4.269}$$

$$= \begin{cases} \eta(n\_sig\_rnd_{mode \star s_{pr}}(\eta(s_{pr}, e_{pr}, fpr))) & \text{if } fpr \in [1, 2[\\ \eta(n\_sig\_rnd_{mode \star s_{pr}}(\eta(s_{pr}, e_{pr}, fpr))) & \text{if } fpr \in [2, 4[\end{cases} \tag{4.270}$$

$$= \begin{cases} \eta(s_{pr}, e_{pr}, rnd_{mode \star s_{pr}, vp1}(fpr)) & \text{if } fpr \in [1, 2[\\ \eta(s_{pr}, e_{pr} + 1, rnd_{mode \star s_{pr}, vp2}(fpr/2)) & \text{if } fpr \in [2, 4[, \end{cases} \tag{4.271}$$

where according to definition 2.9, the variable rounding positions $vp1, vp2$ are given by

$$vp1 = p-1-max\{0, e_{min}-(e_{pr}+wec)\} \tag{4.272}$$

$$vp2 = p-1-max\{0, e_{min}-(e_{pr}+wec+1)\}. \tag{4.273}$$

In the above formulae, the rounding positions $vp1$ and $vp2$ could be in a very large range, namely because floating-point results $x \in \mathcal{FP}_{n,p}$ could even have the magnitude $2^{2e_{min}-2p+2}$ (see section 2.4.2), the variable significand rounding positions could be in the range $vp1, vp2 \in [e_{min}-p-1 : p-1]$. Based on the fact, that the significand product fpr is smaller than 4, the significand rounding can be simplified for rounding positions $vp1, vp2 < -2$. In these cases we know for sure, that the rounding operand has a magnitude smaller than half of the smallest representable number, so that in these cases the rounded result has to be selected between ± 0 or $\pm x_{min}$. By a separate selection for these small results, the ranges for the variable rounding positions in the remaining cases is reduced to $[-2 : p-1]$ and the range of the rounded significands is limited to $[0, 4]$. For these reasons, the rounding computations and the computation of the unbounded normalization shift can be simplified. This will be further discussed after the next lemma.

Lemma 4.27 *With the definition of the condition* WINZIG*, that detects results with very small exponents by*

$$\begin{aligned} \text{WINZIG} &\iff (e_{pr} + wec \leq e_{min} - 3 - p + 1) \\ &\iff (e_{pr} \leq e_{min} - 3 - p + 1) \ AND \ \overline{\text{UNF\_EN}} \end{aligned}$$

the rounded result can be selected by

$$(s_{nr1}, e_{nr1} + wec, f_{nr1}) = \begin{cases} (s_{pr}, e_{min} - p + 1, 0) & \text{if WINZIG} \wedge \overline{\text{SR\_MODE}[1]} \\ (s_{pr}, e_{min} - p + 1, 1) & \text{if WINZIG} \wedge \text{SR\_MODE}[1] \\ \eta(s_{pr}, e_{pr} + wec, rnd_{mode \star s_{pr}, vp1}(fpr)) & \text{if } \overline{\text{WINZIG}} \wedge fpr \in [1, 2[\\ \eta(s_{pr}, e_{pr} + 1 + wec, rnd_{mode \star s_{pr}, vp2}(\frac{fpr}{2})) & \text{if } \overline{\text{WINZIG}} \wedge fpr \in [2, 4[. \end{cases}$$

Proof: For the proof we distinguish between the cases: (a) WINZIG = 1; and (b) WINZIG = 0.

(a) For WINZIG = 1, we have $(e_{pr} + wec \leq e_{min} - 3 - p + 1)$, so that because of $fpr < 4$, the magnitude of the exact product $val(0, e_{pr} + wec, fpr)$ is smaller than $x_{min}/2 = 2^{e_{min}-p}$. Because we deal with non-zero operands, also the exact product is non-zero, so that the magnitude of the exact product is in the range $0 < val(0, e_{pr} + wec, fpr) < x_{min}/2$.

Thus, the nearest representable numbers to the exact product are $0 = val(s_{pr}, e_0, 0)$ and $(-1)^{s_{pr}} x_{min} = val(s_{pr}, e_{min} - p + 1, 1)$, so that according to the IEEE rounding definition in section 2.3.1 the exact product is rounded to $(s_{pr}, e_{min} - p + 1, 0)$ in rounding mode $srmode \in \{RZ, RNE\}$ and to $(s_{pr}, e_{min} - p + 1, 1)$ in rounding mode $srmode = RI$. Because the rounding mode RI is signaled by the bit SR_MODE[1], this agrees with the first two lines of the rounding formula in this lemma.

(b) For WINZIG $= 0$, the rounding equations are copied identically from equation 4.271. For WINZIG $= 0$, we have $e_{pr} + wec \geq e_{min} - 2 - p + 1$. From this condition on the exponent $e_{pr} + wec$, it follows, that the variable rounding positions $vp1$ and $vp2$ are limited to the ranges $vp1' \in [-2 : p-1]$ and $vp2' \in [-1 : p-1]$. These conditions can be used for the rounding implementation. □

Because the above selection of the rounded result is simple for WINZIG $= 1$, we focus on the computation of the cases for WINZIG $= 0$ in the following. For this purpose, we introduce the notation:

$$fprnd12 = rnd_{mode \star s_{pr}, vp1'}(fpr) \tag{4.274}$$

$$fprnd24 = rnd_{mode \star s_{pr}, vp2'}(fpr/2)) \tag{4.275}$$

$$(s_{prnd}, e_{prnd}, fprnd) = \begin{cases} \eta(s_{pr}, e_{pr}, fprnd12) & \text{if } fpr \in [1, 2[\\ \eta(s_{pr}, e_{pr} + 1, fprnd24) & \text{if } fpr \in [2, 4[. \end{cases} \tag{4.276}$$

With this notation the result of first step (equation 4.267) can be written as:

$$(s_{nr1}, e_{nr1} + wec, f_{nr1}) = \begin{cases} (s_{pr}, e_{min} - p + 1, < \overline{\text{SR\_MODE}[1]} >) & \text{if WINZIG} \\ (s_{pr}, e_{prnd} + wec, fprnd) & \text{otherwise.} \end{cases} \tag{4.277}$$

Because in the rounding computations for $fprnd$ we can use that WINZIG $= 0$, the ranges of the variable rounding positions $vp1$ and $vp2$ for the computation of $fprnd$ are limited to $vp1 \in [-2 : p-1]$ and $vp2 \in [-1 : p-1]$ according to the proof of case (b) in the previous lemma. To indicate that we only have to consider these limited rounding position ranges, we write $vp1'$ and $vp2'$ for the rounding positions with limited ranges and have $vp1' = vp1$ for $vp1 \in [-2 : p-1]$ and $vp2' = vp2$ for $vp2 \in [-1 : p-1]$. From $fpr \in [1, 4[$ and from the ranges of the variable rounding positions $vp1'$ and $vp2'$, it follows that the rounded significands $fprnd12$ and $fprnd24$ are bounded by $fprnd12 \in [0, 4]$ and $fprnd24 \in [0, 2]$, and thus, they can be represented according to $fprnd12 = <\text{FPRND12}[-2 : 52]>_{neg}$ and $fprnd24 = <\text{FPRND24}[-1 : 52]>_{neg}$.

The selection and computations in the two cases of equation 4.276 can be simplified by selecting the upper choice only for $fprnd12 \in [0, 2[$ and the other choice for all other cases. In this way, the selection condition is based on the rounded significand value $fprnd12$ instead of the value of the unrounded significand product fpr. The following lemma shows, that we do not make a mistake by this substitution, but that the new ranges, for that we consider $fprnd12$ and $fprnd24$, allow to simplify the normalization shifts, that are required after the rounding.

Lemma 4.28 *Equation 4.276 can be simplified to*

$$(s_{prnd}, e_{prnd}, fprnd) = \begin{cases} (s_{pr}, e_{pr}, fprnd12) & \textit{if } fprnd12 < 2 \\ post\_norm(s_{pr}, e_{pr} + 1, fprnd24) & \textit{otherwise.} \end{cases}$$

Proof: We divide the proof of the lemma into two steps: In step (a) we show, that the values on both sides in the equation of the lemma are the same. Then, we show in step

(b), that the unbounded normalization shifts from equation 4.276 can be replaced by a post-normalization shift respectively by no shift for the two cases.

(a) The normalization shifts do not change the values of the factorings. Thus, we only have to show the equality of the selected values

$$\begin{aligned} val(s_{pr}, e_{prnd}, fprnd) &= \begin{cases} val(s_{pr}, e_{pr}, fprnd12) & \text{if } fpr \in [1,2[\\ val(s_{pr}, e_{pr}+1, fprnd24) & \text{otherwise.} \end{cases} \\ &= \begin{cases} val(s_{pr}, e_{pr}, fprnd12) & \text{if } fprnd12 < 2 \\ val(s_{pr}, e_{pr}+1, fprnd24) & \text{otherwise.} \end{cases} \end{aligned}$$

Because the equality is obvious, if the conditions $(fprnd12 < 2)$ and $(fpr \in [1,2[)$ have the same value, we only have to consider the cases, where: (a.i) $(fprnd12 \geq 2)$ and $(fpr \in [1,2[)$; and (a.ii) $(fprnd12 < 2)$ and $(fpr \in [2,4[)$. Thus, to show the above equality, it suffices to show that $fprnd24 = fprnd12/2$ in the cases (a.i) and (a.ii).
(a.i) In the computation of $fprnd12$, we have to consider the rounding positions $vp1' \in [-2:p-1]$ and in the computation of $fprnd24$, we have to consider the rounding positions $vp2' \in [-1:p-1]$. For rounding positions $vp1' \in [-1:p-1]$, it follows from $(fpr \in [1,2[)$, that $(fprnd12 \leq 2)$. Thus, in case (a.i) we either have $fprnd12 = 2$, or $vp1' = -2$ and thus $fprnd12 = 4$ (in this case $vp2' = -1$).

From the definitions of the variable rounding positions $vp1'$ and $vp2'$ (see equations 4.272-4.273), it follows that $vp2' \in \{vp1', vp1'+1\}$, so that we always have $vp2' \leq vp1'+1$. The rounded significand $fprnd12$ can be written as a rounding function of $fpr/2$ with rounding position $vp1'+1$:

$$fprnd12 = rnd_{srmode,vp1'}(fpr) = 2 \cdot rnd_{srmode,vp1'+1}(fpr/2).$$

Thus, because of $vp2' \leq vp1'+1$, the computation of the rounded significand $fprnd24 = rnd_{srmode,vp2'}(fpr/2)$ can be interpreted as a second gradual rounding step on the significand $fprnd12/2$ at the rounding position $vp2'$. We now consider $fprnd12 = 2$ and $fprnd12 = 4$, which are the two possible values of $fprnd12$ for case (a.i). Because $fprnd12/2 = 1$ is already a multiple of $2^{-vp2'}$ for $vp2' \in [0:p-1]$, we get in this case also for the second gradual rounding step $fprnd24 = 1 = fprnd12/2$, and because $fprnd12/2 = 2$ is already a multiple of $2^{-vp2'}$ for $vp2' = -1$, we get in this case also for the second gradual rounding step $fprnd24 = 2 = fprnd12/2$. This completes the proof of case (a.i)
(a.ii) In the computation of $fprnd12$, the rounding position could be in the range $vp1' \in [-2:p-1]$. Because we assume $fpr \in [2,4[$, the rounded significand $fprnd12$ can not become smaller than 2 for the rounding positions $vp1' \in [-1:p-1]$. Only for the rounding position $vp1' = -2$, the significand $fprnd12$ could become smaller than 2, and the only possible case for this is $fprnd12 = 0$. For $vp1' = -2$, we have $vp2' = -1$ and it follows from

$$0 = fprnd12 = rnd_{srmode,-2}(fpr) = 2 \cdot rnd_{srmode,-1}(fpr/2) = 2 \cdot fprnd24 = 0,$$

that also in case (a.ii) we have $fprnd12/2 = fprnd24$, as required.

(b) For the proof of part (b) we distinguish between the two cases: (b.i) $fprnd12 < 2$; and (b.ii) $fprnd12 \geq 2$.
(b.i) For $fprnd12 < 2$, the upper choice is selected. For this choice, we have to consider the rounding positions $vp1' \in [-2:p-1]$ in the computation of $fprnd12$. For rounding positions $vp1' \geq 0$, it follows from $fpr \in [1,4[$, that $fprnd12 \geq 1$, so that the resulting factoring is already normalized in these cases and the additional normalization shift can

be neglected. For the remaining rounding positions $vp1' \in \{-1,-2\}$, it follows from $fpr \in [1,4[$, that $fprnd12 \in \{0,2,4\}$. Among these cases, only for the result 0, the condition for case (b.i) is given and the upper choice is selected. Because the unbounded normalization shift is defined to compute the identity function for factorings of zero, the normalization shift can be neglected for all rounding positions, that have to be considered. (b.ii) For $fprnd12 \geq 2$, the factoring $(s_{pr}, e_{pr}+1, fprnd24)$ is selected. For this choice, we have to consider the rounding positions $vp2' \in [-1:p-1]$ in the computation of $fprnd24$. From this range of rounding positions with $fpr/2 \in [0.5,2[$, it follows that $fprnd24 \leq 2$ and because $fprnd12/2 \geq 1$, it follows that $fprnd24 \in [1,2]$. Because a post-normalization shift (see definition 2.11) normalizes factorings with significands in the range $[1,2]$, a post-normalization shift suffices to normalize the factoring $(s_{pr}, e_{pr}+1, fprnd24)$, so that the unbounded normalization shift can be replaced by a post-normalization shift in the case (b.ii). Thus, the conclusion of step (a), case (b.i) and case (b.ii) is, that

$$(s_{prnd}, e_{prnd}, fprnd) = \begin{cases} (s_{pr}, e_{pr}, fprnd12) & \text{if } fprnd12 < 2 \\ post\_norm(s_{pr}, e_{pr}+1, fprnd24) & \text{otherwise,} \end{cases}$$

as required by the lemma. □

Definition 4.11 *We define two significand overflow conditions* CFOVF1 *and* CFOVF2*:*

$$\begin{aligned} \text{CFOVF1} &\iff (fprnd12 \geq 2) \\ &\iff \text{FPRND12}[-2] \vee \text{FPRND12}[-1] \\ \text{CFOVF2} &\iff (fprnd24 = 2) \\ &\iff \text{FPRND24}[-1] \end{aligned}$$

With this definition of the significand overflow conditions CFOVF1 and CFOVF2 and with the definition of the post-normalization shift (see equation 2.11), the equation from lemma 4.28 can obviously be written as

$$(s_{prnd}, e_{prnd}, fprnd) = \begin{cases} (s_{pr}, e_{pr}, fprnd12) & \text{if } \overline{\text{CFOVF1}} \\ (s_{pr}, e_{pr}+1, fprnd24) & \text{if CFOVF1 AND } \overline{\text{CFOVF2}} \\ (s_{pr}, e_{pr}+2, 1) & \text{if CFOVF1 AND CFOVF2} \end{cases} \quad (4.278)$$

Lemma 4.29 *For exponents $e_{pr} + wec \geq e_{min}$, the condition* CFOVF2 *can not be fulfilled:*

$$(e_{pr} + wec \geq e_{min}) \quad \Longrightarrow \quad \overline{\text{CFOVF2}}.$$

Proof: From $(e_{pr} + wec \geq e_{min})$ it follows, that the variable rounding positions $vp1$ and $vp2$ are fixed to $vp1 = vp2 = p-1$. Because $fa, fb \leq 2 - 2^{-p+1}$ and thus $fpr/2 < 2 - 2^{-p+1}$, it follows from the rounding position $vp2 = p-1$, that the rounded significand $fprnd24 < 2$. Therefore, we get as required CFOVF2 $= 0$. □

We postpone a detailed description of the rounding implementations for $fprnd12$ and $fprnd24$, and consider the description of the exponent rounding and the exponent wrapping according to the second computation step from equation 4.268 in the following. Because the conditions WINZIG and OVF can not both be fulfilled at the same time and no exponent wrapping is required for UNF_EN $= 0$, the exponent rounding selection from

equation 4.268 can be written in combination with the definition of exponent rounding (see equation 2.12) and with equation 4.277 as

$$(s_{nrc},e_{nrc},f_{nrc}) = \begin{cases} (s_{pr}, e_{max}, f_{max}) & \text{if OVF} \wedge \overline{\text{OVF\_EN}} \wedge \overline{\text{OR(SR\_MODE[1:0])}} \\ (s_{pr}, e_{\infty}, f_{\infty}) & \text{if OVF} \wedge \overline{\text{OVF\_EN}} \wedge \text{OR(SR\_MODE[1:0])} \\ (s_{pr}, e_{min}-p+1, <\overline{\text{SR\_MODE[1]}}>) & \text{if WINZIG} \\ (s_{pr}, e_{prnd}+wec, fprnd) & \text{otherwise.} \end{cases} \quad (4.279)$$

We integrate the selection of the $\pm x_{max}$ and $\pm\infty$ results with the selection of the ± 0 and $\pm x_{min}$ results in the factoring $(s_{pr}, e_{sel}, f_{sel})$ by the selection:

$$(s_{pr}, e_{sel}, f_{sel}) = \begin{cases} (s_{pr}, e_{max}, f_{max}) & \text{if } \overline{\text{WINZIG}} \wedge \overline{\text{OR(SR\_MODE[1:0])}} \\ (s_{pr}, e_{\infty}, f_{\infty}) & \text{if } \overline{\text{WINZIG}} \wedge \text{OR(SR\_MODE[1:0])} \\ (s_{pr}, e_{min}-p+1, <\overline{\text{SR\_MODE[1]}}>) & \text{if WINZIG} \end{cases}$$

so that the factoring $(s_{nrc},e_{nrc},f_{nrc})$ can be selected by:

$$(s_{nrc}, e_{nrc}, f_{nrc}) = \begin{cases} (s_{pr}, e_{sel}, f_{sel}) & \text{if WINZIG OR (OVF} \wedge \overline{\text{OVF\_EN}}) \\ (s_{pr}, e_{prnd}+wec, fprnd) & \text{otherwise.} \end{cases} \quad (4.280)$$

This already describes, how the significand f_{nrc} is selected. For the computation of the exponent we additionaly have to consider the implementation of the exponent wrapping.

For the computation of the exponent wrapping, we predict the wrapping exponent constant wec by the sign of the exponent $e_{pr} = <\text{EPR}[12:0]>_2$ (which is EPR[12]) similar to the computation in the addition unit III according to

$$pwec = \begin{cases} +\alpha & \text{if EPR[12]} \\ -\alpha & \text{otherwise.} \end{cases}$$

so that with the definition of the condition EWRAP, that signals the requirement for exponent wrapping by

$$\text{EWRAP} \iff (\text{UNF} \wedge \text{UNF\_EN}) \text{ OR } (\text{OVF} \wedge \text{OVF\_EN}), \quad (4.281)$$

the exponent wrapping can be included into equation 4.278 by

$$e_{prnd}+wec = \begin{cases} e_{pr} & \text{if } \overline{\text{EWRAP}} \text{ AND } \overline{\text{CFOVF1}} \\ e_{pr}+1 & \text{if } \overline{\text{EWRAP}} \text{ AND CFOVF1 AND } \overline{\text{CFOVF2}} \\ e_{pr}+2 & \text{if } \overline{\text{EWRAP}} \text{ AND CFOVF1 AND CFOVF2} \\ e_{pr}+pwec & \text{if EWRAP AND } \overline{\text{CFOVF1}} \\ e_{pr}+1+pwec & \text{if EWRAP AND CFOVF1} \end{cases} \quad (4.282)$$

Note, that the exponent $e_{pr}+2+pwec$ does not have to be considered in this equation, because $e_{pr}+wec \geq e_{min}$ for EWRAP $= 1$ (see corollary 2.10) and because of lemma 4.29. Based on the equations 4.280 and 4.282 the computation of the exponent e_{nrc} is implemented by the following six selections:

$$eopi = \begin{cases} e_{sel} & \text{if (WINZIG} \vee \overline{\text{OVF\_EN}}) \\ ewi = e_{pr}+1+pwec & \text{otherwise} \end{cases}$$

$$eci = \begin{cases} eprii = e_{pr}+2 & \text{if CFOVF2} \\ epri = e_{pr}+1 & \text{otherwise} \end{cases}$$

$$
\begin{aligned}
eni &= \begin{cases} eopi & \text{if } (\text{WINZIG} \vee \text{OVF}[1] \vee (\text{UNF}[1] \wedge \text{UNF\_EN})) \\ eci & \text{otherwise} \end{cases} \\
eop &= \begin{cases} e_{sel} & \text{if } (\text{WINZIG} \vee \overline{\text{OVF\_EN}}) \\ ew = e_{pr} + pwec & \text{otherwise} \end{cases} \\
en &= \begin{cases} eop & \text{if } (\text{WINZIG} \vee \text{OVF}[0] \vee (\text{UNF}[0] \wedge \text{UNF\_EN})) \\ e_{pr} & \text{otherwise} \end{cases} \\
e_{nrc} &= \begin{cases} eni & \text{if CFOVF1} \\ en & \text{otherwise,} \end{cases}
\end{aligned}
$$

where OVF[1] and UNF[1] indicate the case of an overflow resp. underflow under the assumption that CFOVF1 = 1 and OVF[0] and UNF[0] indicate the case of an overflow resp. underflow under the assumption that CFOVF1 = 0.

Because the exponent eni is selected only for CFOVF1 = 1, we assume in the selection for eni that CFOVF1 = 1. Therefore, we use the signals UNF[1] and OVF[1] instead of UNF and OVF in this selection. Accordingly, we use UNF[0] and OVF[0] instead of UNF and OVF in the selection for en. The condition in the seletions for eop and $eopi$ is based on

$$\text{OVF} \wedge \overline{\text{OVF\_EN}} \vee \text{EWRAP} \Longleftrightarrow \text{OVF} \vee (\text{UNF} \wedge \text{UNF\_EN}).$$

This completes the description of the selections for the exponent e_{nrc}.

In the following, we consider the rounding implementation for the rounded significands $fprnd12$ and $fprnd24$. The computation of $fprnd12$ and $fprnd24$ is based on the injection-based rounding mode reduction (see section 2.5.2). To be able to use injection-based rounding, we have to consider the rounding modes RZ, RNU, RI first and to correct the rounded result in the case of the rounding mode RNE by the additional L-bit fix at the least significant bit position of the significand (see section 2.3.2).

For the rounding of a significand, which is an integral multiple of 2^{-105}, at a bit position $vp < 105$ in the rounding mode $srmode \in \{RZ, RNU, RI\}$, the rounding injection inj_{vp} is defined by

$$
\begin{aligned}
inj_{vp} &= < \text{INJ}_{vp}[-2:105] >_{2neg} \\
&= \begin{cases} 0 & \text{if } srmode = RZ \\ 2^{-vp-1} & \text{if } srmode = RNU \\ 2^{-vp} - 2^{-105} & \text{if } srmode = RI \end{cases}
\end{aligned}
$$

so that according to lemma 2.18 for $srmode \in \{RZ, RNU, RI\}$, the injection-based rounding of a significand f at the position vp can be written as

$$rnd_{srmode,vp}(f) = rnd_{RZ,vp}(f + inj_{vp}).$$

For our rounding computations we have to generate the injections $inj_{vp1'}$ and $inj_{vp2'}$ according to rounding positions $vp1'$ and $vp2'$. We denote the injected significands by

$$
\begin{aligned}
finj12 &= fpr + inj_{vp1'} \\
finj24 &= fpr + inj_{vp2'}.
\end{aligned}
$$

By the truncation of $finj12$ and $finj24$ after bit position $vp1'$ resp. $vp2'$, we get the rounded significands, that consider the rounding modes $srmode \in \{RZ, RNU, RI\}$:

$$
\begin{aligned}
fprnd12' &= rnd_{RZ,vp1'}(finj12) \\
fprnd24' &= rnd_{RZ,vp2'}(finj24).
\end{aligned}
$$

We then get the required rounded significands $fprnd12$ and $fprnd24$ from $fprnd12'$ and $fprnd24'$ by an additional L-bit-fix for the rounding mode RNE at significand position $vp1'$ resp. $vp2'$. Because we only have to consider $fprnd12 < 8$ and fpr is an integral multiple of 2^{-104} for both single and double precision, it suffices to consider the bit positions $[-2:104]$ in the binary representations of the values $inj_{vp1'}$ and $finj12$, and because we only have to consider $fprnd24 < 4$ and $fpr/2$ is an integral multiple of 2^{-105} for both single and double precision, it suffices to consider the bit positions $[-1:105]$ in the binary representations of the values $inj_{vp2'}$ and $finj24$.

Based on the above notations we overview the computation steps, that are required for the computation of $fprnd12$ and $fprnd24$ in the significand path:

(A) By the partial product generation and reduction, a carry-save representation of the exact significand product fpr is computed. Because in this case no rounding injection is added during the reduction, we can use the implementations of step (A) from the multiplication unit I (both, the half-sized version, which is depicted in figure 4.22, and the full-sized version, which is depicted in figure 4.21).

(B) Step (B) contains the compression, the IEEE rounding and post-normalization shift of the significand product fpr from one of its carry-save representations that we get from step (A). The rounding for the computation of $fprnd12$ and $fprnd24$ in the rounding modes $srmode \in \{RZ, RNE, RI\}$ is computed in two steps:

(I) Computation of $fprnd12'$ and $fprnd24'$ considering the rounding modes $srmode \in \{RZ, RNU, RI\}$ by injection-based rounding with the steps:

(1) Generation of the injections $inj_{vp1'}$ and $inj_{vp2'}$ and addition with the carry-save representation of fpr by a full-adder line and a carry-look-ahead adder that implement:

$$finj12 = < \text{FINJ}12[-2:105] >_{neg} = fpr + inj_{vp1'} \quad (4.283)$$

$$finj24 = < \text{FINJ}24[-1:105] >_{neg} = fpr + inj_{vp2'}. \quad (4.284)$$

(2) Truncation of $finj12$ after bit position $vp1'$ and of $finj24$ after bit position $vp2'$. Because the truncation position is not fixed in this case, the truncation is more complicated to be computed than in the previous section and has to be considered separately.

$$fprnd12' = rnd_{RZ,vp1'}(finj12) \quad (4.285)$$

$$= < \text{FINJ}12[-2:vp1'] >_{neg} \quad (4.286)$$

$$fprnd24' = rnd_{RZ,vp2'}(finj24) \quad (4.287)$$

$$= < \text{FINJ}24[-1:vp1'] >_{neg} \quad (4.288)$$

(II) Computation of the rounded significands $fprnd12$ and $fprnd24$, that consider the rounding modes $srmode \in \{RZ, RNE, RI\}$ from $fprnd12'$ and $fprnd24'$, that considered the rounding modes $srmode \in \{RZ, RNU, RI\}$ by implementing the L-bit-fix for the rounding mode RNE.

The significand position of the L-bit is $vp1'$ resp. $vp2'$. This bit has to be pulled down if the L-bit-fix condition is fulfilled, namely iff the number lies exactly between two representable rounding results. Because in this case the injected

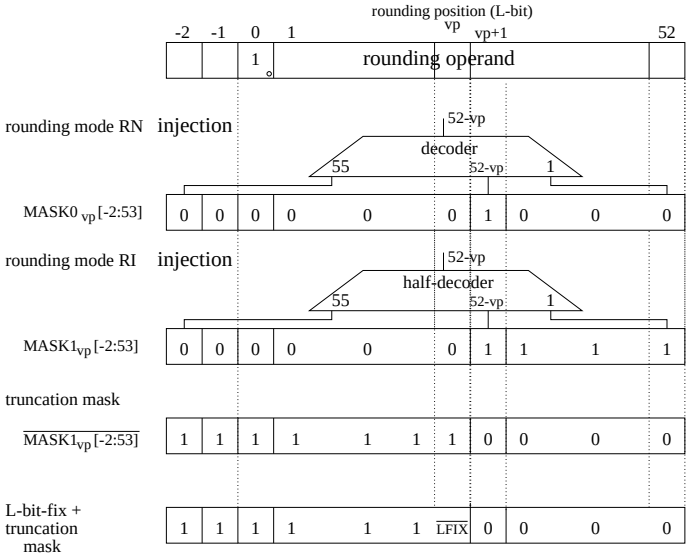

Figure 4.28: Generation of the injections for the variable rounding position $vp \in [-2:51]$.

significands already contain the injections $2^{-vp1'-1}$ resp. $2^{-vp2'-1}$, the L-bit-fix conditions are given by:

$$\begin{aligned} \text{LFIX12} &= \text{SR\_MODE}[0] \text{ AND } (\text{FINJ12}[vp1'+1:104] = 0^{104-vp1'}) \\ \text{LFIX24} &= \text{SR\_MODE}[0] \text{ AND } (\text{FINJ24}[vp2'+1:105] = 0^{105-vp2'}). \end{aligned}$$

Because step (A) is implemented like in the multiplication unit I, no implementation details have to be added for this step. The missing implementation details for the computation step (B) are described in the following:

(B) For the implementation of step (B.I.1), the generation of the rounding injections $inj_{vp1'}$ and $inj_{vp2'}$ has to be described. The binary representations of these injections are composed from two parts, a fixed mask that accounts for results with values of normalized numbers from $\mathcal{NOR}_{n,p}$, in which case the significand has to be rounded at the position $p-1$, and a variable mask, that is adjusted corresponding to the variable rounding position for results with values of denormalized numbers from $\mathcal{DEN}_{n,p}$. Moreover, we distinguish between a fixed injection mask for the rounding mode RNU, which we call FIXMASK0, and a fixed injection for the rounding mode RI, which we call FIXMASK1. For the cases, where the significand rounding position is different from $p-1$, the binary representations of the rounding injections is generated with the help of a decoder for the rounding mode RI and a half-decoder for the rounding mode RNU. These decoders account for the correcting terms $max\{0, e_{min} - (e_{pr} + wec)\}$ and $max\{0, e_{min} - (e_{pr} + wec + 1)\}$ in the equations of the rounding positions $vp1'$ and $vp2'$. For the rounding modes RNU and RI the generation of these variable rounding injections is illustrated in figure 4.28 considering a variable rounding position $vp \in [-2:51]$. Moreover, this figure depicts how the masks that are used to generate the binary representations of the injections could also be used for the truncation and L-bit-fix computations, that are required in steps (B.I.2) and (B.II).

A formal description of the injection generations for $\text{INJ}_{vp1'}$ and $\text{INJ}_{vp2'}$ is given by the following lemma. In this lemma, several different masks are defined. In general, we append a $'0'$ to the names of masks, that are used to generate injections for the rounding

mode RNU. To the names of the corresponding masks for the rounding mode RI, we append a $'1'$:

Lemma 4.30 *With the condition*

$$\text{VRTINY} \iff (e_{pr} + 1 - e_{min} < 0) \wedge \overline{\text{UNF\_EN}}$$

and the computation of

$$\begin{aligned}
\text{FIXMASK0}[-2:53] &= (0^{26}, \overline{\text{DBL}}, 0^{28}, \text{DBL}) \\
\text{FIXMASK1}[-2:53] &= (0^{26}, \overline{\text{DBL}}^{29}, 1) \\
varterm &= <\text{VARTERM}[12:0]>_2 \\
&= \begin{cases} e_{min} - e_{pr} - 1 & \textit{if } \text{DBL} \\ e_{min} - e_{pr} - 1 + 29 & \textit{otherwise.} \end{cases} \\
\text{VARMASK0}[-2:52] &= \text{DECO}(\text{VARTERM}[5:0])[55:1] \\
\text{VARMASK1}[-2:52] &= \text{HDEC}(\text{VARTERM}[5:0])[55:1] \\
\text{MASK0}_{vp1'}[-2:53] &= \begin{cases} (\text{VARMASK0}[-2:52], 0) & \textit{if } \text{VRTINY} \\ \text{FIXMASK0}[-2:53] & \textit{otherwise.} \end{cases} \\
\text{MASK0}_{vp2'}[-1:53] &= \begin{cases} \text{VARMASK0}[-2:52] & \textit{if } \text{VRTINY} \\ \text{FIXMASK0}[-1:53] & \textit{otherwise.} \end{cases} \\
\text{MASK1}_{vp1'}[-2:53] &= \begin{cases} (\text{VARMASK1}[-2:52], 1) & \textit{if } \text{VRTINY} \\ \text{FIXMASK1}[-2:53] & \textit{otherwise.} \end{cases} \\
\text{MASK1}_{vp2'}[-1:53] &= \begin{cases} \text{VARMASK1}[-2:52] & \textit{if } \text{VRTINY} \\ \text{FIXMASK1}[-1:53] & \textit{otherwise.} \end{cases}
\end{aligned}$$

the rounding injections can be generated by

$$\begin{aligned}
\text{INJ}_{vp1'}[-2:105] &= \begin{cases} (\text{MASK1}_{vp1'}[-2:53], 1^{52}) & \textit{if } \text{SR_MODE}[1] \\ (\text{MASK0}_{vp1'}[-2:53], 0^{52}) & \textit{if } \text{SR_MODE}[0] \\ 0^{108} & \textit{otherwise.} \end{cases} \\
\text{INJ}_{vp2'}[-2:105] &= \begin{cases} (0, \text{MASK1}_{vp2'}[-1:53], 1^{52}) & \textit{if } \text{SR_MODE}[1] \\ (0, \text{MASK0}_{vp2'}[-1:53], 0^{52}) & \textit{if } \text{SR_MODE}[0] \\ 0^{108} & \textit{otherwise.} \end{cases}
\end{aligned}$$

Proof: Based on the value of the condition VRTINY, the definitions of the variable rounding positions $vp1'$ and $vp2'$ can be split into

$$\begin{aligned}
vp1' &= \begin{cases} p - 1 - e_{min} + e_{pr} & \text{if VRTINY} \\ p - 1 & \text{otherwise} \end{cases} \\
vp2' &= \begin{cases} p - 1 - e_{min} + e_{pr} + 1 & \text{if VRTINY} \\ p - 1 & \text{otherwise,} \end{cases}
\end{aligned}$$

so that the injections can be generated separately in a fixed part considering rounding position $p - 1$ for the case $\text{VRTINY} = 0$ and in a variable part for the case $\text{VRTINY} = 1$. In this way we use in particular, that for the case $\text{VRTINY} = 0$, we have $vp1' = vp2' = p - 1$. In the following we proove the lemma separately for the three rounding modes RZ, RNU and RI:

In the rounding mode RZ, the injections are defined to be $inj_{vp1'} = inj_{vp2'} = 0$ for both the fixed and the variable case. Because the rounding mode RZ is encoded with $\text{SR\_MODE}[0] = \text{SR\_MODE}[1] = 0$, we get by the selection from the lemma

$$\text{INJ}_{vp1'}[-2:105] = \text{INJ}_{vp2'}[-2:105] = 0^{108},$$

as required by the definition for the rounding mode RZ.

In the rounding mode RNU, the injections are defined by

$$\begin{aligned} inj_{vp1'} &= 2^{-vp1'-1} \\ &= <\text{INJ}_{vp1'}[-2:105]>_{neg} \\ &= <(0^{vp1'+3}, 1, 0^{104-vp1'})>_{neg} \qquad (4.289) \\ inj_{vp2'} &= 2^{-vp2'-1} \\ &= <\text{INJ}_{vp2'}[-2:105]>_{neg} \\ &= <(0^{vp2'+3}, 1, 0^{104-vp2'})>_{neg}. \qquad (4.290) \end{aligned}$$

We distinguish between the cases: (a) $\text{VRTINY} = 0$ and (b) $\text{VRTINY} = 1$:

(a) For $\text{VRTINY} = 0$, we have $vp1' = vp2' = p-1$, so that by definition $\text{INJ}_{vp1'}[-2:105] = \text{INJ}_{vp2'}[-2:105] = (0^{p+2}, 1, 0^{105-p})$. Because the rounding mode RNU is encoded by $\text{SR\_MODE}[0] = 1$, by the selection from the lemma

$$\begin{aligned} \text{INJ}_{vp1'}[-2:105] = \text{INJ}_{vp2'}[-2:105] &= (\text{FIXMASK0}[-2:53], 0^{52}) \\ &= \begin{cases} (0^{55}, 1, 0^{52}) & \text{if DBL} \\ (0^{26}, 1, 0^{81}) & \text{otherwise} \end{cases} \\ &= (0^{p+2}, 1, 0^{105-p}) \end{aligned}$$

This agrees with the definition.

(b) For $\text{VRTINY} = 1$, we have $vp1' = p - 1 - e_{min} + e_{pr} \in [-2 : p-2]$ and $vp2' = p-1-e_{min}+e_{pr}+1 \in [-2:p-2]$, so that $vp2 = vp1+1$. The difference of the fixed rounding position $p-1$ and the rounding position $vp1'$ is given by

$$\begin{aligned} p - 1 - vp1' &= e_{min} - e_{pr} \\ &= \begin{cases} varterm + 1 & \text{if DBL} \\ varterm - 28 & \text{otherwise.} \end{cases} \end{aligned}$$

From the above range of the rounding position $vp1'$, it follows, that the value $varterm$ is in the range $varterm \in [0:53]$, so that it can be represented by $varterm = <\text{VARTERM}[5:0]>$. Because $vp2' \leq 51$, we can write starting from the definition

$$\begin{aligned} \text{INJ}_{vp1'}[-2:105] &= (0^{vp1'+3}, 1, 0^{104-vp1'}) \\ &= (0^{vp1'+3}, 1, 0^{51-vp1'}, 0^{53}) \\ &= \begin{cases} (0^{vp1'+3}, 1, 0^{p-2-vp1'}, 0^{53}) & \text{if DBL} \\ (0^{vp1'+3}, 1, 0^{p-2-vp1'+29}, 0^{53}) & \text{otherwise} \end{cases} \\ &= (0^{54-varterm}, 1, 0^{varterm}, 0^{53}) \\ &= (\text{DECO}(\text{VARTERM}[5:0])[54:0], 0^{53}) \\ &= (\text{VARMASK0}[-2:52], 0^{53}) \\ &= (\text{MASK0}_{vp1'}[-2:53], 0^{52}) \end{aligned}$$

as required by the lemma for the injection representation $\text{INJ}_{vp1'}[-2:105]$.

Because in case (b) we have $vp2' = vp1'+1$, we can write for the injection representation $\text{INJ}_{vp2'}[-2:105]$ starting from the definition

$$\begin{aligned}
\text{INJ}_{vp2'}[-2:105] &= (0^{vp2'+3}, 1, 0^{104-vp2'}) \\
&= (0, 0^{vp2'+2}, 1, 0^{104-vp2'}) \\
&= (0, 0^{vp1'+3}, 1, 0^{103-vp1'}) \\
&= (0, \text{VARMASK0}[-2:52], 0^{52}) \\
&= (0, \text{MASK0}_{vp2'}[-1:53], 0^{52}).
\end{aligned}$$

This agrees with the selection according to the lemma for this case. In this way, the proof for the rounding mode RNU is completed.

In the rounding mode RI, the injections are defined by

$$\begin{aligned}
inj_{vp1'} &= 2^{-vp1'} - 2^{-105} \\
&= <\text{INJ}_{vp1'}[-2:105]>_{neg} \\
&= <(0^{vp1'+3}, 1, 1^{104-vp1'})>_{neg} \qquad (4.291) \\
inj_{vp2'} &= 2^{-vp2'} - 2^{-105} \\
&= <\text{INJ}_{vp2'}[-2:105]>_{neg} \\
&= <(0^{vp2'+3}, 1, 1^{104-vp2'})>_{neg}. \qquad (4.292)
\end{aligned}$$

We compare the injections in the rounding mode RNU (see equations 4.289 and 4.290) and in the rounding mode RI (see equations 4.291 and 4.292). In the rounding mode RNU, the binary representation of the injection $\text{INJ}_{vp}[-2:105]$ only contains a single bit that is one, namely $\text{INJ}_{vp}[vp+1] = 1$. In the representation of an injection for the rounding mode RI, exactly the bits $\text{INJ}_{vp}[vp+1:105]$ are all ones. Thus, to get the equations for the injections in the rounding mode RI from the equations for the injections in the rounding mode RNU, only the bits $\text{INJ}_{vp}[vp+2:105]$ which are zero in the rounding mode RNU, have to be inverted for the rounding mode RI. This can easily be checked in the equations for $\text{MASK0}_{vp1'}$, $\text{MASK0}_{vp2'}$ and $\text{MASK1}_{vp1'}$, $\text{MASK1}_{vp2'}$, so that the proof of the lemma is completed. □

In the following we describe the implementation of the truncations according to step (B.I.2) and the implementation of the L-bit-fix according to step (B.II). The computation of the truncations according to equations 4.285-4.288 is based on the masks $\overline{\text{MASK1}_{vp1'}[-2:52]}$ and $\overline{\text{MASK1}_{vp2'}[-1:52]}$, that have exactly ones in the positions that are relevant in the truncated significands. For the L-bit-fix we compute the masks $\text{LPDMASK12}[-2:52]$ and $\text{LPDMASK24}[-1:52]$, that have in their L-bits $\text{LPDMASK12}[vp1']$ resp. $\text{LPDMASK24}[vp2']$ the value of the L-bit-fix condition LFIX12 resp. LFIX24, and that have zeros in all other positions.

For the computations of the masks $\text{LPDMASK12}[-2:52]$ and $\text{LPDMASK24}[-1:52]$, the masks $\text{MASK1}_{vp1'}$ and $\text{MASK0}_{vp2'}$, that were involved in the rounding injection generation for the rounding mode RI, are used to select the proper L-bit position and to truncate the injected significands after bit positions $vp1'$ resp. $vp2'$ according to equations 4.285-4.288.

To detect the L-bit fix condition for the rounding position $vp1'$ according to equation 4.289, the condition $\overline{\text{STICKY12}'[vp1']} \iff (\text{FINJ12}[vp1'+1:104] = 0^{104-vp1'})$ is required. Accordingly, for the L-bit fix at the rounding position $vp2'$ (see equation 4.289), the condition $\overline{\text{STICKY24}'[vp2']} \iff (\text{FINJ24}[vp2'+1:105] = 0^{105-vp2'})$ is required. Because these

bits are only required for the L-bit-fix in the rounding mode RNU, where SR_MODE[0] = 1 and SR_MODE[1] = 0, we can also use the sticky bits

$$\overline{\text{STICKY12}[vp1']} \iff (\text{FINJ12}[vp1'+1:104] = \text{SR\_MODE}[1]^{104-vp1'}) \quad (4.293)$$

$$\overline{\text{STICKY24}[vp2']} \iff (\text{FINJ24}[vp2'+1:105] = \text{SR\_MODE}[1]^{105-vp2'}) \quad (4.294)$$

for the computation of the L-bit-fix condition. The use of the bits $\overline{\text{STICKY12}[vp1']}$ and $\overline{\text{STICKY24}[vp2']}$ has the advantage, that these bits are also required for the detection of the inexact exception in all three reduced rounding modes.

Because the variable rounding positions $vp1'$ and $vp2'$ have to be considered in the ranges $vp1' \in [-2:52]$ and $vp2' \in [-1:52]$, the sticky-bit strings $\overline{\text{STICKY12}[-2:52]}$ and $\overline{\text{STICKY24}[-1:52]}$ are required for the computation of the L-bit fix conditions. For the computation of the inexact conditions we additionaly require $\overline{\text{STICKY12}[53]}$ and $\overline{\text{STICKY24}[53]}$.

We compute the sticky-bits $\overline{\text{STICKY12}[vp1']}$ and $\overline{\text{STICKY24}[vp2']}$ using the technique from [4] for detecting the condition $'A + B = K'$. In contrast to a straight-forward implementation of equations 4.293-4.294, that include the computation the binary representation of $\text{FINJ12}[vp1'+1:104]$ resp. $\text{FINJ24}[vp2'+1:105]$, with the technique from [4], the sticky-bits can be directly computed from the carry-save representation of $finj12$ resp. $finj24$ without requiring a carry-propagate addition. This allows to compute the sticky-bits $\overline{\text{STICKY12}[vp1']}$ and $\overline{\text{STICKY24}[vp2']}$ in parallel to the compressions of $finj12$ and $finj24$ from the carry-save representations to the binary representations. The details of these sticky-bit computations are described by the following lemma. In this lemma, we denote a carry-save representation of $finj12$ by the bit-strings FINJ12C[–2:104] and FINJ12S[–2:104], and a carry-save representation of $finj24$ by the bit-strings FINJ24C[–1:105] and FINJ24S[–1:105].

Lemma 4.31 *With the computation of*

$$\begin{aligned}
\text{P12}[-2:104] &= (\text{FINJ12C}[-2:104] \oplus \text{FINJ12S}[-2:104]) \\
\text{G12}[-2:104] &= (\text{FINJ12C}[-2:104] \wedge \text{FINJ12S}[-2:104]) \\
\text{V12}[-2:105] &= ((\text{P12}[-2:104] \wedge \overline{\text{SR_MODE}[1]}) \vee \text{G12}[-2:104], 0) \\
\text{W12}[-2:104] &= (\text{P12}[-2:104] \oplus \text{SR_MODE}[1]) \\
\text{CSSTICKY12}[-2:104] &= \text{W12}[-2:104] \overline{\oplus} \text{V12}[-1:105] \\
\text{P24}[-1:105] &= (\text{FINJ24C}[-1:105] \oplus \text{FINJ24S}[-1:105]) \\
\text{G24}[-1:105] &= (\text{FINJ24C}[-1:105] \wedge \text{FINJ24S}[-1:105]) \\
\text{V24}[-1:106] &= ((\text{P24}[-1:105] \wedge \overline{\text{SR_MODE}[1]}) \vee \text{G24}[-1:105], 0) \\
\text{W24}[-1:105] &= (\text{P24}[-1:105] \oplus \text{SR_MODE}[1]) \\
\text{CSSTICKY24}[-1:105] &= \text{W24}[-1:105] \overline{\oplus} \text{V24}[0:106]
\end{aligned}$$

we get for each $vp1' \in [-2:104]$ *and* $vp2' \in [-1:105]$

$$\begin{aligned}
\text{ANDTREE}(\text{CSSTICKY12}[vp1'+1:104]) &\iff (\text{FINJ12}[vp1'+1:104] = \text{SR_MODE}[1]^{104-vp1'}) \\
\text{ANDTREE}(\text{CSSTICKY24}[vp2'+1:105]) &\iff (\text{FINJ24}[vp2'+1:105] = \text{SR_MODE}[1]^{105-vp2'})
\end{aligned}$$

so that the sticky-bits $\overline{\text{STICKY12}[vp1']}$ *and* $\overline{\text{STICKY24}[vp2']}$ *can be computed by*

$$\begin{aligned}
\overline{\text{STICKY12}[vp1']} &= \text{ANDTREE}(\text{CSSTICKY12}[vp1'+1:104]) \\
\overline{\text{STICKY24}[vp2']} &= \text{ANDTREE}(\text{CSSTICKY24}[vp2'+1:105])
\end{aligned}$$

Proof: The proof can be found in [4] by setting $k_i = \text{SR\_MODE}[1]$ for $i \in [-2:105]$, $\text{A}[-2:104] = \text{FINJ12C}[-2:104]$ resp. $\text{A}[-1:105] = \text{FINJ12C}[-1:105]$, and $\text{B}[-2:104] = \text{FINJ12S}[-2:104]$ resp. $\text{B}[-1:105] = \text{FINJ24S}[-1:105]$. □

In this lemma only the computations for each single sticky-bit $\overline{\text{STICKY12}[vp1']}$ and $\overline{\text{STICKY24}[vp2']}$ are described. The computation of the whole sticky-bit string $\overline{\text{STICKY12}[-2:53]}$ is implemented by the use of the parallel-prefix ANDSYMB-function PPAND, that computes from an input string $\text{INPUT}[n_1:n_2]$ in its nth output $\text{PPAND}(\text{INPUT}[n_1:n_2])[n] = \text{ANDTREE}(\text{INPUT}[n:n_2])$, so that according to the previous lemma we get $\text{PPAND}(\text{CSSTICKY12}[-2:104])[vp1'+1] = \overline{\text{STICKY12}[vp1']}$ and, thus, $\text{PPAND}(\text{CSSTICKY12}[-2:104])[-1:54] = \overline{\text{STICKY12}[-2:53]}$. Accordingly, the sticky-bit string $\overline{\text{STICKY24}[-1:52]}$ is computed by $\overline{\text{STICKY24}[-1:53]} = \text{PPAND}(\text{CSSTICKY24}[-1:105])[0:54]$. This completes the description of the implementation for the sticky-bit strings $\overline{\text{STICKY12}[-2:53]}$ and $\overline{\text{STICKY24}[-1:52]}$.

Based on the sticky-bit strings $\overline{\text{STICKY12}[-2:53]}$ and $\overline{\text{STICKY24}[-1:53]}$ and the masks $\text{MASK1}_{vp1'}[-2:53]$ and $\text{MASK1}_{vp2'}[-1:53]$ from the generation of the injections in the previous lemma, the following lemma describes the computation of the truncation and the computation of the L-bit-fix.

Lemma 4.32 *(a) The truncations according to equations 4.285-4.288 can be computed by*

$$\begin{aligned}
\text{FPRND12}'[-2:52] &= \text{FINJ12}[-2:52] \;\; \textit{AND} \;\; \overline{\text{MASK1}_{vp1'}[-2:52]} \\
\text{FPRND24}'[-1:52] &= \text{FINJ24}[-1:52] \;\; \textit{AND} \;\; \overline{\text{MASK1}_{vp2'}[-1:52]}.
\end{aligned}$$

(b) With the detection of the L-bit fix conditions by the masks

$$\begin{aligned}
\text{LPDMASK12}[-2:52] &= \text{SR_MODE}[0] \;\; \textit{AND} \;\; \text{MASK1}_{vp1'}[-1:53] \;\; \textit{AND} \;\; \overline{\text{STICKY12}[-2:52]} \\
\text{LPDMASK24}[-1:52] &= \text{SR_MODE}[0] \;\; \textit{AND} \;\; \text{MASK1}_{vp2'}[0:53] \;\; \textit{AND} \;\; \overline{\text{STICKY24}[-1:52]},
\end{aligned}$$

the combination of the truncation and the L-bit-fix can be computed by

$$\begin{aligned}
\text{FPRND12}[-2:52] &= \text{FPRND12}'[-2:52] \;\; AND \;\; \overline{\text{LPDMASK12}[-2:52]} \\
&= \text{FINJ12}[-2:52] \;\; AND \;\; \overline{\text{MASK1}_{vp1'}[-2:52]} \;\; AND \;\; \overline{\text{LPDMASK12}[-2:52]} \\
\text{FPRND24}[-1:52] &= \text{FPRND24}'[-1:52] \;\; AND \;\; \overline{\text{LPDMASK24}[-1:52]} \\
&= \text{FINJ24}[-1:52] \;\; AND \;\; \overline{\text{MASK1}_{vp2'}[-1:52]} \;\; AND \;\; \overline{\text{LPDMASK24}[-1:52]}.
\end{aligned}$$

Proof: (a) It follows from equation 4.291 in the previous lemma, that:

$$\text{MASK1}_{vp1'}[-2:52] \;=\; (0^{vp1'+3}, 1^{52-vp1'}).$$

According to this equation the bit string $\text{MASK1}_{vp1'}[-2:52]$ has exactly zeros in the positions $[-2:vp1']$. Thus, starting from equations 4.285-4.286, we get

$$\begin{aligned}
fprnd12' &= <\text{FPRND12}'[-2:52]>_{neg} \\
&= rnd_{RZ,vp1'}(finj12) \\
&= <\text{FINJ12}[-2:vp1']>_{neg} \\
&= <(\text{FINJ12}[-2:52] \;\; \text{AND} \;\; \overline{\text{MASK1}_{vp1'}[-2:52]})>_{neg},
\end{aligned}$$

so that as required

$$\text{FPRND12}'[-2:52] = \text{FINJ12}'[-2:52] \;\; \text{AND} \;\; \overline{\text{MASK1}_{vp1'}[-2:52]}.$$

The equation for $\text{FPRND24}'[-1:52]$ can be shown analogously.

(b) According to equation 4.289, the L-bit fix condition for the rounding position $vp1'$ is given by:

$$\begin{aligned}
\text{LFIX12} &= \text{SR_MODE}[0] \;\; \text{AND} \;\; \overline{\text{ORTREE}(\text{FINJ12}[vp1'+1:105])} \\
&= \text{SR_MODE}[0] \;\; \text{AND} \;\; \overline{\text{STICKY12}[vp1'])}
\end{aligned}$$

Considering only the valid positions $[-2:vp1']$ of the truncated significand $\text{FPRND12}'[-2:52]$, the bit string $\text{LSEL12}[-2:52] = \text{MASK1}_{vp1'}[-1:53]$ masks the L-bit position $vp1'$ by $\text{LSEL12}[-2:vp1'-1] = 0^{vp1'+2}$ and $\text{LSEL12}[vp1'] = 1$. Because

$$\text{LPDMASK12}[-2:52] = \text{SR\_MODE}[0] \;\; \text{AND} \;\; \text{LSEL12}[-2:52] \;\; \text{AND} \;\; \text{STICKY12}[-2:52],$$

it follows from the above, that $\text{LPDMASK12}[-2:vp1'] = (0^{vp1'+2}, \text{LFIX12})$. Because the rounded significand $\text{FPRND12}'[vp1'+1:52]$ is already truncated with $\text{FPRND12}'[vp1'+1:52] = 0^{52-vp1'}$, we get

$$\begin{aligned}
\text{FPRND12}[-2:52] &= \text{FINJ12}[-2:52] \;\text{AND}\; \overline{\text{MASK1}_{vp1'}[-2:52]} \;\text{AND}\; \overline{\text{LPDMASK12}[-2:52]} \\
&= \text{FPRND12}'[-2:52] \;\; \text{AND} \;\; \text{AND} \; \overline{\text{LPDMASK12}[-2:52]} \\
&= (\text{FPRND12}[-2:vp1'-1], \text{FPRND12}[vp1'] \wedge \overline{\text{LFIX12}}, 0^{52-vp1'})
\end{aligned}$$

This equation agrees with the definition of the L-bit-fix for the computation of the rounded significand $fprnd12$ from the significand $fprnd12'$. The equations for the computation of $\text{FPRND24}[-2:52]$ can be shown analogously. □

This completes the descriptions of the equations for the implementation of step (B), thus, leaving the description of the exception detections. The detection of the invalid exception INV was already included in the special cases computations. The detections of the inexact, the underflow and the overflow exceptions INX, UNF and OVF are described in the following lemma.

Lemma 4.33 *With the definition of (note, that* TINY1 *was already used in the computation of* VRTINY *in lemma 4.30) :*

$$\begin{aligned}
\text{LARGE0} &\iff (e_{max} - e_{pr} < 0)\\
\text{LARGE1} &\iff (e_{max} - (e_{pr}+1) < 0)\\
\text{RMASK12}[-1:53] &\iff (\text{MASK1}_{vp1'}[-2], \text{MASK1}_{vp1'}[-1:52] \;\; AND \;\; \overline{\text{MASK1}_{vp1'}[-2:53]})\\
\text{RMASK24}[0:53] &\iff (\text{MASK1}_{vp2'}[-1], \text{MASK1}_{vp1'}[0:52] \;\; AND \;\; \overline{\text{MASK1}_{vp1'}[-1:53]})\\
\text{INX12} &\iff \text{ORTREE}(\text{RMASK12}[-1:53] \;\; AND \;\; \text{STICKY12}[-1:53]) \;\; OR\\
&\quad (\text{SR_MODE}[1] \vee \text{SR_MODE}[0]) \oplus \text{ORTREE}(\text{RMASK12}[-1:53] \wedge \text{FINJ12}[-1:53])\\
\text{INX24} &\iff \text{ORTREE}(\text{RMASK24}[0:53] \;\; AND \;\; \text{STICKY24}[0:53]) \;\; OR\\
&\quad (\text{SR_MODE}[1] \vee \text{SR_MODE}[0]) \oplus \text{ORTREE}(\text{RMASK24}[-1:53] \wedge \text{FINJ24}[-1:53])\\
\text{TINY0} &\iff (e_{pr} - e_{min} < 0)\\
\text{TINY1} &\iff (e_{pr} + 1 - e_{min} < 0)\\
\text{TINY2} &\iff (e_{pr} + 2 - e_{min} < 0),
\end{aligned}$$

the overflow, the inexact and the underflow exception can be detected by

$$\text{OVF} = \begin{cases} \overline{\text{SPCA}} \wedge \overline{\text{WINZIG}} \wedge \text{LARGE1} & \text{CFOVF}\mathit{1}\\ \overline{\text{SPCA}} \wedge \overline{\text{WINZIG}} \wedge \text{LARGE0} & \mathit{otherwise} \end{cases}$$

$$\text{INX} = \begin{cases} \overline{\text{SPCA}} \wedge (\text{WINZIG} \vee \text{INX24}) \vee \text{OVF}) & \mathit{if}\ \text{CFOVF1}\\ \overline{\text{SPCA}} \wedge (\text{WINZIG} \vee \text{INX12}) \vee \text{OVF}) & \mathit{otherwise} \end{cases}$$

$$\text{UNF} = \begin{cases} \overline{\text{SPCA}} \wedge (\text{INX24} \vee \text{UNF\_EN}) \wedge ((\text{TINY2} \wedge \text{CFOVF2}) \vee (\text{TINY1} \wedge \overline{\text{CFOVF2}})) & \mathit{if}\ \text{CFOVF1}\\ \overline{\text{SPCA}} \wedge (\text{INX12} \vee \text{UNF\_EN}) \wedge \text{TINY0} & \mathit{otherwise.} \end{cases}$$

Proof: In this multiplication unit the overflow condition can be written as

$$\text{OVF} \iff \overline{\text{SPCA}} \wedge \overline{\text{WINZIG}} \wedge (|val(s_{pr}, e_{prnd}, fprnd)| \geq 2^{e_{max}+1}.)$$

An overflow could only occur for results with very large exponents where $e_{pr}+2 \geq e_{prnd} >> e_{min}$. Because of corollary 2.10 we then also have $e_{pr} + wec > e_{min}$. Thus, it follows from lemma 4.29, that $\text{OVF} \Longrightarrow \overline{\text{CFOVF2}}$ and we do not have to consider the case $\text{CFOVF2} = 1$ in the overflow detection. In this way we get according to equation 4.278, where we only have to consider $fprnd12 < 2$ and $fprnd24 < 2$:

$$\begin{aligned}
(|val(s_{pr}, e_{prnd}, fprnd)| \geq 2^{e_{max}+1}) &\iff \begin{cases} ((e_{pr}+1 > e_{max}) & \text{CFOVF1}\\ (e_{pr} > e_{max}) & \text{otherwise} \end{cases}\\
&\iff \begin{cases} \text{LARGE1} & \text{CFOVF1}\\ \text{LARGE0} & \text{otherwise.} \end{cases}
\end{aligned}$$

In this way we get the equation for OVF from the lemma.

Because all special cases results with ($\text{SPCA} = 1$) are exact, the condition for an inexact exception (see section 2.4) can be written as

$$\text{INX} \iff (\overline{\text{SPCA}} \wedge \text{RNDINX}) \vee \text{OVF}) \tag{4.295}$$

where the bit RNDINX signals the significand rounding inexactness, namely the case, that significand rounding changes the value of the significand product.

According to the use of RNDINX in equation 4.295, we can assume for the computation of RNDINX that $\text{SPCA} = 0$ and that no overflow occurs. Thus, according to equation

4.279 and equation 4.278 we have to consider the following three cases for the detection of RNDINX: (a) (WINZIG = 1); (b) ($\overline{\text{WINZIG}} \wedge \overline{\text{CFOVF1}} = 1$); and (c) ($\overline{\text{WINZIG}} \wedge \text{CFOVF1} = 1$).

Because only non-zero significand products have to be considered for (WINZIG = 1), it is obvious that we have RNDINX = 1 in case (a). For the rounding inexactness conditions in the cases (b) and (c) we use equation 2.51 regarding the $(vp1'+1)$-representative of fpr, that relates to the computation of $fprnd12$ (case (b)), and the $(vp2'+1)$-representative of $fpr/2$, that relates to the computation of $fprnd24$ (case(c)). In this way we get

$$\text{RNDINX} = \begin{cases} \text{WINZIG} \vee \text{ORTREE}(\text{FPR}[vp2':104]) & \text{if CFOVF1} \\ \text{WINZIG} \vee \text{ORTREE}(\text{FPR}[vp1'+1:104]) & \text{otherwise.} \end{cases} \tag{4.296}$$

Because we do not compute a representation of the exact significand product fpr, the above equation has to be computed from the injected significand products $finj12$ and $finj24$. By considering the injections that are included in the injected significands in the rounding modes RNU and RI, the above ORTREE-conditions can be computed based on the representations of the injected significands by

$$\begin{aligned} \text{ORTREE}(\text{FPR}[vp1'+1:104]) &= \begin{cases} \text{ORTREE}(\overline{\text{FINJ12}[vp1'+1]}, \overline{\text{FINJ12}[vp1'+2:104]}) & \text{if SR\_MODE}[1] \\ \text{ORTREE}(\overline{\text{FINJ12}[vp1'+1]}, \text{FINJ12}[vp1'+2:104]) & \text{if SR\_MODE}[0] \\ \text{ORTREE}(\text{FINJ12}[vp1'+1:104]) & \text{otherwise.} \end{cases} \\ &= (\text{SR\_MODE}[1] \vee \text{SR\_MODE}[0]) \oplus (\text{FINJ12}[vp1'+1]) \\ &\quad \vee \text{ORTREE}(\text{SR\_MODE}[1] \oplus \text{FINJ12}[vp1'+2:104]) \\ &= (\text{SR\_MODE}[1] \vee \text{SR\_MODE}[0]) \oplus (\text{FINJ12}[vp1'+1]) \\ &\quad \vee \text{STICKY12}[vp1'+1]) \end{aligned} \tag{4.297}$$

$$\begin{aligned} \text{ORTREE}(\text{FPR}[vp2':104]) &= \begin{cases} \text{ORTREE}(\overline{\text{FINJ24}[vp2'+1]}, \overline{\text{FINJ24}[vp2'+2:105]}) & \text{if SR\_MODE}[1] \\ \text{ORTREE}(\overline{\text{FINJ24}[vp2'+1]}, \text{FINJ24}[vp2'+2:105]) & \text{if SR\_MODE}[0] \\ \text{ORTREE}(\text{FINJ24}[vp2'+1:05]) & \text{otherwise.} \end{cases} \\ &= (\text{SR\_MODE}[1] \vee \text{SR\_MODE}[0]) \oplus (\text{FINJ24}[vp2'+1]) \\ &\quad \vee \text{ORTREE}(\text{SR\_MODE}[1] \oplus \text{FINJ24}[vp2'+2:105]) \\ &= (\text{SR\_MODE}[1] \vee \text{SR\_MODE}[0]) \oplus (\text{FINJ24}[vp2'+1]) \\ &\quad \vee \text{STICKY24}[vp2'+1] \end{aligned} \tag{4.298}$$

For the computations in equation 4.297, we have to select the bit $\text{FINJ12}[vp1'+1]$ from the bit string $\text{FINJ12}[-2:52]$ and to select the bit $\text{STICKY12}[vp1'+1]$ from the bit string $\text{STICKY12}[-2:52]$. For this purpose we require a mask, that exactly has a one in position $vp1'+1$ and zeros in all other positions. This is exactly the case for

$$\begin{aligned} \text{RMASK12}[-1:53] &= (\text{MASK1}_{vp1'}[-2], \text{MASK1}_{vp1'}[-1:52] \ \text{AND} \ \overline{\text{MASK1}_{vp1'}[-2:51]}) \\ &= (0^{vp1'+2}, 1^{53-vp1'}) \ \text{AND} \ (1^{vp1'+3}, 0^{52-vp1'}) \\ &= (0^{vp1'+2}, 1, 0^{52-vp1'}). \end{aligned}$$

Thus, we get

$$\begin{aligned} \text{FINJ12}[vp1'+1] &= \text{ORTREE}(\text{RMASK12}[-1:53] \ \text{AND} \ \text{FINJ12}[-2:52]) \\ \text{STICKY12}[vp1'+1] &= \text{ORTREE}(\text{RMASK12}[-1:53] \ \text{AND} \ \text{STICKY12}[-2:52]) \end{aligned}$$

and equation 4.297 can be written as:

$$\begin{aligned}\text{ORTREE}(\text{FPR}[vp1'+1:104]) &= \text{ORTREE}(\text{RMASK12}[-1:53] \text{ AND } \text{STICKY12}[-2:52]) \text{ OR} \\ &\quad (\text{SR\_MODE}[1] \vee \text{SR\_MODE}[0]) \oplus \text{ORTREE}(\text{RMASK12}[-1:53] \wedge \text{FINJ12}[-2:52]) \\ &= \text{INX12}\end{aligned}$$

It can be shown analogously, that equation 4.298 can be computed by

$$\begin{aligned}\text{ORTREE}(\text{FPR}[vp2':104]) &= \text{ORTREE}(\text{RMASK24}[0:53] \text{ AND } \text{STICKY24}[-1:52]) \text{ OR} \\ &\quad (\text{SR\_MODE}[1] \vee \text{SR\_MODE}[0]) \oplus \text{ORTREE}(\text{RMASK24}[0:53] \wedge \text{FINJ24}[-1:52]) \\ &= \text{INX24}\end{aligned}$$

The substitution of RNDINX in equation 4.295 according to equation 4.296 and the substitution of ORTREE(FPR$[vp1'+1:104]$) and ORTREE(FPR$[vp2':104]$) by INX12 resp. INX24 according to the previous two equations then yield the equation for INX from the lemma.

For the multiplication unit the condition for an underflow exception is defined by (the function $TINY$ is defined in definition 2.10):

$$\begin{aligned}\text{UNF} &= \begin{cases} \overline{\text{SPCA}} \wedge TINY(s_{pr}, e_{prnd}, fprnd) & \text{if UNF\_EN} \\ \overline{\text{SPCA}} \wedge \text{RNDINX} \wedge TINY(s_{pr}, e_{prnd}, fprnd) & \text{otherwise.} \end{cases} \\ &= \overline{\text{SPCA}} \wedge (\text{RNDINX} \vee \text{UNF\_EN}) \wedge TINY(s_{pr}, e_{prnd}, fprnd) \\ &= \begin{cases} \overline{\text{SPCA}} \wedge (\text{INX24} \vee \text{UNF\_EN}) \wedge TINY(s_{pr}, e_{pr}+1, fprnd24) & \text{if CFOVF1} \\ \overline{\text{SPCA}} \wedge (\text{INX12} \vee \text{UNF\_EN}) \wedge TINY(s_{pr}, e_{pr}, fprnd12) & \text{otherwise} \end{cases}\end{aligned}$$

Because for CFOVF1 $= 0$ the rounded significand $fprnd12$ is smaller than 2, we get

$$\begin{aligned}TINY(s_{pr}, e_{pr}, fprnd12) &\iff (e_{pr} < e_{min}) \\ &\iff \text{TINY0}\end{aligned}$$

Because $fprnd24 < 2$ for CFOVF2 $= 0$ and $fprnd24 = 2$ for CFOVF2 $= 0$, we get for the function

$$\begin{aligned}TINY(s_{pr}, e_{pr}+1, fprnd24) &\iff \begin{cases} (e_{pr}+2 < e_{min}) & \text{if CFOVF2} \\ (e_{pr}+1 < e_{min}) & \text{otherwise} \end{cases} \\ &\iff ((\text{TINY2} \wedge \text{CFOVF2}) \vee (\text{TINY1} \wedge \overline{\text{CFOVF2}}))\end{aligned}$$

With the substitution of $TINY(s_{pr}, e_{pr}, fprnd12)$ and $TINY(s_{pr}, e_{pr}+1, fprnd24)$ in equation 4.299 according to the previous equations, we get the equation for UNF from the lemma. □

This lemma completes the description of the computations for the exception flags, so that the description of the whole multiplication unit III is completed.

Figure 4.27 depicts the main structure of the multiplication unit III, figure 4.29 depicts a detailed block diagram of the implementation of step (B) and a detailed block diagram of the exceptions and exponent computations is given in figure 4.30.

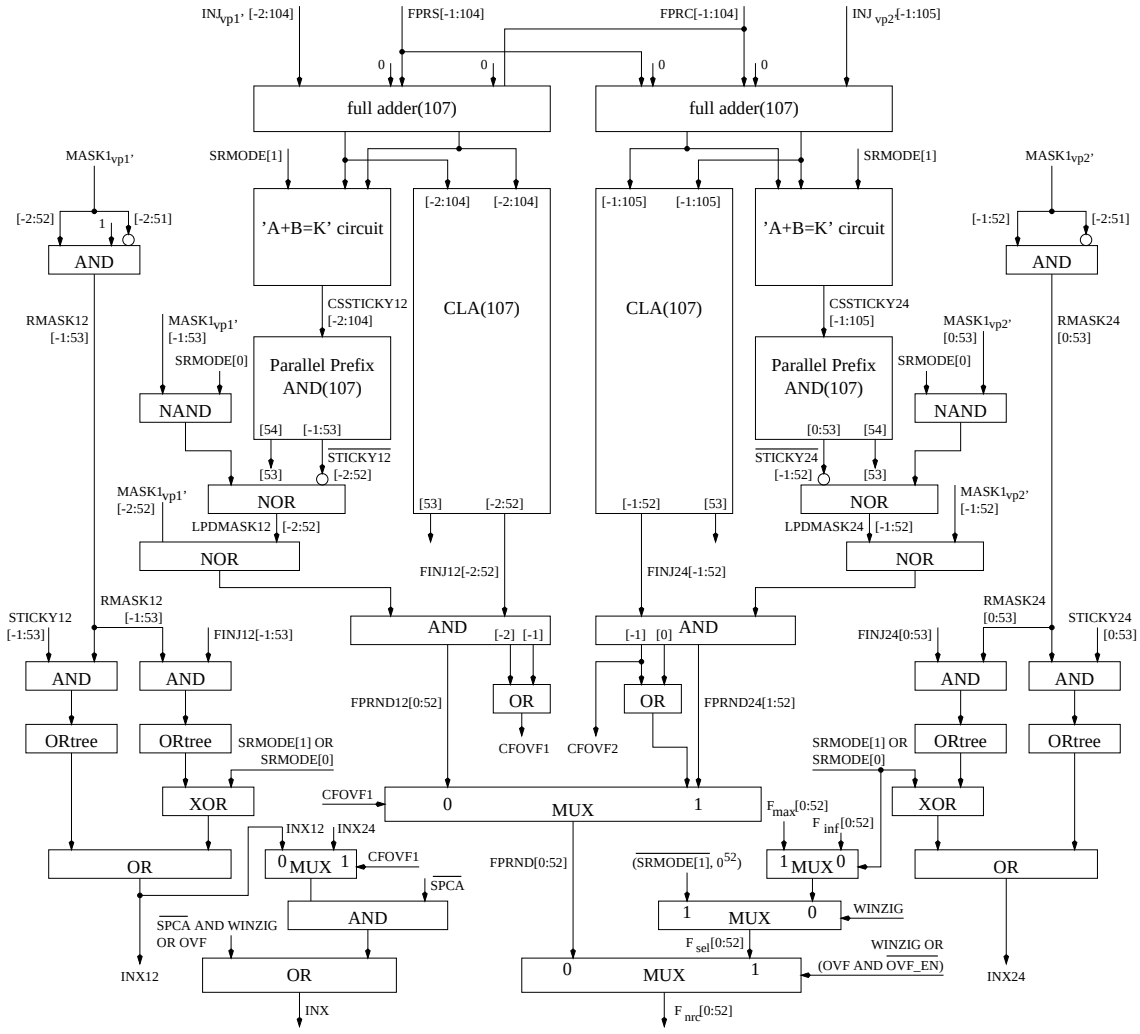

Figure 4.29: Implmentation of the significand compression and variable position rounding in the multiplication unit III.

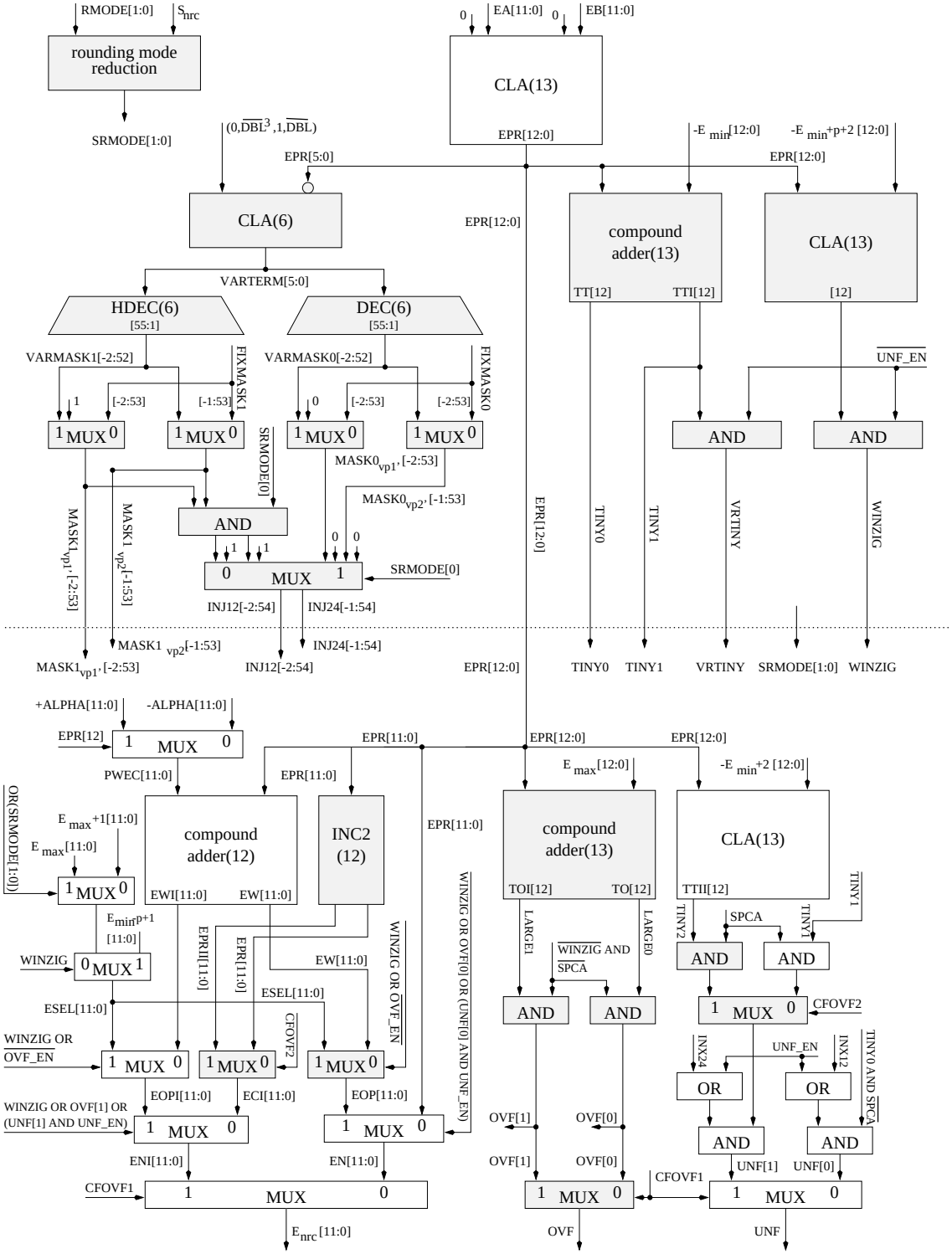

Figure 4.30: Implementation of the exponent&exceptions circuit of the multiplication unit III.

4.4 Division

In this section the implementations of the floating-point division are described. Like in the previous sections for the addition and the multiplication implementations, also for the division, the descriptions are separated into three subsections for the microarchitectures I, II and III. For the division implementations the main details have to be described about the computation of the significand quotient. Because a very similar significand quotient implementation is used for all three microarchitectures, we will only describe it once for the implementation for microarchitecture I. For the other two microarchitectures we will only describe the small adjustments, that are required. The implementation of the significand quotient uses an initial approximation for the reciprocal of the divisor. Because the description of our implementation of this initial reciprocal approximation (see also [36, 39]) is quite complex, we describe it separately in the next subsection in preparation for the division implementations.

4.4.1 Initial Reciprocal Approximation

The circuit for the reciprocal approximation should approximate the reciprocal of a normalized input significand $y = <\text{Y}[0:52]>_{neg} \in [1,2[$. We denote the approximated reciprocal by $arecip(y) \approx 1/y$ and define the approximation error by $err(y) = 1/y - arezip(y)$. For the approximated reciprocal result $arezip(y)$ the computation has to guarantuee an upper bound on the absolute approximation error $|err(y)|$. In particular, for the implementations of the FP division, we will require initial reciprocal approximations with absolute approximation errors that are bounded by $|err(y)| < 2^{-8}$, $|err(y)| < 2^{-15}$ and $|err(y)| < 2^{-28}$ respectively.

In literature the initial reciprocal approximations fall into two groups: The constant approximation [13, 18, 35, 6, 5, 15, 42] is easy to implement in 1 clock cycle by a simple lookup table, but due to the huge cost it is limited to small accuracies ($\geq 2^{-16}$). The linear [18, 35] and modified linear [18] approximation approaches can achieve even twice the accuracy of constant approximations at nearly the same cost, but the implementations corresponding to [7, 18, 35] require about 3 clock cycles for an approximation: one lookup and decode cycle, one cycle for the adder tree of the full-size multiplication and one clock cycle for the carry-propagate addition of this multiplication.

We present a faster linear approximation implementation for the reciprocal. A description of this implementation can also be found in [36, 39]. In comparison to the previous linear reciprocal approximation implementations from literature, our implementation is accelerated by the use of the following new ideas:

1. a linear approximation formula, that reduces the widths of table lookup inputs and multiplication operands for a given approximation accuracy.

2. the use of a specific small Booth multiplier (with less than 8 partial products in the implementation for $|err(y)| < 2^{-28}$) for the computation of the linear approximation formula.

3. a fast redundant compression from carry-save representations to redundant Booth-Digit representations, a redundant format, that can directly be fed into the large Booth multiplier of the FP multiplication unit. This fast partial compression avoids the slow carry propagate addition step in the multiplication of the linear approximation formula.

For the description of the implementations, we first develop the linear approximation formula for the approximation of the reciprocal. We then introduce the new intermediate format, the redundant Booth-Digit (redBD) representation, in that the reciprocal approximation should be output. Based on the approximation formula, we finally describe the implementation of the computations from the binary representation of the input $\mathrm{Y}[0:52]$ to the redundant Booth-Digit representation of the approximated reciprocal $arecip(y)$ for a given appoximation accuracy. In particular we consider the implementations for the three target accuracies that will be required for the implementations of the floating-point divisions.

4.4.1.1 Approximation formula

We consider a linear approximation formula for the reciprocal. The linear and the constant parameter of this linear approximation are not fixed for the whole range of y, but the range of y is partitioned into 2^m subintervalls and for each of these subintervalls a specific a linear approximation $arezip_p(y)$ with an appropriate linear and an appropriate constant parameter is used to approximate the reciprocal function.

We consider the 2^m equidistant subintervalls $[p, p+2^{-m}[$ with $p \in \{2^m, \cdots, 2^{m+1}-1\}/2^m$. Because $y \in [1, 2-2^{-52}]$, one of these intervals contains $y \in [p, p+2^{-m}[$. We get the left endpoint of this interval by $p = <\mathrm{Y}[0:m]>_{neg}$ and we get the right endpoint by $<\mathrm{Y}[0:m]>_{neg} +2^{-m}$. The linear approximation formula for the interval $[p, p+2^{-m}[$ can be written as

$$arezip_p(y) = C0_p + C1_p \cdot (y - p)$$

with the constant parameter $C0_p$ and the linear parameter $C0_p$. For the approximation formulae in the 2^m different intervals, we require 2^m different constants $C0_p$ and 2^m different constants $C1_p$. In the implementation we will get these constants by a table lookup from a ROM for $C0_p$ and from a ROM for $C1_p$. Because $y \in [1, 2[$ is normalized and we always have $\mathrm{Y}[0] = 1$, the ROMs with the 2^m entries for $C0_p$ and $C1_p$ can be addressed by $\mathrm{Y}[1:m]$, where m is the input width of the table lookup ROMs.

In this way the delay for the implementation of the linear approximation formula can be mainly influenced by the following parameters:

- the input width m of the lookup tables, because it determines the delay of the ROM tables.
- the widths of the multiplication operands within the linear approximation formula, that influence the delay and the cost of the additional small multiplier.

We consider the linear approximation formula with the focus to minimize these parameters for a given accuracy in the follwing lemma.

Lemma 4.34 *For $y \in [p, p+2^{-m}[$ with $p \in \{2^m, \cdots, 2^{m+1}-1\}/2^m$, the reciprocal approximation of $f(y) = 1/y$ by the linear function*

$$arecip_p(y) = rnd_{RZ,wr}(C0_p - C1_p \cdot rnd_{RZ,wy}(y - p))$$

with

$$\begin{aligned} C1_p &= rnd_{RNE,wc1}\left(\frac{1}{(p+2^{-m-1})^2}\right) \\ C0_p &= rnd_{RNE,wc0}\left(\frac{1}{p+2^{-m-1}} + 2^{-2m-3} + 2^{-m-1} \cdot C1_p\right) \end{aligned}$$

results in the approximation error

$$|err(y)| = |1/y - arecip_p(y)| < 2^{-2m-3} + 2^{-wc1-m-2} + 2^{-wc0-1} + 2^{-wy} + 2^{-wr}.$$

Proof: Taylor approximation of degree 1 of the function $f(y) = 1/y$ developed at the midpoint $p + 2^{-m-1}$ of the interval $[p, p + 2^{-m}[$ yields the linear approximation formula

$$\begin{aligned} r_{taylor}(y) &= f(p + 2^{-m-1}) + f'(p + 2^{-m-1}) \cdot \left(y - (p + 2^{-m-1})\right) \\ &= \frac{1}{p + 2^{-m-1}} - \frac{1}{(p + 2^{-m-1})^2} \cdot \left(y - (p + 2^{-m-1})\right). \end{aligned}$$

Using the Lagrange error formula, the approximation error $error_{taylor} = 1/y - r_{taylor}(y)$ in the interval $[p, p + 2^{-m}[$ is bounded by

$$|error_{taylor}| \leq \left| \frac{f''(p + 2^{-m-1})}{2} \cdot \xi^2 \right| \text{ with } \xi \in [-2^{-m-1}, 2^{-m-1}).$$

As $f''(y) = \frac{2}{y^3}$ and $y \in [1, 2)$ we have

$$|error_{taylor}| \leq 2^{-2m-2}.$$

The 2nd derivative of $1/y$ is positive for $y \in [1, 2)$ and $r_{taylor}(y)$ describes a tangent of the graph of $1/y$. Therefore, $error_{taylor}$ can not become negative. By adding half of the maximum error in the approximation formula

$$r_1(y) = r_{taylor}(y) + 2^{-2m-3}$$

we halve the absolute error

$$|error_1| = |1/y - r_1(y)| \leq 2^{-2m-3}.$$

In $|error_1|$ only the approximation error, produced by the linear approximation using infinite precision numbers, is considered. We have to consider the additional influence of the discretization errors by using finite precision numbers.

First, we discretize the derivative $C1_p = rnd_{RNE,wc1}\left(\frac{1}{(p+2^{-m-1})^2}\right)$ at position $wc1$ and bring then the linear term in $r_1(y)$ to the form of the linear term in $arezip_p(y)$:

$$r_2(y) = \frac{1}{p + 2^{-m-1}} + 2^{-2m-3} + 2^{-m-1} \cdot C1_p - C1_p \cdot (y - p).$$

Because $|y - (p + 2^{-m-1})| \leq 2^{-m-1}$, and because the rounding function $rnd_{RNE,wc1}$ produces a discretization error smaller than or equal to 2^{-wc1-1}, we get the error bound

$$|error_2| = |1/y - r_2(y)| \leq 2^{-2m-3} + 2^{-wc1-m-2}.$$

Discretizing the constant part at position $wc0$:

$$C0_p = rnd_{RNE,wc0}\left(\frac{1}{p + 2^{-m-1}} - 2^{-2m-3} + 2^{-m-1} \cdot C1_p\right)$$

and the linear factor $(y - p)$ at position wy by $rnd_{RZ,wy}(y - p)$ yields

$$r_3(y) = C0_p + C1_p \cdot rndRZ, wy(y - p).$$

Because the rounding function $rnd_{RZ,wy}$ produces a discretization error smaller than 2^{-wy}, the error bound increases to

$$|error_3| = |1/y - r_3(y)| < 2^{-2m-3} + 2^{-wc1-m-2} + 2^{-wc0-1} + 2^{-wy}.$$

The linear approximation formula $arecip_p(y) = rnd_{RZ,wr}(r_3(y))$ contains then the final approximation error, that is bounded by

$$|err(y)| = |1/y - arecip_p(y)| < 2^{-2m-3} + 2^{-wc1-m-2} + 2^{-wc0-1} + 2^{-wy} + 2^{-wr}.$$

□

Corollary 4.35 *For the implementation of the reciprocal approximation we will use the linear approximation formula from lemma 4.34 with* $wc1 = m+2, wy = 2m+6$, $wc0 = 2m+5$ *and* $wr = 2m+5$, *so that we get:*

$$\begin{aligned} arecip_p(y) &= rnd_{RZ,2m+5}(C0_p - C1_p \cdot rnd_{RZ,2m+6}(y-p)) \\ C1_p &= rnd_{RNE,m+2}\left(\frac{1}{(p+2^{-m-1})^2}\right) \\ C0_p &= rnd_{RNE,2m+5}\left(\frac{1}{p+2^{-m-1}} + 2^{-2m-3} + 2^{-m-1} \cdot C1_p\right) \end{aligned}$$

This approximation formula results in an absolute error $|err(y)| < 2^{-2m-2}$.

Because $rnd_{RZ,wy}(y-p) < 2^{-m}$ and $|C1_p| < 1$, the binary representation of $rnd_{RZ,wy}(y-p)$ contains $wy-m = m+6$ non-zero positions and the binary representation of $-C1_p$ contains $wc1 = m+2$ non-zero positions. For $0.5 \leq C0_p < 1$, the most significant bit of $C0_p$ in the position with weight 2^{-1} is always a 1. Therefore, only $wc0 - 1 = 2m+4$ bits have to be saved in a lookup table entry for $C0_p$. In this way a straightforward implementation of the linear approximation formula according to corollary 4.35 requires a m-bit-in lookup table for $C0_p$ with a bit width of $2m+4$ and a m-bit-in lookup table for $-C1_p$ with a bit width of $m+2$. A $(m+2)$-bit by $(m+6)$-bit multiplication is required to compute the multiplication of this linear approximation formula.

For the three target accuracies of the reciprocal approximation, that we will require for the implementations of the floating-point division, we consider the linear reciprocal approximation formula from corollary 4.35 with $m = 13$ to get $|err(y)| < 2^{-28}$, with $m = 7$ to get $|err(y)| < 2^{-16}$ and with $m = 3$ to get $|err(y)| < 2^{-8}$.

For these three cases we define the linear approximation equations (note, that in these equations $p = <\text{Y}[0:m]>_{neg}$ and that $y - p = <\text{Y}[m+1:52]>_{neg}$)

$$arecip28(y) = rnd_{RZ,31}(C0\_28_p - C1\_28_p \cdot rnd_{RZ,32}(y-p)) \quad (4.299)$$

$$arecip16(y) = rnd_{RZ,19}(C0\_16_p - C1\_16_p \cdot rnd_{RZ,20}(y-p)) \quad (4.300)$$

$$arecip08(y) = rnd_{RZ,11}(C0\_08_p - C1\_08_p \cdot rnd_{RZ,12}(y-p)), \quad (4.301)$$

where the constants are defined by:

$$\begin{aligned} C0\_28_p &= rnd_{RNE,31}\left(\tfrac{1}{p+2^{-14}} + 2^{-29} + 2^{-14} \cdot C1\_28_p\right) & C1\_28_p &= rnd_{RNE,15}\left(\tfrac{1}{(p+2^{-14})^2}\right) \\ C0\_16_p &= rnd_{RNE,19}\left(\tfrac{1}{p+2^{-8}} + 2^{-17} + 2^{-8} \cdot C1\_16_p\right) & C1\_16_p &= rnd_{RNE,9}\left(\tfrac{1}{(p+2^{-8})^2}\right) \\ C0\_08_p &= rnd_{RNE,11}\left(\tfrac{1}{p+2^{-4}} + 2^{-9} + 2^{-4} \cdot C1\_08_p\right) & C1\_08_p &= rnd_{RNE,5}\left(\tfrac{1}{(p+2^{-4})^2}\right). \end{aligned}$$

function	m	$\|err(y)\| <$	non-zero bit positions in the representations of CO_p	$-C1_p$	$rnd_{RZ,wy}(y-p)$	$arecip$
$arecip28$	13	2^{-28}	$\mathrm{C0}_p[1:31]$	$\mathrm{C1}_p[1:15]$	$\mathrm{Y}[14:32]$	$\mathrm{ARECIP28}[0:32]$
$arecip16$	7	2^{-16}	$\mathrm{C0}_p[1:19]$	$\mathrm{C1}_p[1:9]$	$\mathrm{Y}[8:20]$	$\mathrm{ARECIP16}[0:20]$
$arecip08$	3	2^{-8}	$\mathrm{C0}_p[1:11]$	$\mathrm{C1}_p[1:5]$	$\mathrm{Y}[4:12]$	$\mathrm{ARECIP08}[0:12]$

Table 4.2: Bit positions of the operands in the linear reciprocal approximation formulae for the functions $arecip28$, $arecip16$ and $arecip08$.

The computation of $arecip28(y)$ requires a 15-bit by 19-bit multiplication and an addition with a 31-bit value, the computation of $arecip16(y)$ requires a 9-bit by 15-bit multiplication and an addition with a 19-bit value, and the computation of $arecip08(y)$ requires a 5-bit by 9-bit multiplication and an addition with a 11-bit value. The required bit positions for these computations are listed in table 4.2. We postpone a detailed description of the implementation and introduce the intermediate format, in that the reciprocal approximation should be represented in the following section.

4.4.1.2 Redundant Booth-Digit Representations

For the definition of redundant Booth-digit representations we shortly review Booth recoding. Following the descriptions of [3, 30] a number $b = <\mathrm{B}[m-1:0]>$ is recoded in Booth-2 recoding as suggested in Fig. 4.31. With $\mathrm{B}[m+1] = \mathrm{B}[m] = \mathrm{B}[-1] = 0$ and $m' = \lceil (m+1)/2 \rceil$ one writes

$$\begin{aligned} b &= <\mathrm{B}[m-1:0]> && (4.302)\\ &= 2b - b \;=\; 2<\mathrm{B}[m-1:0]> - <\mathrm{B}[m-1:0]> && (4.303)\\ &= \sum_{j=0}^{m'-1} B_{2j} \cdot 4^j && (4.304) \end{aligned}$$

where

$$\begin{aligned} B_{2j} &= 2\mathrm{B}[2j] + \mathrm{B}[2j-1] - 2\mathrm{B}[2j+1] - \mathrm{B}[2j] && (4.305)\\ &= -2\mathrm{B}[2j+1] + \mathrm{B}[2j] + \mathrm{B}[2j-1]. && (4.306) \end{aligned}$$

For $0 \le j \le m'-1$ this equation computes the *Booth-digits* $B_{2j} \in \{-2,-1,0,1,2\}$ for the number $b = <\mathrm{B}[m-1:0]>$. The Booth-digits B_{2j} of a number are not unique and not only the set of values according to equation 4.306, but each set of values $B_{2j} \in \{-2,-1,0,1,2\}$ that fulfills equation 4.304, is defining a set of Booth-digits for the number b. A string of Booth digits $(B_{2j})_{0 \le j \le m'-1}$, that fulfills equation 4.304, is called a *Booth digit representation* of the number b.

According to equation 4.306 each Booth digit B_{2j} can be computed from three consecutive bits $(\mathrm{B}[2j+1], \mathrm{B}[2j], \mathrm{B}[2j-1])$ of the binary representation of the number $b = <\mathrm{B}[m-1:0]>$. For an arbitrary string of tripels $(\mathrm{RB3}_j, \mathrm{RB2}_j, \mathrm{RB1}_j)_{0 \le j \le m'-1}$, we define the corresponding Booth digits by

$$B_{2j} = -2\mathrm{RB3}_j + \mathrm{RB2}_j + \mathrm{RB1}_j. \qquad (4.307)$$

The Booth digits B_{2j} that are computed according to this equation represent the number $b = \sum_{j=0}^{m'-1} B_{2j} \cdot 4^j = \sum_{j=0}^{m'-1} (-2\mathrm{RB3}_j + \mathrm{RB2}_j + \mathrm{RB1}_j) \cdot 4^j$. In this way the number b is

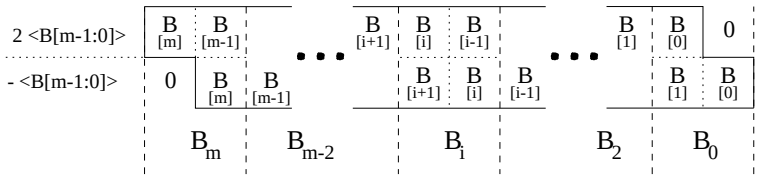

Figure 4.31: Booth digits B_{2j}

also represented by the string of tripels $(\text{RB3}_j, \text{RB2}_j, \text{RB1}_j)_{0 \le j \le m'-1}$. We define, that the string of tripels $(\text{RB3}_j, \text{RB2}_j, \text{RB1}_j)_{0 \le j \le m'-1}$ is called a *redundant Booth-digit representation* of b, iff $b = \sum_{j=0}^{m'-1} (-2\text{RB3}_j + \text{RB2}_j + \text{RB1}_j) \cdot 4^j$.

Note, that because also the string of tripels $(\text{B}[2j+1], \text{B}[2j], \text{B}[2j-1])_{0 \le j \le m'-1}$ is a redundant Booth-digit representation of b according to equations 4.304 and 4.306, a binary representation of a number can be easily converted into a redundant Booth-digit representation of the number. We denote this conversion from the binary representation of the number b to a redundant Booth-digit representation of the number b by the operation $redBD$ with

$$(\text{B}[2j+1], \text{B}[2j], \text{B}[2j-1])_{0 \le j \le m'-1} = redBD(\text{B}[m-1:0]).$$

Multiplier with Input of Redundant Booth-Digit Representation. In an ordinary implementation of a multiplier that uses Booth recoding, the binary input of one of the operands is encoded by Booth encoders (we call this the second operand and denote it by $b = <\text{B}[m-1:0]>$). Each of these Booth encoders computes a sign-magnitude representation of one Booth digit and gets as input the three consecutive bits $(\text{B}[2j+1], \text{B}[2j], \text{B}[2j-1])$ from that a Booth digit is originally computed according to equation 4.306. We change the specification of the multiplier, so that as the second operand not the binary representation of the number b, but the set of tripels $(\text{B}[2j+1], \text{B}[2j], \text{B}[2j-1])$ has to be input. (This change can easily be realized, because these are exactly the inputs of the Booth encoders). Thus, to multiply a number a by b, not the binary representation of b, but the redundant Booth digit representation $(\text{B}[2j+1], \text{B}[2j], \text{B}[2j-1])_{0 \le j \le m'-1}$ is required as input of the new multiplier. Because in equations 4.306 and 4.307 the weights of the bits in $(\text{B}[2j+1], \text{B}[2j], \text{B}[2j-1])$ and $(\text{RB3}_j, \text{RB2}_j, \text{RB1}_j)$ correspond to each other, it does not change the value of the second operand if the redundant Booth-digit representation $(\text{B}[2j+1], \text{B}[2j], \text{B}[2j-1])_{0 \le j \le m'-1}$ is replaced by an arbitrary Booth-digit representation $(\text{RB3}_j, \text{RB2}_j, \text{RB1}_j)_{0 \le j \le m'-1}$ of the number b. In this way we get a multiplier, that multiplies a number $a = <\text{A}[k-1:0]>$, that is given in the binary representation $\text{A}[k-1:0]$, with a number b that is given by a an arbitrary redundant Booth-digit representation.

Compression from Carry-Save to Redundant Booth-Digit representations. The following lemma describes how a number can be converted from carry-save to redundant Booth-digit representation. This technique will help us to avoid the carry-propagate addition in the implementation of the multiplication for the linear reciprocal approximation.

Lemma 4.36 *Let the compression injection compinj be defined by* $compinj = \sum_{j=0}^{m'-1} 2 \cdot 4^j$. *Then, from a carry-save representation of* $b + compinj$, *a redundant Booth-digit rep-*

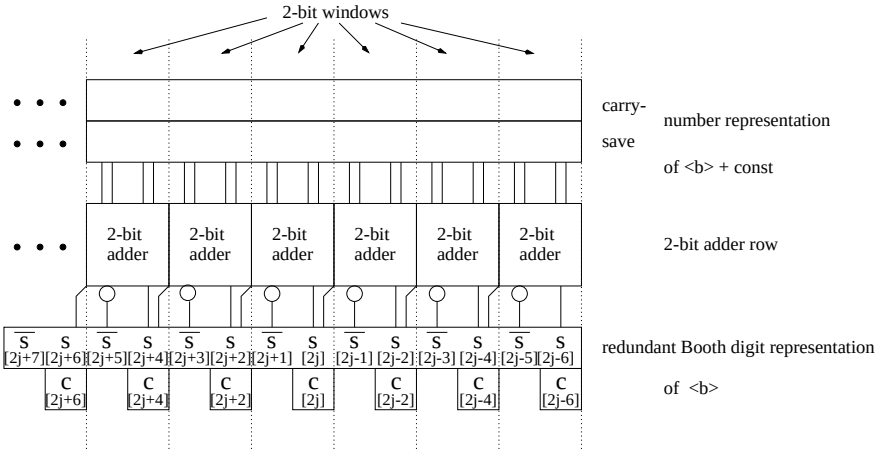

Figure 4.32: Compression from carry-save to redundant Booth-digit representation.

resentation of b can be computed by a line of two-bit adders with inverted most significand sum-bit outputs like depicted in figure 4.32.

Proof: We get a carry-save representation of $b + compinj$. Looking at bit windows of width 2 in this carry-save representation of the number $b + compinj$, each window j contains 4 bits (see Fig. 4.32); two with the weight of one and two with the weight of two. Therefore, the binary value w_j of the part of the number within a window j is in the range $w_j \in \{0, \cdots, 6\}$. The number $b + compinj$ can then be written as:

$$b + compinj = \sum_{j=0}^{m'-1} w_j 4^j.$$

If we input the 4 bits of a window j into a 2-bit adder, we get three output bits $\mathrm{c}[2j+2]$, $\mathrm{s}[2j+1]$ and $\mathrm{s}[2j]$, that represent the value of the window by:

$$w_j = 4 \cdot \mathrm{c}[2j+2] + 2 \cdot \mathrm{s}[2j+1] + \mathrm{s}[2j].$$

With $\mathrm{s}[2m'+1] = \mathrm{s}[2m'] = [-2] = 0$, we have

$$\begin{aligned} b + compinj &= \sum_{j=0}^{m'-1} (4 \cdot \mathrm{c}[2j+2] + 2 \cdot \mathrm{s}[2j+1] + \mathrm{s}[2j]) 4^j \\ &= \sum_{j=0}^{m'} (2 \cdot \mathrm{s}[2j+1] + \mathrm{s}[2j] + \mathrm{c}[2j]) 4^j. \end{aligned}$$

Now we subtract the additive constant $compinj$ on both sides, and get the value of b:

$$\begin{aligned} b + compinj - \sum_{j=0}^{m'-1} 2 \cdot 4^j &= \sum_{j=0}^{m'} (2 \cdot \mathrm{s}[2j+1] + \mathrm{s}[2j] + \mathrm{c}[2j] - 2) 4^j \\ b &= \sum_{j=0}^{m'} (2 \cdot (\mathrm{s}[2j+1] - 1) + \mathrm{s}[2j] + \mathrm{c}[2j]) 4^j \end{aligned}$$

As $\mathrm{x} - 1 \equiv -\overline{\mathrm{x}}$ for $\mathrm{x} \in \{0, 1\}$, we can substitute $\mathrm{s}[2j+1] - 1$ by $-\overline{\mathrm{s}[2j+1]}$ and have:

$$b = \sum_{j=0}^{m'} (-2\overline{\mathrm{s}[2j+1]} + \mathrm{s}[2j] + \mathrm{c}[2j]) 4^j,$$

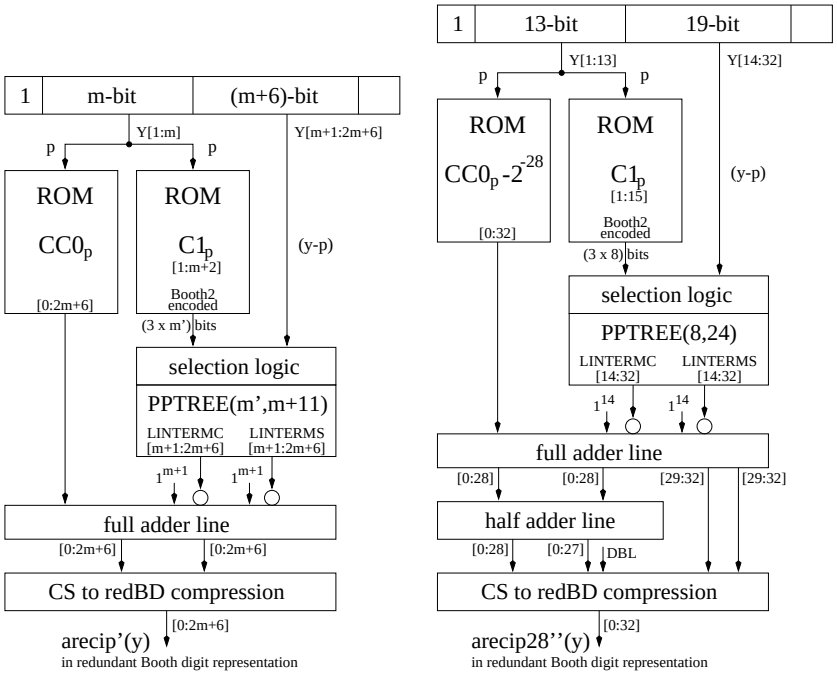

Figure 4.33: Structure of the reciprocal approximation implementation for the computation of: (a) $arecip16'(y)$ with $m = 7$ and $arecip08'(y)$ with $m = 3$; (b) for $arecip28''(y)$.

so that the string of tripels $(\overline{\mathrm{S}[2j+1]}, \mathrm{S}[2j], \mathrm{C}[2j])_{0 \le j \le m'-1}$ is a redundant Booth-digit representation of b. Thus, a partial compression from a carry-save representation of $b + compinj$ to a redundant Booth-digit representation of b can be implemented by a line of 2-bit adders with inverted most significand sum bit outputs like depicted in figure 4.32.

□

4.4.1.3 Implementation

In this section the implementations of the three initial reciprocal approximations that implement the equations for $arecip28$, $arecip16$ and $arecip08$ (see equations 4.299-4.301) are described. These implementations have to compute a multiplication of $C1_p$ and $rnd_{RZ,wy}(y-p)$. For this multiplication we use Booth encoding. Because we read one of the operands from a ROM and this operand is not required for anything else than this multiplication, we already save this multiplicand in its Booth encoded form in the ROM. Because in this way the Booth encoders for the multiplication are not required, this technique saves cost and delay for this multiplication. In [36, 39], we encode this operand even by Booth3 recoding, which further accelerates the computations, but to simplify the description in this work, here only Booth2 recoding is used. The multiplication of $C1_p = <\mathrm{C1}_p[1:m+2]>_{neg}$ and $rnd_{RZ,wy}(y-p) = <\mathrm{Y}[m+1:2m+6]>_{neg}$ results in a positive product, that can be written as

$$\begin{aligned} linterm &= C1_p \cdot rnd_{RZ,wy}(y-p) \\ &= <\mathrm{C1}_p[1:m+2]>_{neg} \cdot <\mathrm{Y}[m+1:2m+6]>_{neg} \\ &= <\mathrm{LINTERM}[m+1:3m+8]>_{neg}. \end{aligned}$$

To compute $arecip(y)$, we have to consider the difference $C0_p - linterm$, and we only have the carry-save representation of

$$\begin{aligned} linterm &= lintermc + linterms \\ &= <\text{LINTERMC}[m+1:3m+8]>_{neg} + <\text{LINTERMS}[m+1:3m+8]>_{neg}. \end{aligned}$$

Because the computation of the bit positions $[0:m+5]$ of the binary representation of $arecip(y)$ includes a truncation after the bit position $[m+5]$ with an truncation error bounded by $2^{-(m+5)}$ and the truncation error of the truncation of a carry-save representation after bit position $[m+6]$ is also at most $2^{-(m+5)}$, we can also consider the positions $[0:m+6]$ of the carry-save representation of $arecip(y)$ to achieve the same absolute error bounds according to corollary 4.35 (we denote the corresponding approximation by $arecip'(y)$). By the use of two's complement number representations we get:

$$\begin{aligned} arecip'(y) &= <(01, \text{C}0_p[2:2m+6])>_{2neg} - \\ &\quad - <\text{LINTERMC}[m+1:2m+6]>_{2neg} - <\text{LINTERMS}[m+1:2m+6])>_{2neg} \\ &= <(01, \text{C}0_p[2:2m+6])>_{2neg} + <(1^{m+1}, \overline{\text{LINTERMC}[m+1:2m+6]})>_{2neg} + \\ &\quad + <(1^{m+1}, \overline{\text{LINTERMS}[m+1:2m+6]})>_{2neg} + 2 \cdot 2^{-2m-6} \end{aligned}$$

We store the binary representation of $CC0_p = C0_p + 2 \cdot 2^{-2m-6} + compinj$ in the ROM for the constant parameter. If we compress the inverted carry-save representation of $linterm$ with the bit strings $(1^{m+1}, \overline{\text{LINTERMC}[m+1:2m+6]})$ and $(1^{m+1}, \overline{\text{LINTERMS}[m+1:2m+6]})$ and the binary representation of $CC0_p = <\text{CC}0_p[0:2m+6]>_{neg}$ by a full-adder-line, we get a carry-save representation of $arecip'(y) + compinj$. A redundant Booth-Digit representation of $arecip'(y)$ can then easily be computed by the partial compression technique from the previous section according to lemma 4.36.

In this way, the reciprocal approximation is implemented in the following steps:

- Table lookup of the binary representation of $CC0_p = <\text{CC}0_p[0:2m+6]>_{neg}$ and the Booth2 encoded representation of $C1_p$ from two ROMs with the input of $\text{Y}[1:m]$.
- Multiplication of the Booth2 encoded representation of $C1_p$ and $\text{Y}[m+1:2m+6]$ by selection logics and the partial product reduction with an adder tree.
- Addition of the carry-save representation from the output of the adder tree with the binary representation of $CC0_p$ by a full-adder line.
- Partial compression of the carry-save representation from the output of the full-adder line to a redundant Booth-digit representation of the reciprocal approximation according to lemma 4.36.

Figure 4.33(a) depicts the structure of the implementations for the redundant Booth-Digit representations of $arecip16'(y)$ and $arecip08'(y)$. In the case of single precision, not the approximation $arecip28'$, but $arecip28''(y) = arecip28'(y) - \overline{\text{DBL}} \cdot 2^{-28}$ will be required to guarantuee a positive error $0 < (arecip''28(y) - 1/y) = (err(y) - 2^{-28}) < 2^{-27}$ in single precision. For this reason, the implementation of $arecip''28$ is slightly changed (see figure 4.33(b)). Note, that for double precision, we have $arecip28''(y) = arecip28'(y)$.

4.4.2 Division I (normalized $\longrightarrow$ representative format)

Specification. This section describes a FP division implementation, that is able to divide two FP numbers given in the normalized representations (section 2.6.3):

$$BUSa_{NF}[69:0] = (\text{SA}, \text{EA}[11:0], \text{FA}[0:52], \text{ZEROA}, \text{INFA}, \text{QNANA}, \text{SNANA}) \quad (4.308)$$
$$BUSb_{NF}[69:0] = (\text{SB}, \text{EB}[11:0], \text{FB}[0:52], \text{ZEROB}, \text{INFB}, \text{QNANB}, \text{SNANB}), \quad (4.309)$$

which represent the factorings $(sa, ea, fa) = fact_{NF}(BUSa_{NF}[69:0])$ and $(sb, eb, fb) = fact_{NF}(BUSb_{NF}[69:0])$. Additionaly, we get as input the bit DBL, which signals the case of double precision by $\text{DBL} = 1$, and an active bit $\text{ISDIV} = 1$, that signals the case, that the operation which is actually perfomed is a division.

In the case, that both operands have representable values and the second operand is non-zero with $\text{ZEROB} = 0$, the exact quotient $exact_{div}$ is defined by (section 2.2.4):

$$exact_{div} = (-1)^{\text{SA}\oplus\text{SB}} \cdot 2^{ea-eb} \cdot fa \cdot 1/fb. \quad (4.310)$$

If (s_{rc}, e_{rc}, f_{rc}) is a RF factoring of this exact quotient $exact_{div}$ for non-zero representable inputs, then for the general case of arbitrary input values, a RF factoring of the required quotient is given by (see equation 2.17):

$$(s_{RF}, e_{RF}, f_{RF}) = \begin{cases} (0, e_{qNaN}, f_{qNaN}) & \text{if SCQNAN} \\ (s_{inf}, e_{\infty}, f_{\infty}) & \text{if SCINF} \\ (sa, ea, fa) & \text{if SCX} \\ (sb, eb, fb) & \text{if SCY} \\ (s_0, e_0, 0) & \text{if SCZERO} \\ (s_{rc}, e_{rc}, f_{rc}) & \text{otherwise.} \end{cases} \quad (4.311)$$

The quotient output of the division I implementation is then specified by the corresponding representation in the representative format $BUS_{RF}[73:0] = \text{RF}(s_{RF}, e_{RF}, f_{RF})$. Moreover, in the the division I implementation the exception flags INV and DVZ should be signaled according to the occurance of an invalid exception and the occurance of a division by zero, respectively.

Implementation. Because we will consider a multiplicative implementation of the significand quotient that shares the fixed-point multiplier with the FP multiplication implementation, the whole division I implementation will be integrated into the multiplication unit I.

The equations for the special conditions in equation 4.311 are already summarized in section 2.4.4 by equations 2.34-2.40. Among these equations, only the equations for the conditions SCINF and SCZERO differ from the equations 2.27-2.33 for the special conditions for FP multiplication. Thus, we change the computations of these two signals in the special cases circuit according to:

$$\text{SCINF} = \begin{cases} \overline{\text{SCQNAN}} \wedge \overline{\text{SCX}} \wedge \overline{\text{SCY}} \wedge (\text{INFA} \vee \text{ZEROB}) & \text{if ISDIV} \\ \overline{\text{SCQNAN}} \wedge \overline{\text{SCX}} \wedge \overline{\text{SCY}} \wedge (\text{INFA} \vee \text{INFB}) & \text{otherwise} \end{cases} \quad (4.312)$$

$$\text{SCZERO} = \begin{cases} \overline{\text{SCQNAN}} \wedge \overline{\text{SCX}} \wedge \overline{\text{SCY}} \wedge (\text{INFB} \vee \text{ZEROA}) & \text{if ISDIV} \\ (\text{ZEROA} \wedge \text{ZEROB}) \vee \overline{(\text{SCQNAN} \vee \text{SCX} \vee \text{SCY})} & \text{otherwise.} \end{cases} \quad (4.313)$$

Based on the selection of the special cases results in the special cases circuit of the multiplication unit according to

$$(s_{sc}, e_{sc}, f_{sc}) = \begin{cases} (0, e_{qNaN}, f_{qNaN}) & \text{if SCQNAN} \\ (s_{inf}, e_{\infty}, f_{\infty}) & \text{if SCINF} \\ (sa, ea, fa) & \text{if SCX} \\ (sb, eb, fb) & \text{if SCY} \\ (s_0, e_0, 0) & \text{otherwise,} \end{cases} \tag{4.314}$$

we get the final division result also by the selection

$$(s_{RF}, e_{RF}, f_{RF}) = \begin{cases} (s_{sc}, e_{sc}, f_{sc}) & \text{if SPCA} \\ (s_{rc}, e_{rc}, f_{rc}) & \text{otherwise.} \end{cases} \tag{4.315}$$

Like for multiplications, also for divisions the invalid flag is given by DVZ $\Longleftrightarrow$ SCQNAN (compare tables 2.8 and 2.9), so that the implementation for INV from the multiplication unit does not have to be changed. The case of a division by zero can only occur for divisions with ZEROA $= 0$ and ZEROB $= 1$. Thus, the flag DVZ is computed in the special cases circuit of the multiplication unit by

$$\text{DVZ} = \text{ISDIV} \wedge \overline{\text{ZEROA}} \wedge \text{ZEROB}.$$

This already completes of the description of the computations for the special cases and the detection of the exceptions.

In the following the computation of the RF factoring (s_{rc}, e_{rc}, f_{rc}) for divisions of non-zero representable operands (regular case) is described. According to equation 4.310, for the regular case the exact quotient can be written as

$$\begin{aligned} exact_{div} &= (-1)^{\text{SA}\oplus\text{SB}} \cdot 2^{ea-eb} \cdot fa \cdot 1/fb \\ &= (-1)^{\text{SA}\oplus\text{SB}} \cdot 2^{ea-eb-1} \cdot 2 \cdot fa \cdot 1/fb. \end{aligned}$$

Because the significands fa and fb were extracted from normalized representations and because the operands are non-zero in the regular case, we have $fa \in [1, 2[$ and $fb \in [1, 2[$. From this it follows that the quotient $2 \cdot fa \cdot 1/fb$ is in the range $]1, 4[$. In this way, the factoring $(sa \oplus sb, ea - eb - 1, rep_{53}(2 \cdot fa \cdot 1/fb)$ fulfills the requirements of RF factorings according to definition 2.21 and a RF factoring of the exact quotient $exact_{div}$ is given by:

$$(s_{rc}, e_{rc}, f_{rc}) = (sa \oplus sb, ea - eb - 1, rep_p(2 \cdot fa \cdot 1/fb)).$$

The computation of this factoring is described in the following separately for the sign, the exponent and the significand.

The computation of s_{rc} according to equation 4.316 is identical to the sign computation from the multiplication unit I, so that the sign computation of the multiplication I unit can be used unchanged also for the division I implementation.

With the 13-bit wide representations of the exponents $ea = <(\text{EA}[11], \text{EA}[11:0])>_2$ and $eb = <(\text{EB}[11], \text{EB}[11:0])>_2$. we exactly get the exponent e_{rc} for divisions (where ISDIV $= 1$) as required according to equation 4.316 by

$$\begin{aligned} e_{rc} &= ea - eb - 1 \\ &= <(\text{EA}[11], \text{EA}[11:0])>_2 + <(\text{EB}[11] \oplus \text{ISDIV}, \text{EB}[11:0] \oplus \text{ISDIV})>_2 \end{aligned}$$

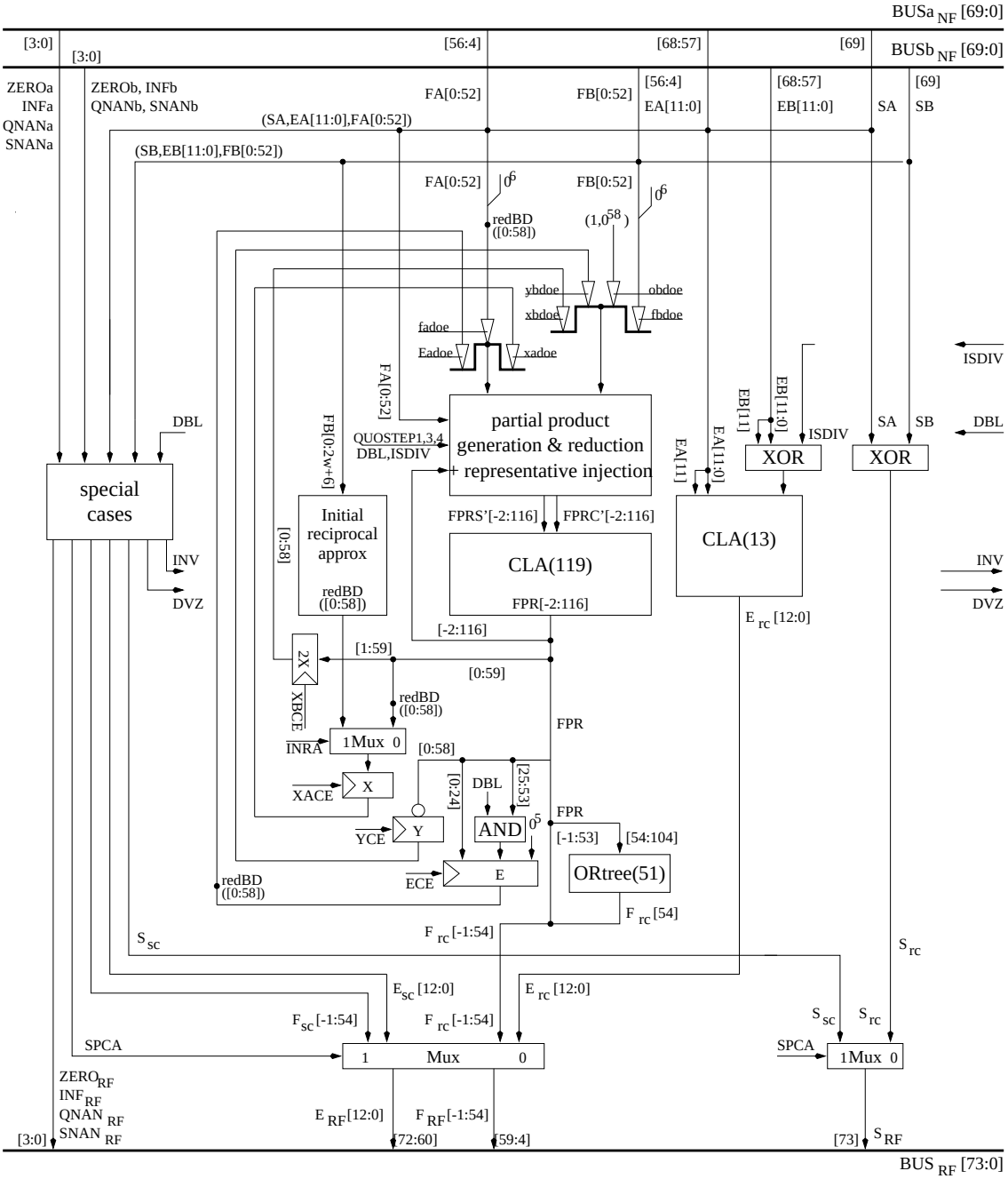

Figure 4.34: Structure of the integrated multiplication/division unit I.

and the exponent e_{rc} for multiplications (where $\text{ISDIV} = 0$) as required according to equation 4.213 by

$$\begin{aligned} e_{rc} &= <(\text{EA}[11], \text{EA}[11:0])>_2 + <(\text{EB}[11] \oplus \text{ISDIV}, \text{EB}[11:0] \oplus \text{ISDIV})>_2 \\ &= ea + eb. \end{aligned}$$

Thus, the exponent implementation from the multiplication I unit can be adopted to be usable for both the division I and the multiplication I implementation, if we replace the

second input ($\text{EB}[11], \text{EB}[11:0]$) of the exponent addition from the multiplication unit I by ($\text{EB}[11] \oplus \text{ISDIV}, \text{EB}[11:0] \oplus \text{ISDIV}$). This implementation for the exponent is depicted in figure 4.34 completing the description of the exponent computations and leaving the description for the computations of the significand f_{rc}.

The significand $f_{rc} = rep_{53}(2 \cdot fa \cdot 1/fb)$ is computed in two steps:

1. We first compute an approximation $aquot \approx (2 \cdot fa \cdot 1/fb)$ with an approximation error $errquot = (2fa/fb - aquot)$ in the range $0 \leq errquot < 2^{-p}$. Note, that in this case the signed value of the error and not only the absolute value has to be bounded.

2. in the second step, the 53-representative $f_{rc} = rep_{53}(2 \cdot fa \cdot 1/fb)$ is computed from the approximation $aquot$.

The implementation of these two steps is described in the following two subsection:

4.4.2.1 Approximation of the quotient (step 1.)

In this subsubsection we describe how to compute the approximation of the quotient $aquot \approx (2 \cdot fa/fb)$. We extract the problem of finding an approximation for the reciprocal of a significand fb first. By the multication of the approximated reciprocal with $2 \cdot fa$ we will then get the approximation $aquot \approx (2 \cdot fa/fb)$ of the quotient.

For the approximation of the reciprocal $1/fb$ we use the Newton-Raphson iteration with an initial approximation of $1/fb$ that is computed with the implementations from section 4.4.1. Because we can assume the significand fb to be normalized $fb \in [1, 2[$, the reciprocal $1/fb$ is known to be in the range $]0.5, 1]$. From an approximation x_i of the reciprocal $x = 1/fb$, the Newton-Raphson algorithm iteratively determines a better approximation x_{i+1} by the equation:

$$x_{i+1} = x_i(2 - fb \cdot x_i). \tag{4.316}$$

We define the approximation error after the iteration i by $\delta_i = 1/fb - x_i$. After the iteration $i + 1$, we then get the approximation error

$$\begin{aligned}
\delta_{i+1} &= 1/fb - x_{i+1} \\
&= 1/fb - (x_i(2 - fb \cdot x_i)) \\
&= fb(\frac{1}{fb^2} - \frac{2x_i}{fb} + {x_i}^2) \\
&= fb(\frac{1}{fb} - x_i)^2 \\
&= fb \cdot {\delta_i}^2,
\end{aligned}$$

so that because of $fb \in [1, 2[$, we get

$${\delta_i}^2 \leq \delta_{i+1} < 2 \cdot {\delta_i}^2.$$

Thus, starting with an accurate initial reciprocal approximation, the approximation error converges quadratically with the number i of iterations.

According to equation 4.316 the following two dependent multiplications are required for the computation of each Newton-Raphson iteration:

$$\begin{aligned}
y_i &= fb \cdot x_i \\
x_{i+1} &= x_i \cdot (2 - y_i)
\end{aligned}$$

In an exact computation the number of significant bit positions increases after each of these multiplications, so that already after a few iterations, we would have to handle very long operands and require a very large multiplier. To avoid this problem, we limit the number of significant bit positions and truncate each product after a fixed bit position $wdiv$ with $wdiv \geq p$. Thus, we consider

$$y'_i = rnd_{RZ,wdiv}(fb \cdot x'_i) \tag{4.317}$$
$$x'_{i+1} = rnd_{RZ,wdiv}(x'_i \cdot (2 - y'_i)) \tag{4.318}$$

where the values x'_{i+1}, x'_i, y'_i and fb can be represented in a binary representation with bit positions $[0\!:\!wdiv]$. For $y'_i < 2$, the difference in equation 4.318 can be written as

$$\begin{aligned}(2 - y'_i) &= 2 - <\mathrm{Y}'_i[0\!:\!wdiv]>_{neg} \\ &= <\overline{\mathrm{Y}'_i[0\!:\!wdiv]}>_{neg} + 2^{wdiv}.\end{aligned}$$

We simplify the computation of this difference by neglecting the addition of the 2^{wdiv} and only compute the approximation

$$x''_{i+1} = rnd_{RZ,wdiv}(x''_i \cdot <\overline{\mathrm{Y}'_i[0\!:\!wdiv]}>_{neg}).$$

In the analysis of the approximation error for x''_{i+1}, which we denote by $\delta''_{i+1} = 1/fb - x''_{i+1}$, we now additionaly have to consider the errors due to the product truncations and due to the lazy computation of the difference $(2 - y'_i)$. Each product truncation produces a discretization error in the range $[0, 2^{-wdiv}[$, so that because of $x'_i \leq 1$ we get

$$\begin{array}{rcccl}
y'_i + 2^{-wdiv} & > & y_i & \geq & y'_i \\
(2 - y'_i - 2^{-wdiv}) & < & (2 - y_i) & \leq & (2 - y'_i) \\
<\overline{\mathrm{Y}'_i[0\!:\!wdiv]}>_{neg} & < & (2 - y_i) & \leq & <\overline{\mathrm{Y}'_i[0\!:\!wdiv]}>_{neg} + 2^{-wdiv} \\
x''_i \cdot (<\overline{\mathrm{Y}'_i[0\!:\!wdiv]}>_{neg}) & < & x_i \cdot (2 - y_i) & \leq & x''_i \cdot (<\overline{\mathrm{Y}'_i[0\!:\!wdiv]}>_{neg} + 2^{-wdiv}) \\
x''_{i+1} & < & x_{i+1} & < & x''_{i+1} + 2^{-wdiv}
\end{array}$$

Note, that in the above error analysis of one iteration step, we consider $x_i = x'_i = x''_i$ and $\delta_i = \delta'_i = delta''_i$. Thus, we get for the approximation error $\delta''_{i+1} = 1/fb - x''_{i+1}$ after one iteration step:

$$\begin{array}{rcccl}
x_{i+1} & > & x''_{i+1} & > & x_{i+1} - 2^{-wdiv} \\
1/fb - x_{i+1} & < & 1/fb - x''_{i+1} & < & 1/fb - x_{i+1} + 2^{-wdiv} \\
\delta_{i+1} & < & \delta''_{i+1} & < & \delta_{i+1} + 2^{-wdiv} \\
fb \cdot \delta_i^2 & < & \delta''_{i+1} & < & fb \cdot \delta_i^2 + 2^{-wdiv} \\
\delta_i^2 & < & \delta''_{i+1} & < & 2 \cdot \delta_i^2 + 2^{-wdiv}
\end{array}$$

For the approximation of the significand quotient, we have to multiply a reciprocal approximation x''_i by $(2fa)$: $aquot_i = 2fa \cdot x''_i$. Because $2fa \in [2, 4[$, in this way the approximation error for the quotient approximation $errquot_i = 2fa/fb - aquot_i$ is in the range

$$2 \cdot \delta_i < errquot_i < 4 \cdot \delta_i \tag{4.319}$$
$$2 \cdot \delta_{i-1}^2 < errquot_i < 8 \cdot \delta_{i-1}^2 + 4 \cdot 2^{wdiv} \tag{4.320}$$
$$2 \cdot \delta_{i-2}^4 < errquot_i < 16 \cdot \delta_{i-2}^4 + 12 \cdot 2^{wdiv} \tag{4.321}$$
$$2 \cdot \delta_{i-3}^8 < errquot_i < 32 \cdot \delta_{i-3}^8 + 28 \cdot 2^{wdiv} \tag{4.322}$$

		requirements for			fulfilled for	
precision	i	$errquot_i <$	$\|\delta_0\| <$	$wdiv \geq$	$x_0 =$	$wdiv =$
double	3	2^{-53}	$2^{-61/8}$	58	$arecip08'$	58
single	2	2^{-24}	$2^{-29/4}$	29	$arecip08'$	58
double	2	2^{-53}	$2^{-59/4}$	57	$arecip16'$	58
single	1	2^{-24}	2^{-14}	28	$arecip16'$	58
double	1	2^{-53}	2^{-28}	56	$arecip28''$	58
single	0	2^{-24}	2^{-26}	32	$arecip28''$	58

Table 4.3: Requirements on the initial reciprocal approximations using the Newton-Raphson iteration with $i \in \{1, 2, 3\}$ iterations in double precision and $i \in \{0, 1, 2\}$ iterations in single precision.

For the computation of the p-representative in step 2, we require an approximation of the quotient $aquot_i$ in the range $0 < errquot_i < 2^{-p}$. We are interested in approximations that are computed after $i \in \{0, 1, 2, 3\}$ and have to know the initial approximation error δ_0, that is required for the initial reciprocal approximation. Because for $i > 0$, we know that $errquot_i > 0$, we only have to consider the upper bound on the absolute initial approximation error δ_0 for $i > 0$. To get into the target range for $errquot_i$, the truncation position $wdiv$ has to fulfill $wdiv \geq p + 5$ for $i = 3$ (see equation 4.322). To fulfill this condition for single and double precision, we set $wdiv = 58$ in the following. The requirements for the initial reciprocal approximations are listed in table 4.3. Thus, the initial reciprocal approximation $arecip08'(fb)$ can be used for the computation of an appropriate quotient approximation $aquot_i$ after $i = 2$ iterations for single precision and after $i = 3$ iterations for double precision. The initial reciprocal approximation $arecip16'(fb)$ can be used for the computation of an appropriate quotient approximation $aquot_i$ after $i = 1$ iteration for single precision and after $i = 2$ iterations for double precision. The initial reciprocal approximation $arecip28''(fb)$ can be used for the computation of an appropriate quotient approximation $aquot_i$ after $i = 0$ iterations for single precision and after $i = 1$ iteration for double precision. Note, that the use of $arecip28''$ instead of $arecip28'$ guarantuees a positive error $err28'' > 0$, so that also for this case the lower bound on the approximation error $aquot_0$ fulfills the requirements for the computation of the representative in step 2.

4.4.2.2 Computation of the p-representative for f_{rc} (step 2.)

From the quotient approximations in the previous section we get

$$\begin{array}{rcccl} 0 & < & errquot_i = 2fa/fb - aquot_i & < & 2^{-p} \\ aquot_i & < & 2fa/fb & < & aquot_i + 2^{-p} \end{array}$$

Thus,

$$rnd_{RZ,p}(aquot_i) \;<\; aquot_i \;<\; 2fa/fb \;<\; aquot_i + 2^{-p} \;<\; rnd_{RZ,p}(aquot_i) + 2^{-p+1}.$$

In other words,

$$E = rnd_{RZ,p}(aquot_i)$$

is an approximation of $q = 2fa/fb$, and the exact quotient lies in the open interval $(E, E + 2^{-p+1})$. Moreover, we have

$$rep_p(2fa/fb) = \begin{cases} E + 2^{-(p+1)} & \text{if} \quad 2fa/fb < E + 2^{-p} \\ E + 2^{-(p)} & \text{if} \quad 2fa/fb = E + 2^{-p} \\ E + 3 \cdot 2^{-(p+1)} & \text{if} \quad 2fa/fb > E + 2^{-p} \end{cases}$$

For any relation $\circ \in \{<, =, >\}$ we have

$$2fa/fb \circ E + 2^{-p} \iff 0 \circ fb \cdot (E + 2^{-p}) - fa.$$

Thus, with the computation of $g = fb \cdot (E + 2^{-p}) - fa$ and the conditions

$$\begin{aligned} \text{REPZERO} &\iff (g = 0) \\ \text{REPNEG} &\iff (g > 0) \end{aligned}$$

the representative $f_{rc} = rep_p(2fa/fb)$ can be selected by

$$rep_p(2fa/fb) = \begin{cases} E + 2^{-(p+1)} & \text{if} \quad \overline{\text{REPZERO}} \wedge \overline{\text{REPNEG}} \\ E + 2^{-(p)} & \text{if} \quad \text{REPZERO} \wedge \overline{\text{REPNEG}} \\ E + 3 \cdot 2^{-(p+1)} & \text{if} \quad \text{REPZERO} \wedge \text{REPNEG} \end{cases}$$

For this computation of $rep_p(2fa/fb)$, we define the representative increment *repinc*:

$$\begin{aligned} repinc &= < \text{REPINC}[-2:54] >_{neg} \\ &= \begin{cases} E + 2^{-(p+1)} & \text{if} \quad \overline{\text{REPZERO}} \wedge \overline{\text{REPNEG}} \\ E + 2^{-(p)} & \text{if} \quad \text{REPZERO} \wedge \overline{\text{REPNEG}} \\ E + 3 \cdot 2^{-(p+1)} & \text{if} \quad \text{REPZERO} \wedge \text{REPNEG} \end{cases} \\ &= < (000, 0^{23}, (\text{REPZERO} \vee \text{REPNEG}) \wedge \overline{(}\text{DBL}), (\overline{\text{REPZERO}} \vee \text{REPNEG}) \wedge \overline{(}\text{DBL}), \\ &\quad , 0^{27}, (\text{REPZERO} \vee \text{REPNEG}) \wedge \text{DBL}, (\overline{\text{REPZERO}} \vee \text{REPNEG}) \wedge \text{DBL}) >_{neg}, \end{aligned}$$

so that $rep_p(2fa/fb) = E + repinc$. The value $g = fb \cdot (E + 2^{-p}) - fa$ is computed in two steps. We first compute

$$E_b = E + 2^{-(p)}$$

Then we compute $g = E_b \cdot fb - fa$. Again, we define an additive constant, that selects the additive operands also including the case of the representative by

$$\begin{aligned} repinj &= < \text{REPINC}[-2:117] >_{neg} \\ &= \begin{cases} 2^{-p} & \text{if QUOSTEP1} \\ -fa & \text{if QUOSTEP3} \\ repinc & \text{if QUOSTEP4} \\ 0 & \text{otherwise.} \end{cases} \end{aligned}$$

Thus, we get for QUOSTEP1 = 1:

$$E_b = E + repinj \tag{4.323}$$

we get for QUOSTEP3 = 1:

$$g = E_b \cdot fb + repinj \tag{4.324}$$

significand quotient			sequence implementation version					
step	computation	control signals	*i*	*ii*	*iii*	*iv*	*v*	*vi*
init $\approx 1/fb$	$x_0'' = arecip(fb)$	INRA,XACE,XBCE	1	1	1	1	1	1
Newton 1a	$y_i' = x_i'' \cdot fb$	$xadoe$, $fbdoe$, DBL	2, 8	2, 6	2	2	–	–
Newton 1b	"	$xadoe$, $fbdoe$, DBL, 2ND	3, 9	–	3	–	–	–
Newton 1c	"	YCE	4, 10	3, 7	4	3	–	–
Newton 2a	$x_{i+1}'' = x_i'' \cdot \overline{y_i}$	$xadoe$, $ybdoe$, DBL	5, 11	4, 8	5	4	–	–
Newton 2b	"	$xadoe$, $ybdoe$, DBL, 2ND	6, 12	–	6	–	–	–
Newton 2c	"	XACE	7, 13	5, 9	7	5	–	–
Quot 1a	$E_b = fa \cdot 2x_{i+1}'' + repinj$	$fadoe$, $xbdoe$, DBL, QUOSTEP1	14	10	8	6	2	2
Quot 2a	$g = E_b \cdot fb + repinj$	$fadoe$, $xbdoe$, DBL, ECE	15	11	9	7	3	3
Quot 3a	$g = E_b \cdot fb + repinj$	$Eadoe$, $fbdoe$, DBL, ECE, QUOSTEP3	16	12	10	8	4	4
Rep Sel a	$f_{rc} = E \cdot 1 + repinj$	$fadoe$, $obdoe$, DBL, QUOSTEP4	17	13	11	9	5	5
Rep Sel c	"	-	18	14	12	10	6	6

Table 4.4: Computation steps in the six different implementations for the computation of the significand quotient representative in single precision.

and we get for QUOSTEP4 = 1:

$$f_{rc} \quad = \quad rep_p(2fa/fb) = E \cdot 1 + repinj. \tag{4.325}$$

After this last step the path for the computation of f_{rc} for the multiplication contains also the significand f_{rc} for the quotient result. To control the steps of the division we define the control signals XACE, XBCE, ECE, YCE, INRA, $Eadoe$, $fadoe$, $xadoe$, $xbdoe$, $ybdoe$, $obdoe$ and $fbdoe$ that influence the computation paths by controlling drivers and register clocks like depicted in figure 4.34. The changed implementations of the partial product generation and reduction are depicted in figure 4.35(full-sized adder tree) and in figure 4.36(half-sized adder tree). The computation steps including the required values for the control signals are summarized in table 4.4 (single precision) and table 4.5 (double precision) for the six cases: (i) use of $arecip08'(fb)$ and half-sized adder tree; (ii) use of $arecip08'(fb)$ and full-sized adder tree; (iii) use of $arecip16'(fb)$ and half-sized adder tree; (iv) use of $arecip16'(fb)$ and full-sized adder tree; (v) use of $arecip28''fb)$ and half-sized adder tree; (vi) use of $arecip28''(fb)$ and full-sized adder tree. Note, that for multiplications, where ISDIV = 0, the implementations of the RF factoring (s_{rc}, e_{rc}, f_{rc}) are not changed. This completes the description of the integrated multiplication/division I implementations.

significand quotient			sequence implementation version					
step	computation	control signals	*i*	*ii*	*iii*	*iv*	*v*	*vi*
init $\approx 1/fb$	$x''_0 = arecip(fb)$	INRA,XACE	1	1	1	1	1	1
Newton1a	$y'_i = x''_i\ fb$	$xadoe$,$fbdoe$,DBL	2, 8, 14	2, 6, 10	2, 8	2, 6	2	2
Newton1b	"	$xadoe$, $fbdoe$, DBL, 2ND	3, 9, 15	–	3, 9	–	3	–
Newton1c	"	YCE	4, 10, 16	3, 7, 11	4, 10	3, 7	4	3
Newton2a	$x''_{i+1} = x''_i \cdot \overline{y_i}$	$xadoe$,$ybdoe$,DBL	5, 11, 17	4, 8, 12	5, 11	4, 8	5	4
Newton2b	"	$xadoe$, $ybdoe$, DBL, 2ND	6, 12, 18	–	6, 12	–	6	–
Newton2c	"	XACE	7, 13, 19	5, 9, 13	7, 13	5, 9	7	5
Quot 1a	$E_b = fa \cdot 2x''_{i+1}$ $+repinj$	$fadoe$, $xbdoe$, DBL, QUOSTEP1	20	14	14	10	8	6
Quot 1b	$E = fa \cdot 2x''_{i+1}$	$fadoe$, $xbdoe$, DBL, 2ND, QUOSTEP1	21	–	15	–	9	–
Quot 2a	$g = E_b \cdot fb$ $+repinj$	$fadoe$, $xbdoe$, DBL, ECE	22	15	16	11	10	7
Quot 2b	$g = E_b \cdot fb$ $+repinj$	$fadoe$, $xbdoe$, DBL, 2ND	23	–	17	–	11	–
Quot 3a	$g = E_b \cdot fb$ $+repinj$	$Eadoe$, $fbdoe$, DBL, ECE, QUOSTEP3	24	16	18	12	12	8
Quot 3b	$g = E_b \cdot fb$ $+repinj$	$Eadoe$, $obdoe$, DBL, 2ND, QUOSTEP3	25	–	19	–	13	–
Rep Sel a	$f_{rc} = E \cdot 1$ $+repinj$	$fadoe$, $obdoe$, DBL, QUOSTEP4	26	17	20	13	14	9
Rep Sel b	"	$fadoe$, $obdoe$, DBL, 2ND, QUOSTEP4	27	–	21	–	15	–
Rep Sel c	"	-	28	18	22	14	16	10

Table 4.5: Computation steps in the six different implementations for the computation of the significand quotient representative in double precision.

4.4.3 Division II (normalized $\longrightarrow$ gradual result format)

Specification. This section describes a FP division implementation, that is able to divide two FP numbers given in the normalized representations (section 2.6.3):

$$\begin{aligned} BUSa_{NF}[69:0] &= (\text{SA}, \text{EA}[11:0], \text{FA}[0:52], \text{ZEROA}, \text{INFA}, \text{QNANA}, \text{SNANA}) \\ BUSb_{NF}[69:0] &= (\text{SB}, \text{EB}[11:0], \text{FB}[0:52], \text{ZEROB}, \text{INFB}, \text{QNANB}, \text{SNANB}), \end{aligned}$$

which represent the factorings $(sa, ea, fa) = fact_{NF}(BUSa_{NF}[69:0])$ and $(sb, eb, fb) = fact_{NF}(BUSb_{NF}[69:0])$. Additionaly, we get as input the bit DBL, which signals the case of double precision by DBL = 1, and an active bit ISDIV = 1, that signals the case, that the operation which is actually perfomed is a division.

In this section, the exact division result according to equation 4.310 has to be rounded by the general rounding function $ground1$. After this gradual rounding step the quotient

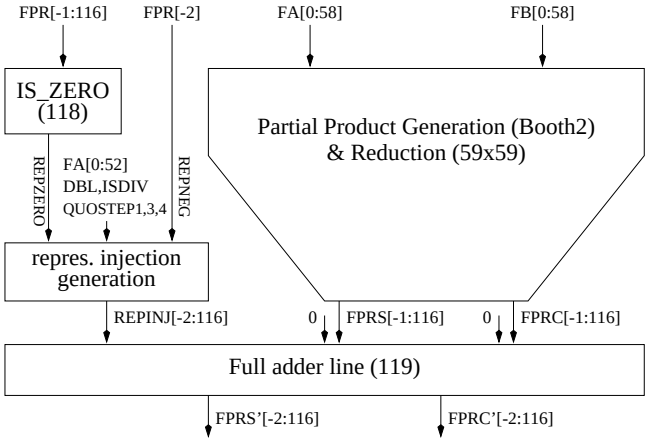

Figure 4.35: Implementation of the full-sized partial product generation and reduction including representative test and injection generation for integrated multiplication/division I/III implementation.

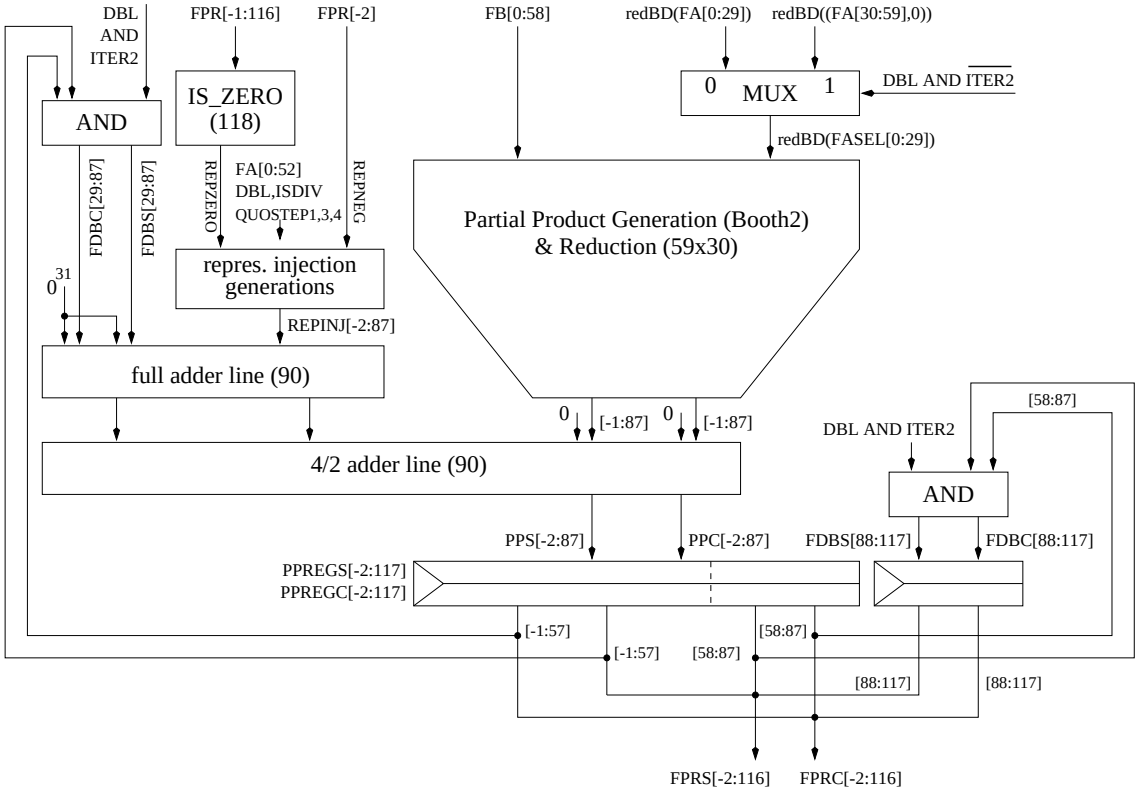

Figure 4.36: Implementation of the half-sized partial product generation and reduction including representative test and injection generation integrated multiplication/division I/III implementation.

should be output in the gradual result format $BUS_{GF}[73:0]$ (section 2.6.5). According to equation 4.316, a RF factoring of the exact product is given by $(s_{rc}, e_{rc}, f_{rc}) = (\text{SA} \oplus \text{SB}, ea - eb - 1, rep_p(2fa \cdot fb))$ for non-zero representable operands. With the gradual rounded product $((s_{grc}, e_{grc}, f_{grc}), \text{TINC}, \text{TINX}) = ground1(s_{rc}, e_{rc}, f_{rc})$ and the following GF factoring of the result for the case of arbitrary IEEE operands

$$((s_{GF}, e_{GF}, f_{GF}), \text{TINC}_{GF}, \text{TINX}_{GF}) = \begin{cases} ((0, e_{qNaN}, f_{qNaN}), 0, 0) & \text{if SCQNAN} \\ ((s_{inf}, e_{\infty}, f_{\infty}), 0, 0) & \text{if SCINF} \\ ((sa, ea, fa), 0, 0) & \text{if SCX} \\ ((sb, eb, fb), 0, 0) & \text{if SCY} \\ ((s_0, e_0, 0), 0, 0) & \text{if SCZERO} \\ ((s_{grc}, e_{grc}, f_{grc}), \text{TINC}, \text{TINX}) & \text{otherwise,} \end{cases} \tag{4.326}$$

the quotient output of the division II implementation is specified by the gradual result representation $BUS_{GF}[73:0] = \text{GF}((s_{GF}, e_{GF}, f_{GF}), \text{TINC}_{GF}, \text{TINX}_{GF})$. The occurance of an invalid exception or a division by zero should be signaled by the bit INV and the bit DVZ also in this section.

Implementation. Like the division I implementation is integrated into the multiplication I unit, we will integrate the implementation of the division II into the multiplication II unit. The changes for the special cases are the same like in the previous section for the integrated multiplication/division I implementation.

Thus, we only have to consider the computations for the regular case. The implementation of the GF factoring of the quotient in this section is just a combination of the computation of the RF factoring of the quotient from the previous section and the implementaton of the gradual rounding function $ground1$ from the multiplication II. The computation of the significand quotient is implemented like in the previous section. By setting the rounding injections to zero during the significand quotient computations by a signal INJSEL = 0 we get in the multiplication II implementation $fpr = finj12$. Thus the binary pruduct output $finj[0:116]$ of the Compression & gradual rounding circuit can be used for the computation of the significand quotient like in the previous section. For the gradual rounding implementation in the last cycle the rounding injection has to be activated again by INJSEL = 0, so that we get as output of the last cycle f_{grc} instead of f_{rc}. The integrated implementation of the multiplication/division implementation is depicted in figure 4.37, where the implementation of the partial product generation & and reduction has to be adopted according to figure 4.38 for the use of a full-sized adder tree and according to figure 4.39 for the use of a half-sized adder tree. This already completes the description of the integrated multiplication/division II implementations.

4.4.4 Division III (normalized $\longrightarrow$ normalized format)

Specification. Like in the previous section also in this section, the FP division is computed from the inputs of the normalized representations $BUSa_{NF}[69:0]$ and $BUSb_{NF}[69:0]$ (section 2.6.3). Because IEEE rounding has to be considered in this section, also the bit DBL, that signals the case of single precision (DBL = 0) or double precision (DBL = 1), the input of the rounding mode, represented by RMODE[1:0], and the underflow and overflow enable bits UNF_EN and OVF_EN are required.

In this section, the exact division result according to equation 4.310 has to be rounded by the rounding function *nround*, that computes the NF factoring of the IEEE rounded result. After this rounding computation the quotient should be output in the normalized

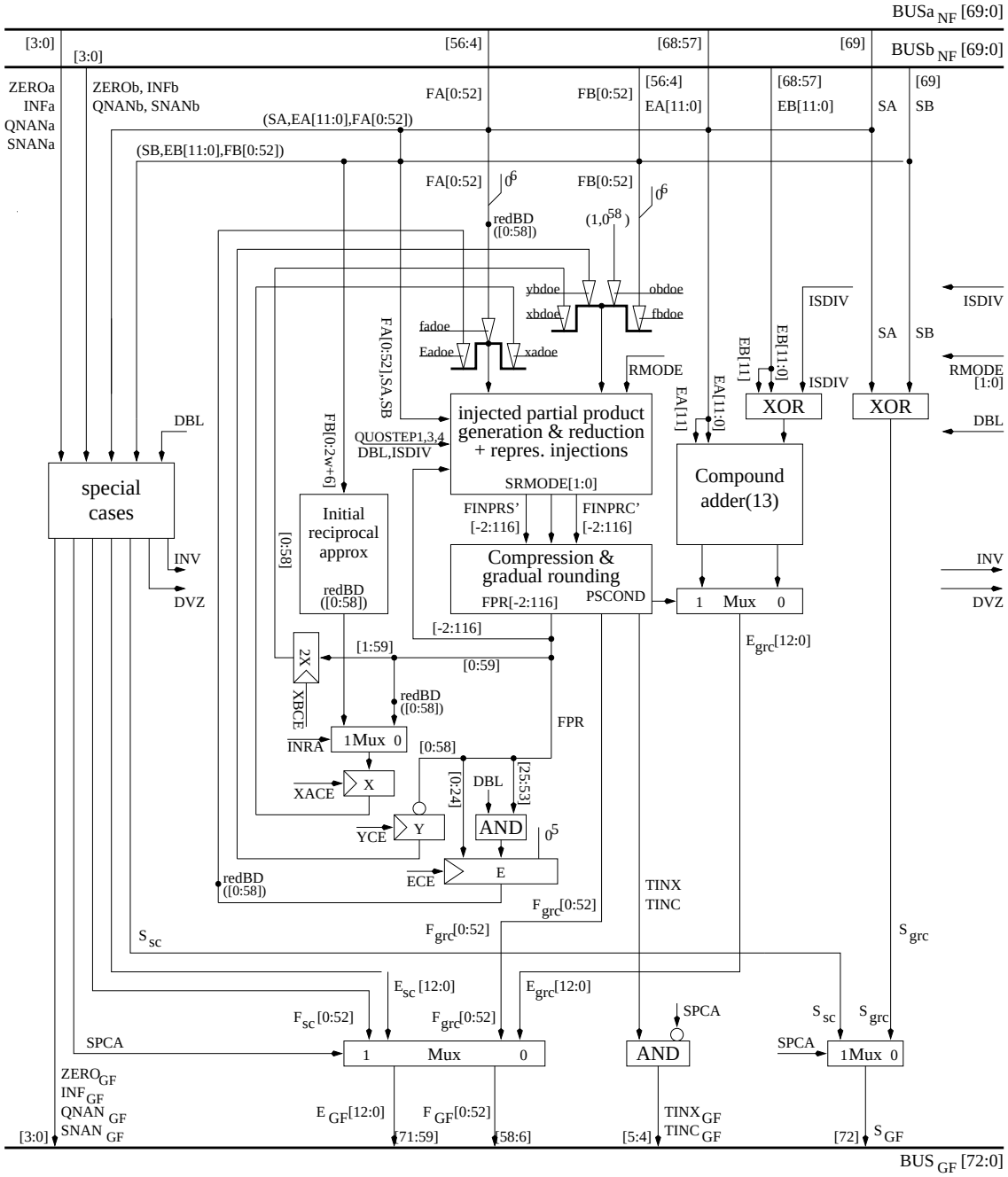

Figure 4.37: Structure of the integrated multiplication/division unit II.

format $BUS_{NF}[69:0]$ (section 2.6.3). According to equation 4.316, a RF factoring of the exact product is given by $(s_{rc}, e_{rc}, f_{rc}) = (\text{SA} \oplus \text{SB}, ea - eb - 1, rep_p(2fa \cdot fb))$ for non-zero representable operands. With the NF factoring of the IEEE result for non-zero representable operands $(s_{nrc}, e_{nrc}, f_{nrc}) = nround(s_{rc}, e_{rc} + wec, f_{rc})$ including the exponent wrapping constant wec according to equation 2.14 and the following NF factoring

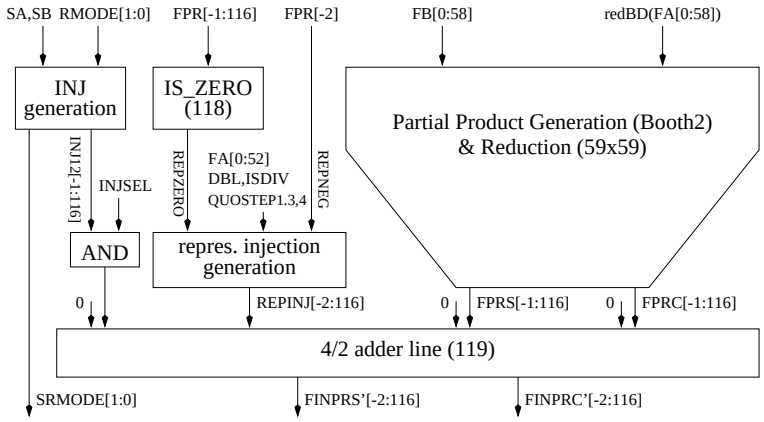

Figure 4.38: Implementation of the half-sized partial product generation and reduction including representative test and injection generation for integrated multiplication/division II implementation.

of the result for the general case of arbitrary operands according to equation 2.16:

$$(s_{NF}, e_{NF}, f_{NF}) = \begin{cases} (0, e_{qNaN}, f_{qNaN}) & \text{if SCQNAN} \\ (s_{inf}, e_{\infty}, f_{\infty}) & \text{if SCINF} \\ (sa, ea, fa) & \text{if SCX} \\ (sb, eb, fb) & \text{if SCY} \\ (s_0, e_0, 0) & \text{if SCZERO} \\ (s_{nrc}, e_{nrc}, f_{nrc}) & \text{otherwise,} \end{cases} \tag{4.327}$$

the quotient output of the division III implementation is specified by the corresponding representation in the normalized format $BUS_{NF}[69:0] = \text{NF}(s_{NF}, e_{NF}, f_{NF})$. In this section, the occurance of an invalid, inexact, overflow, underflow exception should be signaled by the bits INV, INX, OVF, UNF and DVZ, respectively.

Implementation. In an analogous way like in the two previous sections, in this section the implementation of the division III is integrated into the multiplication III unit. Like in the previous section for the special cases computations only the implementation of SCINF, SCZERO and DVZ the have to be changed according to equations 4.312,4.313 and 4.316.

The computations for the regular case are implemented in two steps. First, the RF factoring (s_{rc}, e_{rc}, f_{rc}) is computed like in the integrated division/multiplication I implementation, then the rounding hardware from the multiplication unit III computes the normalized IEEE rounding function *nround* from this RF factoring. To get the binary representation of the product, we have to set $\text{SR\_MODE}[1:0] = 00$ during the computation of the significand quotient, so that the rounding injection is zero and we get $fpr = finj12$. Based on this product output, the significand quotient implementation from the division I implementation is integrated into the multiplication unit III like depicted in figure 4.40. Because the partial product generation and reduction implementation of the multiplication unit III and I are the same, they are changed identically for the integrated division/multiplication implementations III and I. The implementation using a full-sized adder tree is depicted in figure 4.35 and the implementation using a half-sized adder tree is depicted in figure 4.36. During the last two cycles the cleared value of the rounding

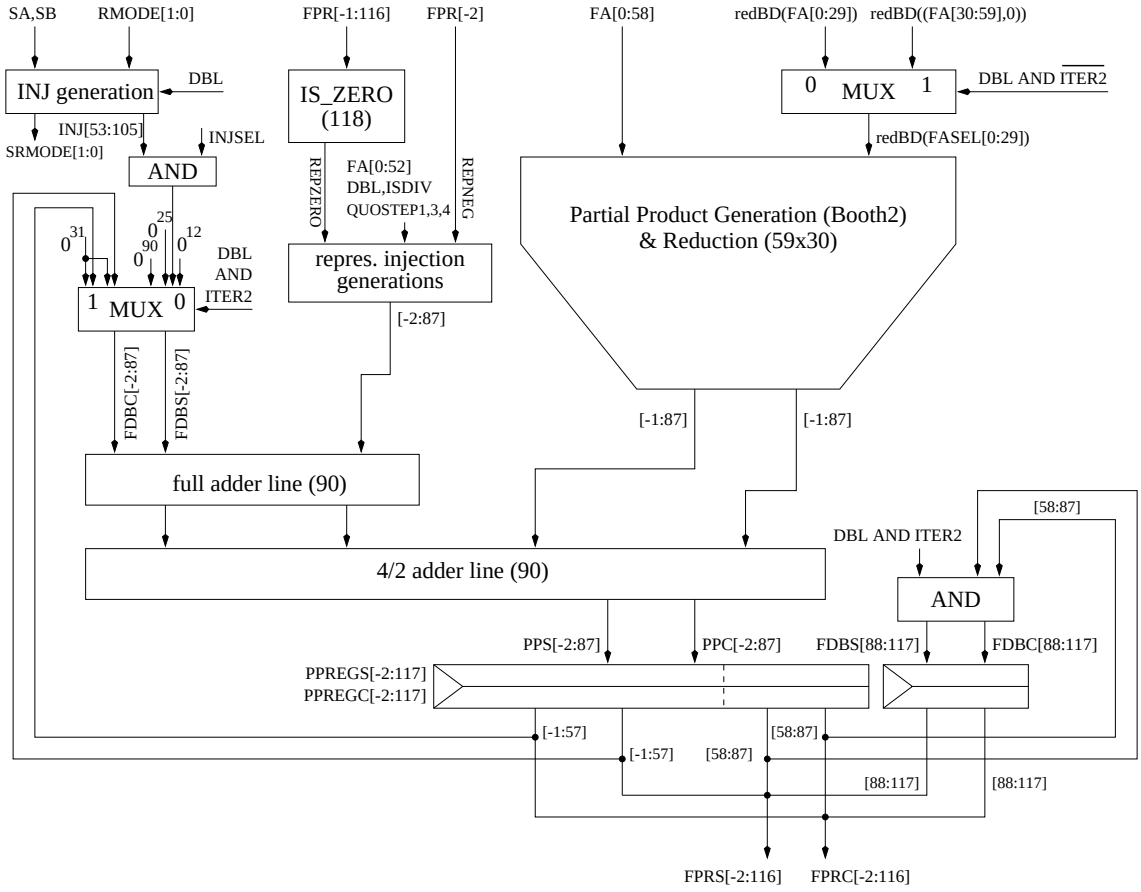

Figure 4.39: Implementation of the half-sized partial product generation and reduction including representative test and injection generation for integrated multiplication/division II implementation.

mode in the bits SR_MODE$[1:0]$ have to be computed from RMODE$[1:0]$ again to guarantuee the correct rounding injection for the computation of the rouning function $nround$ to get the output of $(s_{nrc}, e_{nrc}, f_{nrc})$. This completes the description of the integrated multiplication/division III implementation.

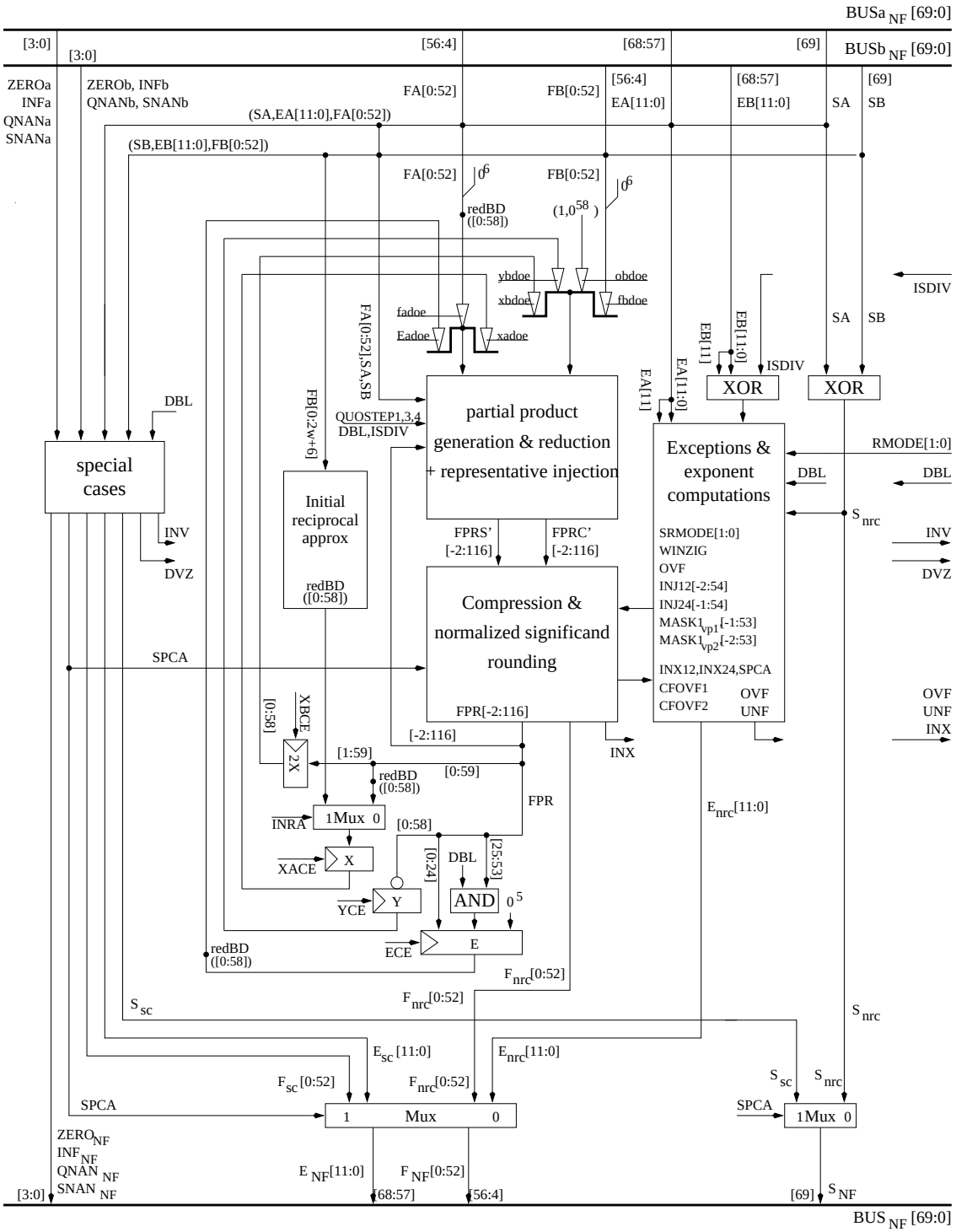

Figure 4.40: Structure of the integrated multiplication/division unit III.

Chapter 5

Evaluation

In this section we quantitatively analyze the FP designs that have been described in the previous sections. For the analysis we use the formal hardware model from [22]. Based on a specification of the FPU designs at gate level, we compute the costs of the designs by counting the gates, that are used in the designs, and by weighting them for a particular, but typical technology [24] (see table 5.1). For any other technology the relative costs of the basic circuits could be changed by the corresponding parameters in the cost formulae. The cost for the FP designs are listed table 5.4. These costs also contain the cost of the pipelined RISC architecture from [23] in that the FP designs are integrated.

The performance of the FP implementations mainly depend on two factors. On the one hand the maximum delay within one cycle of a FP implementations determines the minimum cycle time that would be possible with this FP implementation. We will consider the performance of the FP implementations integrated into a pipelined RISC architecture from [23]. In this setting, a difference between the FP implementations regarding the cycle time only becomes visible, if the FP implementations lie on the critical path and the cycle time of the FP implementation exeeds the cycle time of the microprocessor. This is not the case for any of our FPU designs for the chosen pipelined RISC processor from [23].

Thus, integrated into the microprocessor the performance is measured by the average number of cycles per instructions, that the microprocessor achieves with this FP implementation on an average FP workload. To analyze the performance in this way, we consider a pipelined RISC processor design from [23] as a basis. This design already includes the implementation of pipelining, forwarding, interrupt handling, and a result-shift register [23]. Corresponding to this architecture a trace driven run-time simulator was implemented, so that with the input of the latency and restart-time set of the FPU, the average number of clock cycles that are needed per instruction (CPI) could be simulated. The latencies and restart-times of our proposed FP implementations are listed in table 5.2. For the runtime-simulations, we consider the benchmarks from the SPECfp92 floating-point benchmark suite, because traces using the MIPS R3000 instruction set were already available for them [17].

The results of the analysis are depicted in figure 5.1, where the costs in terms of kG (kilo gates) are displayed against the performances in terms of CPI (cycles per instruction). We separate the results for the three different rounding architectures in three figures. In the topmost figure, the results for the FP units using rounding architecture I with a shared general rounder are depicted, the figure in the middle depicts the results for the FP units using rounding architecture II with a gradual rounder and in the figure on the bottom the results for the FP units using rounding architecture III with variable position rounding

Motorola	Not	$Nand$ Nor	And Or	$Flip-flop$	Xor $Xnor$	Mux	$3-state$ $driver$
delay	1	1	2	2	4	2	2
cost	1	2	2	4	8	3	5

Table 5.1: relative delay and cost of basic gates for the Motorola technology from [24].

FPU	*division*		*multiplication*		*add/sub*	*conv*	*compare*
	double	single	double	single			
Gen rnd I, full, NR28	13/8	9/4	5	5	5	3	1
Gen rnd I, full, NR16	17/12	13/8	5	5	5	3	1
Gen rnd I, full, NR8	21/16	17/12	5	5	5	3	1
Gen rnd I, half, NR28	19/14	9/4	6/2	5	5	3	1
Gen rnd I, half, NR16	25/20	15/10	6/2	5	5	3	1
Gen rnd I, half, NR8	31/26	21/16	6/2	5	5	3	1
Grad rnd II, full, NR28	12/8	8/4	4	4	4	3	1
Grad rnd II, full, NR16	16/12	12/8	4	4	4	3	1
Grad rnd II, full, NR8	20/16	16/12	4	4	4	3	1
Grad rnd II, half, NR28	18/14	8/4	5/2	4	4	3	1
Grad rnd II, half, NR16	24/20	14/10	5/2	4	4	3	1
Grad rnd II, half, NR8	30/26	20/16	5/2	4	4	3	1
Var rnd III, full, NR28	10/8	6/4	2	2	2	3	1
Var rnd III, full, NR16	14/12	10/8	2	2	2	3	1
Var rnd III, full, NR8	18/16	14/12	2	2	2	3	1
Var rnd III, half, NR28	16/14	6/4	3/2	2	2	3	1
Var rnd III, half, NR16	22/20	12/10	3/2	2	2	3	1
Var rnd III, half, NR8	28/26	18/16	3/2	2	2	3	1

Table 5.2: Latencies/restart-times of the FP units for single precision and double precision operations. If there is only one entry, this corresponds to the latency and the operation is fully pipelined.

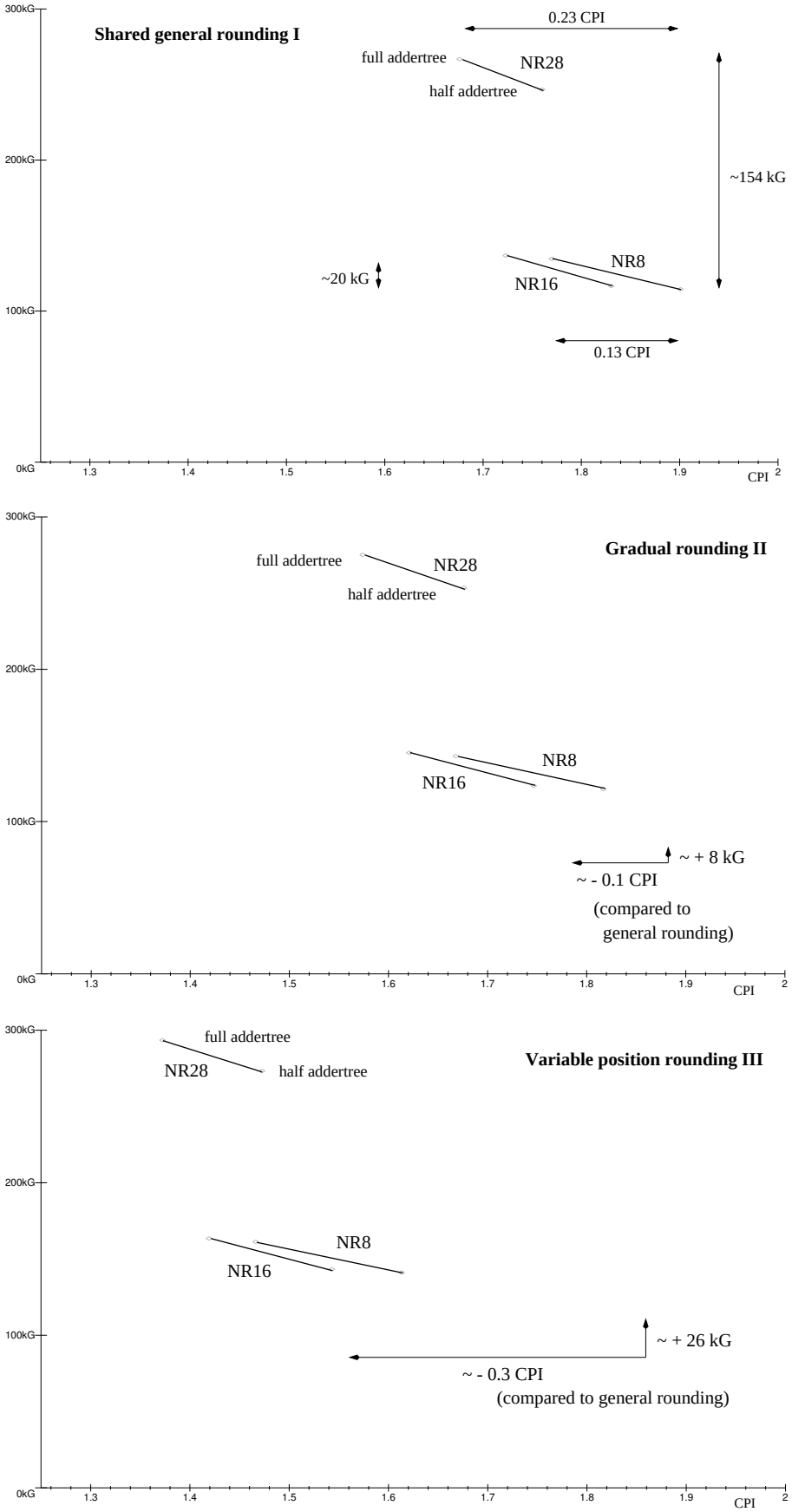

Figure 5.1: Cost (kilo gates) and performance (cycles per instruction) of the different FP designs.

Version	Gen rnd I	Grad rnd II	Var pos rnd III
Newton-Raphson 8, full tree	1.769 *CPI*	1.667 *CPI*	1.466 *CPI*
Newton-Raphson 16, full tree	1.723 *CPI*	1.620 *CPI*	1.419 *CPI*
Newton-Raphson 28, full tree	1.676 *CPI*	1.574 *CPI*	1.3722 *CPI*
Newton-Raphson 8, half tree	1.901 *CPI*	1.817 *CPI*	1.613 *CPI*
Newton-Raphson 16, half tree	1.830 *CPI*	1.746 *CPI*	1.542 *CPI*
Newton-Raphson 28, half tree	1.760 *CPI*	1.676 *CPI*	1.472 *CPI*

Table 5.3: Performance (cycles per instruction in runtime simulations on traces of the SPEC92fp benchmarks) of the different FP units integrated into a pipelined RISC processor.

Version	Gen rnd I	Grad rnd II	Var pos rnd III
Newton-Raphson 8, full tree	134641	142988	161274
Newton-Raphson 16, full tree	136796	145143	163429
Newton-Raphson 28, full tree	266769	275116	293402
Newton-Raphson 8, half tree	114479	121334	141112
Newton-Raphson 16, half tree	116634	123489	143267
Newton-Raphson 28, half tree	246607	253462	273240

Table 5.4: Cost (gate count) of the different FP units integrated into a pipelined RISC processor.

are depicted. In each figure, the result of a FPU version, that uses a full-sized addertree, is connected with the corresponding FPU version, that uses a half-sized adder-tree, by a line, where the full-sized version is always faster and more expensive than the half-sized version. The maximum difference between two connected FPU results is 0.1 CPI and about 20 kG. In this way, the choice of the multiplier options has only a small effect on the performance and small effect on the cost. Comparing all different FPU versions within a particular rounding architecture, the situation is similar in the different figures. The maximum difference of the CPI is 0.24, so that a moderate speed-up can be achieved by using a fast divider and multiplication implementation. But the use of a fast divider increases the cost to a large extent by up to 132 kG, so that a fast divider version might be too expensive. Comparing the different rounding architectures, the best performance with relatively small additional cost is provided by the variable position rounding FPUs using rounding architecture III. In this way, the choice of the rounding architecture has the largest impact on the design quality, differing by about 0.3 CPI among the different architectures, but only by about 26 kG in cost.

In general the use of rounding architecture III, that uses dedicated rounding implementations for each functional unit seems to be the best choice in IEEE compliant FP implementations.

Bibliography

[1] Al-Twaijry, H. *Area and Performance Optimized CMOS Multipliers.* PhD thesis, Stanford University, August 1997.

[2] Anderson, S.W. and Earle, J.G. and Goldschmidt, R.E. and Powers, D.M. The IBM system 360 model 91: Floating-point unit. *IBM J. Res. Dev.*, 11:34–53, January 1967.

[3] Bewick, G.W. *Fast Multiplication: Algorithms and Implementation.* PhD thesis, Stanford University, March 1994.

[4] Cortadella, J. and Llaberia, J.M *Evaluation of $A + B = K$ Conditions without Carry Propagation* IEEE Trans. on Computers, vol. 41, pp. 1484-1488, November, 1992.

[5] Das Sarma, D. and Matula, D. W. *Measuring the Accuracy of ROM Reciprocal Tables.* IEEE Trans. on Computers, vol. 43, pp. 932-940, August, 1994.

[6] Das Sarma, D. and Matula, D. Faithful Bipartite ROM Reciprocal Tables, In *Proceedings of the 12th Symposium on Computer Arithmetic*, vol. 12, pp.17-28, IEEE, 1995.

[7] Das Sarma, D. and Matula, D. Faithful Interpolation in Reciprocal Tables, In *Proceedings of the 13th Symposium on Computer Arithmetic*, vol. 13, pp. 82-91, IEEE, 1997.

[8] Daumas, M. and Matula, D.W. Recoders for partial compression and rounding. Technical Report 97-01, Laboratoire de l'Informatique du Paralllisme, Lyon, France, 1997.

[9] Even, G. and Mueller, S.M. and Seidel, S.M. A Dual Mode IEEE multiplier. In *Proceedings of the 2nd IEEE International Conference on Innovative Systems in Silicon*, pages 282–289. IEEE, 1997.

[10] Even, G. and Paul, W.J. On the design of IEEE compliant floating point units. In *Proceedings of the 13th Symposium on Computer Arithmetic*, volume 13, pages 54–63. IEEE, 1997.

[11] Even, G. and Seidel, P.M. A comparison of three rounding algorithms for IEEE floating-point multiplication. In *Proceedings of the 14th IEEE Symposium on Computer Arithmetic*, pages 225-232, April 1999.

[12] Even, G. and Seidel, P.M. A comparison of three rounding algorithms for IEEE floating-point multiplication. to be published in Special Issue on Computer Arithmetic, IEEE Trans. on Computers, July 2000.

[13] Fowler, D.W. and Smith, J.E. An accurate, high speed implementation of division by reciprocal approximation. In *Proceedings of the 9th Symposium on Computer Arithmetic*, volume 9, pages 60–67. IEEE, September 1989.

[14] Farmwald, M. P. On the design of high performance digital arithmetic units, PhD thesis, Stanford Univ., August, 1981.

[15] Ferrari, D. A division method using a parallel multiplier, IEEE Trans. Electr. Comput., vol. EC-16, pp. 224-226, 1967.

[16] Hennessy, J.L. and Patterson, D.A. *Computer Architecture: A Quantitative Approach.* Morgan Kaufmann Publishers, INC., San Mateo, CA, 2nd edition, 1996.

[17] Hill, M. SPEC92 Traces for MIPS R2000/3000. University of Winconsin, ftp://ftp.cs.newcastle.edu.au/pub/r3000-traces/din', 1995.

[18] Ito, M. and Takagi, N. and Yajima, S. Efficient Initial Approximation and Fast Converging Methods for Division and Square Root", In *Proceedings of the 12th Symposium on Computer Arithmetic*, vol. 12, pp. 2-9, IEEE, 1995.

[19] IEEE standard for binary floating-point arithmetic. ANSI/IEEE754-1985, New York, 1985.

[20] Kane, G. and Heinrich, J. *MIPS RISC Architecture.* Prentice Hall, 1992.

[21] Lee, C. Multistep gradual rounding. *IEEE Transactions on Computers*, 32(4):595–600, April 1989.

[22] Müller, S.M. and Paul,W.J. *The Complexity of Simple Computer Architectures.* Lecture Notes in Computer Science 995. Springer, 1995.

[23] Müller,S.M. and Paul,W.J. *The Complexity of Correctness of Computer Architectures.* Springer, 2000, Draft.

[24] Nakata, C. and Brock,J. *H4C Series: Design Reference Guide. CAD, 0.7 Micron* L_{eff}. Motorola Ltd., 1993. Preliminary.

[25] Nielsen, A.M. and Matula, D.W. and Lyu, C.N. and Even, G. Pipelined packet-forwarding floating point: II. an adder. In *Proceedings 13th Symposium on Computer Arithmetic*, pages 148–155, Asilomar, California, July 6-9 1997.

[26] Oberman, S.F. *Design Issues in High Performance Floating Point Arithmetic Units.* PhD thesis, Stanford University, January 1997.

[27] Oberman, S.F. and Al-Twaijry, H. and Flynn, M.J. The SNAP project: Design of floating point arithmetic units. In *Proceedings of the 13th Symposium on Computer Arithmetic*, volume 13, pages 156–165. IEEE, 1997.

[28] Oberman, S.F. and Flynn,M.J. Fast IEEE rounding for division by functional iteration. Technical Report CSL-TR-96-700, Stanford University, July 1996.

[29] Intel Corporation Pentium Processor Family Developer's Manual Volume 1: Pentium Processors, 1995.

[30] Paul,W.J. and Seidel,P.M. The complexity of Booth recoding. In *Proceedings of the 3rd conference on Real Numbers and Computers RNC3*, pages 199-218, Paris, France, April 1998.

[31] Quach, N. and Flynn, M. Design and implementation of the SNAP floating-point adder. Technical Report CSL-TR-91-501, Stanford University, December 1991.

[32] Quach, N. and Flynn, M.J. An improved algorithm for high-speed floating-point addition. Technical Report CSL-TR-90-442, Stanford University, August 1990.

[33] Quach, N. and Takagi, N. and Flynn, M. On fast IEEE rounding. Technical Report CSL-TR-91-459, Stanford University, January 1991.

[34] Santoro, M.R. and Bewick, G. and Horowitz, M.A. Rounding algorithms for IEEE multipliers. In *Proceedings 9th Symposium on Computer Arithmetic*, pages 176–183, 1989.

[35] Schulte, M.J. and Omar, J. and Swartzlander, E.E., Optimal Initial Approximations for the Newton-Raphson Division Algorithm", Computing, vol. 53, pp. 233-242, August, 1994.

[36] Seidel, P.-M. High-speed redundant reciprocal approximation. In *Proceedings of the 3rd conference on Real Numbers and Computers RNC3*, pages 219-229, Paris, France, April 1998.

[37] Seidel, P.-M. How to half the latency of IEEE compliant floating-point multiplication. In *Proceedings of the 24th Euromicro Conference*, volume 24, pages 329-332. IEEE, 1998.

[38] Seidel, P.-M. *On the architecture of IEEE compliant floating-point units.* to appear in, *Proceedings of the IASTED Conference of Applied Informatics 2000*, February 2000.

[39] Seidel, P.-M. *High-speed redundant reciprocal approximation.* INTEGRATION, the VLSI journal 28 (1999), pp. 1-12.

[40] Seidel, P.-M. and Even, G. How many logic levels does floating-point addition require? In *Proceedings of the IEEE International Conference on Circuit Design (ICCD98)*, pages 142-149, October 1998.

[41] Siemens München. *VENUS-S Semi-Custom Design System: Zellkatalog*, 1988.

[42] Wong, D. and Flynn, M. Fast Division Using Accurate Quotient Approximations to Reduce the Number of Iterations, IEEE Trans. on Computers, vol. 41, pp. 981-995, August, 1992.

[43] Soderquist, P. and Leeser, M.. Floating-point division and square root: Choosing the right implementation. Technical Report EE-CEG-95-3, Cornell University, April 1995.

[44] Yu, R.K. and Zyner, G.B. 167 MHz Radix-4 floating point multiplier. *Proceedings 12th Symposium on Computer Arithmetic*, 12:149–154, 1995.

[45] Zyner, G. Circuitry for rounding in a floating point multiplier. U.S. patent 5150319, 1992.