A TABLE LOOKUP-LESS METHOD FOR CORRECTLY ROUNDED IEEE-754 ELEMENTARY FUNCTIONS

By

George Kamal Kamel Badawi

A Thesis Submitted to the
Faculty of Engineeringat Cairo University
in Partial Fulfillment of the
Requirements for the Degree of
MASTER OF SCIENCE
in
Electronics and Communications Engineering

FACULTY OF ENGINEERING, CAIRO UNIVERSITY
GIZA, EGYPT
2015

A TABLE LOOKUP-LESS METHOD FOR CORRECTLY ROUNDED IEEE-754 ELEMENTARY FUNCTIONS

By

George Kamal Kamel Badawi

A Thesis Submitted to the
Faculty of Engineeringat Cairo University
in Partial Fulfillment of the
Requirements for the Degree of
MASTER OF SCIENCE
in
Electronics and Communications Engineering

Under the Supervision of

Prof. Dr. Hossam A. H. Fahmy
Professor of Electronics and Communications
Electronics and Communications Engineering Department
Faculty of Engineering, Cairo University

FACULTY OF ENGINEERING, CAIRO UNIVERSITY
GIZA, EGYPT
2015

A TABLE LOOKUP-LESS METHOD FOR CORRECTLY ROUNDED IEEE-754 ELEMENTARY FUNCTIONS

By

George Kamal Kamel Badawi

A Thesis Submitted to the
Faculty of Engineeringat Cairo University
in Partial Fulfillment of the
Requirements for the Degree of
MASTER OF SCIENCE
in
Electronics and Communications Engineering

Approved by the Examining Committee:

Prof. Dr. Hossam A. H. Fahmy, Thesis Main Advisor

Associate Prof. Ibrahim Mohamed Qamar, Internal Examiner

Prof. Dr. Mohamed Watheq Ali Kamel El-Kharashi , External Examiner
Professor of Computer and Systems at Faculty of Engineering, Ain Shams University

FACULTY OF ENGINEERING, CAIRO UNIVERSITY
GIZA, EGYPT
2015

Acknowledgments

Firstly, I would also like to express my gratitude to my advisor Dr. Hossam for his patient guidance and unending encouragement during the three years of my MSc program. I would also like to thank all researchers of ENS de Lyon who contributed over the past two decades in order to make the world of computing elementary functions accurate and efficient; without their fundamental works, this thesis would not have been possible. Last, but not least, I would like to thank my family, to whom I dedicate this thesis, for their unconditional care and support throughout my life.

Table of Contents

List of Tables

List of Figures

List of Symbols and Abbreviations

CPA	Carry-Propagate Adder
CSA	Carry-Save Adder
DP	Double-Precision
FMA	Fused Multiply-Add
FPN	Floating-Point Number
GFLOPS	Giga FLoating-point Operations Per Second
GPU	Graphics Processing Unit
QP	Quadruple-Precision
RD	Round to $-\infty$
RN	Round to Nearest Even
RU	Round to $+\infty$
RZ	Round to Zero
SIMD	Single Instruction Multiple Data
SP	Single-Precision
SSE	Streaming SIMD Extensions
TMD	Table Maker's Dilemma
ULP	Unit in the Last Place

Abstract

This thesis presents a hardware-oriented method for computing correctly rounded IEEE-754 single-precision floating-point elementary functions. The method presents a *table "lookup-less"* approach using a minimax approximation polynomial of a high degree within a single wide range as opposed to the currently dominant table-based designs that have to use large look-up tables while maintaining low-degree piecewise polynomials to achieve the accuracy required for correct rounding. The method is firstly applied to trigonometric functions for argument values $|x| < 2^{53}$. Range reduction and polynomial evaluation steps are carried out using a standard double-precision fused-multiply add (FMA) with the help of a simple control unit and combinational logic to support the range reduction, the implicit reconstruction and the final rounding steps. The method uses a single low-cost range reduction algorithm for all arguments without penalizing small and intermediate argument values. This approach makes computing correctly rounded trigonometric functions for a very large range of arguments possible entirely at the hardware level with reasonable logic resources and low memory requirements that can be less than 700 bits. This memory is 3.5 to 5 times smaller than the state-of-the-art faithful quadratic minimax table-based designs that cover a reduced range. Although the thesis addresses the round-to-nearest mode, the method can be easily applied to the directed rounding modes. Moreover, the method is applied for 2^x and $\log_2(x)$ within reduced intervals. We also argue that the method can be extended to involve the double-precision case for trigonometric functions.

Chapter 1: Introduction

1.1 Overview and Motivation

High performance, accurate and IEEE-754 compliant elementary functions are required for many applications such as scientific computing, high-performance computing and general-purpose processors. Several accurate software libraries, such as MPFR [9] and CRlibm [5], are capable of computing correctly rounded results for standard and arbitrary precisions. However, hardware-based techniques still fall behind when it comes to producing correctly rounded results especially for high precisions such as IEEE-754 standard single-precision (SP). The situation becomes almost infeasible for double-precision (DP) and beyond precisions due to the enormous memory requirements using the currently dominant table-based methods.

The IEEE-754 standard requires correct rounding for the fundamental arithmetic operations such as addition, multiplication and FMA. Thankfully, correct rounding for these operations can be easily achieved by adding predetermined extra bits called guard bits that can be obtained using analytic proofs. Unfortunately, such proofs do not exist for elementary functions such as the trigonometric, exponential and logarithmic functions. The hardness of obtaining correctly rounded results for such functions is known as the *Table Maker's Dilemma* or TMD [20]. This term was coined by William Kahan [1]. He says:

> Why can't Y^W be rounded within half a unit in the last place like SQRT? Because nobody knows how much computation it would cost to resolve what I long ago christened "The Table-Maker's Dilemma".
>
> Is it time to quit yet? That's the Table-Maker's Dilemma. No general way exists to predict how many extra digits will have to be carried to compute a transcendental expression and round it correctly to some preassigned number of digits. Even the fact (if true) that a finite number of extra digits will ultimately suffice may be a deep theorem.

Thus, correct rounding for these functions, as well as many other functions, such as $1/\sqrt{x}$ and x^n, only became recommended in the last revision of the standard IEEE-754-2008.

Correct rounding increases the result accuracy to the last bit and leads up to more accurate computations for many algorithms. Furthermore, it preserves numerical portability and other mathematical properties such as monotonicity and symmetry [18]. Monotonicity can be simply described that for a function f, where $f(x+\text{ulp}(x)) > f(x)$, the computed function $\hat{f}$ should maintain that $\hat{f}(x+\text{ulp}(x)) \geq \hat{f}(x)$.

1.2 Goal of the Thesis

The main goal of this thesis is to present a hardware-oriented method for computing correctly rounded results for certain IEEE-754 SP elementary functions. The method is

[1] He is known as the father of floating-point. Kahan received the Turing Award for his fundamental contributions to numerical analysis and floating-point computations.

dedicated for computing correctly rounded $\sin(x)$ and $\cos(x)$ for argument values $|x| < 2^{53}$. Moreover, the method was applied for computing 2^x for $x \in [0,1]$ and $\log_2(x)$ for $x \in [2,4]$.

The basic method uses a single minimax approximation polynomial of a large degree valid over a single relatively wide interval. Thus, the method basically avoids using look-up tables that are used in table-based methods, as all reduced arguments lie within a single domain. A standard DP fused multiply-add (FMA) is assumed to evaluate the approximation polynomial. For the case of trigonometric functions, the FMA additionally carries out the range reduction using the FMA with the help of a simple control unit as well as extra combinational logic to carry out the implicit reconstruction and final rounding.

Our method strictly seeks correct rounding for all SP arguments. Exhaustive tests show that, we have only one *faithful* case (see Section 1.4.3) , i.e. having an error of 1 unit in the last place (ulp), for $\sin(x)$ and two faithful cases for $\cos(x)$ for $|x| \leq 2^{53}$ using degree-16 polynomial with a total memory, including the range reduction memory requirements, of 708 bits using a simple heuristic algorithm to reduce the memory; otherwise, all arguments up to 2^{53} are correctly rounded for the round-to-nearest mode (RN). Similar accuracy results were obtained for 2^x and $\log_2(x)$ within the designated reduced intervals using polynomial degrees of 10 and 15 with consuming memory of 492 and 858 bits respectively. The design was modeled and exhaustively tested using a C program with the help of the MPFR library.

Our method has several advantages over table-based techniques for the correct rounding case:

- The method requires a very small amount of memory, usually hundreds of bits, which is several times smaller compared even to the state-of-the-art faithfully rounded designs that assume reduced arguments.

- As opposed to table-based designs which use specialized hardware such as ad-hoc powering units and multi-operand adders, our method relies on a general-purpose component, a standard DP FMA, which has been adopted by the major processor manufacturers. Thus, with simple additional logic, the FMA can be adapted to apply our method while being used for other purposes.

- Although we mainly address RN mode throughout this thesis for the sake of priority and simplicity, the method can be easily applied for the directed rounding modes.

- The FMA can be shared to evaluate many elementary functions at lower memory cost compared to table-based methods.

- We also argue that this method can be applied for the case of DP trigonometric functions while using quadruple-precision (QP) FMA.

However, the apparent disadvantage of our method is the requirement for a higher latency compared to table-based methods. To mitigate this problem, we propose a set of a few piecewise polynomials, as will be discussed later, to reduce the average and worst case latencies.

1.3 Thesis Layout

Chapter 1 discusses the motivation and the contribution of the thesis. A review for floating-point arithmetic and IEEE-754 is presented. Chapter 2 presents the table maker's dilemma and the strategies adopted to produce correctly rounded elementary transcendental functions. Chapter 3 discusses range reduction algorithms, approximation polynomials and the Remez algorithm used for computing minimax polynomials. Chapter 4 presents the table-based methods used in literature, their tradeoffs and limitations for the correct rounding case. Chapter 5 presents a table lookup-less method for computing correctly rounded IEEE-754 SP floating-point trigonometric functions. The method is also applied for 2^x and $\log_2(x)$ functions within reduced ranges. Chapter 6 discusses the conclusions and future work.

1.4 Floating-Point Representation

Floating-point is a system for representing real numbers, or more precisely, a subset of their approximation, on computers. Floating-point was meant to be an alternative for fixed-point systems to increase the dynamic range with using low storage and logic requirements. The applications of floating-point extend from scientific computing and simulation to graphics and digital signal processing.

The conventional non-redundant floating-point number (FPN) x is represented as defined in [2]:

$$x = (-1)^{S_x} \times M_x \times \beta^{E_x} \tag{1.1}$$

where $S_x = \{0, 1\}$ represents the sign, $M_x \geq 0$ is called the significand or mantissa, β is called the base or radix and E_x is called the exponent where E_x is an integer. FPNs represent a subset of real numbers depending on the storage reserved for the designated precision and the width tradeoff between mantissa and exponent sections.

Floating-point representation increases the dynamic range (the ratio between the largest and the smallest represented numbers) in comparison with the fixed-point representation. This advantage drastically mitigates the problem of the frequent occurrences of overflows and underflows ingrained within fixed-point representations.

Nevertheless, the wide range advantage does not come for free. Floating-point reserves some digits/bits for storing the exponent field. Furthermore, floating-point frequently suffers from loss of accuracy due to round-off errors (see Section 1.4.3).

In order to develop a unique representation for each FPN and simplify the implementation, the FPN can be encoded to be normalized, i.e. the most significant digit becomes a non-zero number. Thus, for $\beta = 2$, the most significant bit always becomes 1 and therefore it is meaningless to store it. Subsequently, the most significant bit is called the hidden bit.

1.4.1 A Brief History of Floating-Point

The first known floating-point system was invented in 1914 by Leonardo Torres y Quevedo, a Spanish inventor, who designed an electro-mechanical Charles Babbage's machine with support of floating-point [20]. In late 1930s and early 1940s, Konrad Zuse, a German

Name	Base	Size in bits	Mantissa bits	Exponent bits	Decimal digits
PDP-11	2	32	23+hidden bit	8	7.2
CDC 6000	2	60	48+hidden bit	11	14.7
IBM 360	16	32	24	7	7.2
Zuse Z3	2	24	16	7	4.8

Table 1.1: Examples of memorable floating-point representations

engineer and inventor, built a series of complex mechanical computers (Z1, Z2, Z3 and Z4) that used binary 22-bit floating-point systems.

Since the late 1940s until 1980s, many floating-point systems were developed. Major manufacturers of the time, such as IBM, DEC and CDC, used different word-lengths, different tradeoffs of exponent-mantissa lengths and different radices (2,8,10 and 16). Table (1.1) shows some old memorable systems. Such differences made it more difficult to maintain and extend programs as every manufacturer used its own rules which led to different accuracy results. Thus, a standardization for floating-point arithmetic became a necessity. The standard for floating-point arithmetic is called IEEE-754 and was established in 1985. The current revision of the standard was published in 2008.

1.4.2 Unit in the Last Place

Frequently, it becomes more convenient and expressive to represent the error relatively rather than offering absolute values. That is usually needed to compute error bounds for results where the absolute value of the error, in many cases, is meaningless. The unit in the last place function is defined simply as the gap between two consecutive floating-point representations. That is for argument x, the ulp function is calculated as:

$$\text{ulp}(x) = \beta^{E_x - p + 1} \tag{1.2}$$

1.4.3 Rounding

While the representation of the operands and outputs must be encoded within a pre-defined unified precision, the output of some operation may initially require a larger precision than the target precision. In order to follow the canonical representation, the output must be rounded to the target precision. The result is rounded according to some rounding mode. IEEE-754 defines four rounding modes for binary floating-point systems. As shown in Table 1.2, The first rounding mode is called rounding-to-nearest mode. The other three rounding modes are called directed rounding modes.

The result is said to be exactly or correctly rounded if the final rounded result equals the infinitely precise result rounded to the target precision. This is somewhat straightforward for the fundamental arithmetic operations such as addition, subtraction, multiplication, division and square root. Accordingly, correct rounding for these operations is mandatory in IEEE-754.

Rounding Type	Comment
Round to nearest, ties to even (RN)	Round to the nearest floating-point representation; if the result falls exactly at the midpoint between the two consecutive floating-point numbers, it is rounded to the even number.
Round toward 0 (RZ)	Directly round towards zero, i.e. truncate the non-redundant result.
Round toward $+\infty$ (RU)	Directly round up to $+\infty$.
Round toward $-\infty$ (RD)	Directly round down to $-\infty$.

Table 1.2: IEEE 754 rounding modes

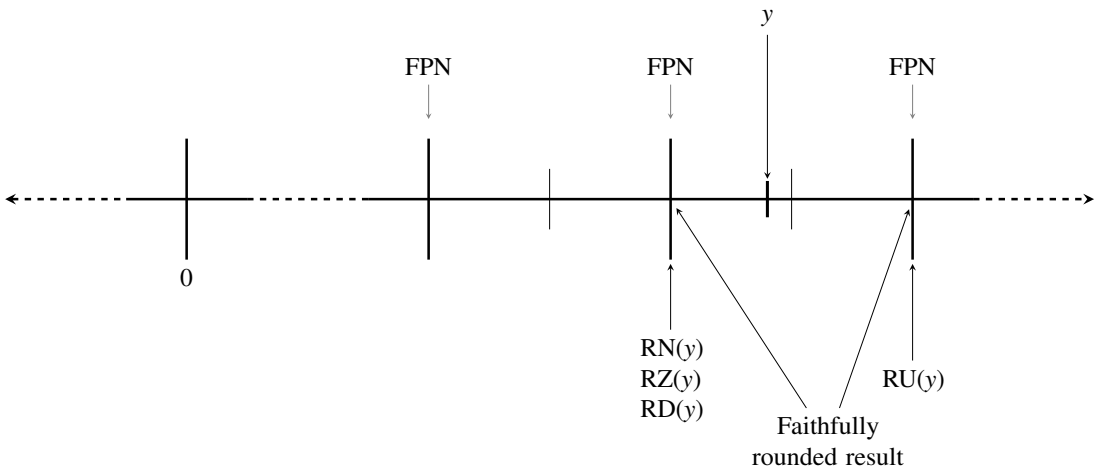


Figure 1.1: Rounding modes

The maximum error due to rounding can be deduced from Figure 1.1. Let y be the pre-rounded result and R be the designated rounding mode to the target precision. Thus, the following condition must be fulfilled:

$$|\mathrm{R}(y) - y| \leq \begin{cases} \frac{1}{2}\mathrm{ulp}(y) & \text{rounding-to-nearest mode} \\ \mathrm{ulp}(y) & \text{directed rounding modes} \end{cases} \tag{1.3}$$

For transcendental functions such as trigonometric and exponential functions, correct rounding is much more complex to obtain and usually requires much higher memory and logic requirements.That is because the pre-rounded result is inevitably obtained within some error (as will be discussed in Chapter 2). These requirements usually grow harder, and may become infeasible for hardware algorithms, as the target precision increases. Many hardware-based implementations may return faithfully rounded results, i.e. the obtained result is one of two consecutive floating-point representations surrounding the exact result (see Figure 1.1). Therefore, the maximum error due to faithful rounding is 1 ulp. Note that faithful rounding is not really a canonical rounding mode such as RN and RZ.

1.5 IEEE 754 Standard

IEEE 754 is the standard that governs binary and decimal floating-point arithmetic. It includes a set of regulations to enhance numerical portability and efficiency. IEEE 754 defines normalized binary FPN as follows:

$$x = (-1)^{S_x} \times 1.M_x \times 2^{E_x} \tag{1.4}$$

where S_x is the sign bit, E_x is the biased exponent (i.e. the exponent equals E_x- bias) and $M_x = m_1 m_2 \ldots m_{p-1}$ is the unsigned stored fraction. Thus the smallest and largest FPN magnitudes are $2^{E_{\min}}$ and $\left(2 - 2^{-(p-1)}\right) \times 2^{E_{\max}}$ respectively.

1.5.1 Formats

The standard defines three binary formats: binary32 (SP), binary64 (DP) and binary128 (QP). Table 1.3 shows the number of reserved bits for mantissa $(p-1)$ and exponent w_e, the bias of the exponent, the minimum and maximum exponents for each format (see also Figure 1.2). Furthermore, the standard allows for using extended precisions for each of the above basic formats. Intel is known to have been using the 80-bit extended DP format (1 bit for sign, 15 bits for exponent and 64 bits for mantissa) for its x87 series.

1.5.2 Rounding Modes

In addition to the four rounding modes defined in Table 1.2, another rounding to nearest was added to the standard in the 2008 revision, called round to nearest, ties away from zero, and is used for decimal formats.

Precision	$p-1$	w_e	bias = $2^{w_e-1}-1$	$E_{\min}$	$E_{\max}$
SP	23	8	127	-126	127
DP	52	11	1023	-1022	1023
QP	112	15	16383	-16382	16383

Table 1.3: IEEE 754 binary formats

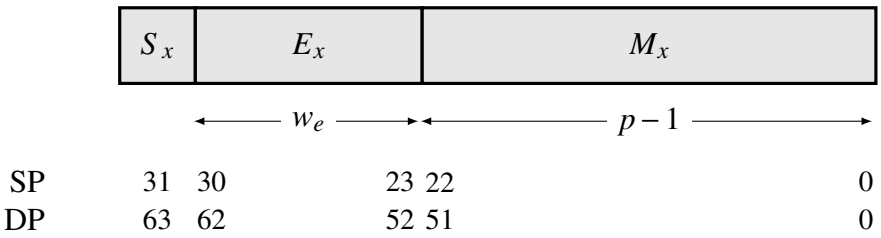


Figure 1.2: IEEE 754 SP and DP binary formats

1.5.3 Operations

IEEE 754 defines four types of operations as follows:

1. General-computational operations such as the addition, subtraction, multiplication, division, square root, fused multiply–add and conversion operations.
2. Quiet-computational operations such as the copy and negate operations.
3. Signaling-computational operations such as the comparison operations.
4. Non-computational operations such as the isNaN, isFinite ans isZero operations.

1.5.4 Subnormals

Numbers below the smallest normalized FPN can be still represented with potential loss of accuracy using the smallest exponent representation of the system, that is used also to represent zeroes. This is often called gradual underflow. Such numbers are called denormalized numbers or subnormals. Thus, the magnitudes of subnormals lie within $\left[2^{E_{\min}-p+1}, 2^{E_{\min}}\right)$.

1.5.5 Exceptions

1. Invalid: occurs for operations such as $0/0$ and $0\times\infty$ and the square root of negative non-zero inputs. The default output is qNaN.
2. Division by zero: occurs when the result is infinite whereas the operation is applied to finite operands (e.g. $1/0$ and $\log_2(+0)$). The default output is ∞ while the resulting sign depends on the operation and the sign of the operands.

3. Overflow: occurs when the magnitude of the result is larger than $\left(2-2^{1-p}\right)\times 2^{E_{\max}}$. The default output is infinity with the same sign of the pre-rounded result in case of rounding to nearest.

4. Underflow: occurs when the magnitude of the result is less than $2^{E_{\min}}$. The default output is the resulting subnormal FPN.

5. Inexact: occurs when the result cannot be exactly represented and is rounded to the target precision.

Chapter 2: Correct Rounding and The Table Maker's Dilemma

2.1 Introduction

A number is said to be algebraic if it is a root of a polynomial having integer coefficients. Thus, a number is said to be transcendental if it is not algebraic. A function f is said to be algebraic if it satisfies the following condition:

$$P(x, f(x)) = 0 \tag{2.1}$$

where P is a polynomial having integer coefficients. A function is said to be transcendental if it is not algebraic [20].

Ferdinand von Lindemann, a German mathematician, proved in 1882 that e^x is transcendental if x is non-zero algebraic number as a corollary to a theorem that proves that π is transcendental [21, 1]. This result is applicable for other elementary functions such as trigonometric and logarithmic functions.

As previously mentioned in Chapter 1, correct rounding can be easily obtained for the fundamental operations, such as addition and multiplication, by adding some guard bits that can be determined analytically. That is, some extra bits for the pre-rounded result that guarantee correct rounding. Algebraic functions, such as reciprocal and square root, have some kind of analytic proofs [11, 13] that determine the upper bound of these extra bits for correct rounding. Unfortunately, no proofs exist for the case of transcendental functions.

Thus, trigonometric, exponential and logarithmic functions cannot be exactly represented with a finite number of arithmetic operations and always produce, except for rare known exceptions such as cos(0) and exp(0), transcendental number outputs whenever the argument is algebraic which is the case for floating-point representations [16]. Therefore, transcendental functions must be approximated to be computed.

2.2 The TMD Problem

Let us assume an IEEE-754 normalized floating-point argument x represented within a target precision p as represented in Equation 1.4. If we seek to compute $f_p(x)$, the correctly rounded floating-point representation of the transcendental function $f(x)$ within the target precision p, $f(x)$ is approximated using a polynomial $P(x)$ within a closed interval $C = [a, b]$. This approximation, which is of minimax type for our case (see Section 3.2), induces an error called the *approximation* error ϵ_{approx}:

$$\epsilon_{\text{approx}} = \max_{x \in C} |f(x) - P(x)| < 2^{-m} \tag{2.2}$$

where m is a positive integer and $m > p$. Thus, $P(x)$ is obtained within an interval I of width of 2×2^{-m} centered around $f(x)$. To obtain the correctly rounded value $f_p(x)$, the following relation should be maintained:

$$f_p(x) = \mathrm{R}_p(P(x)) \tag{2.3}$$

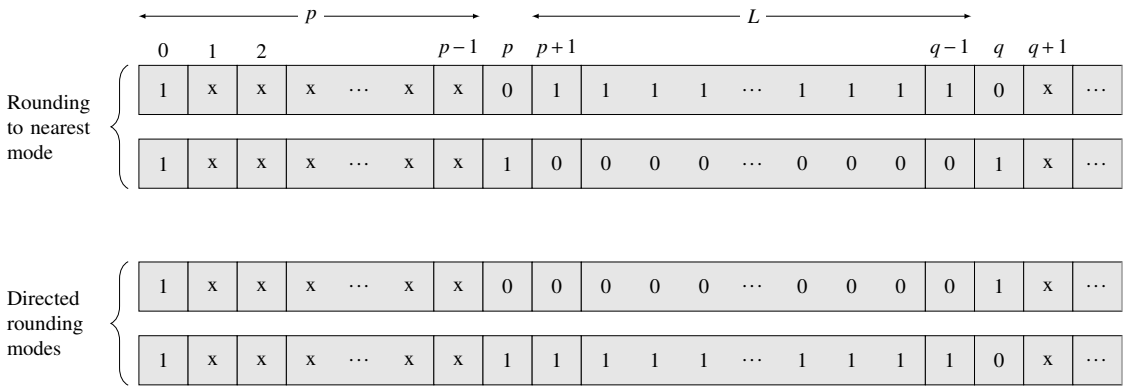


Figure 2.1: The cases for closeness of $f(x)$ to the breakpoint for the different rounding types

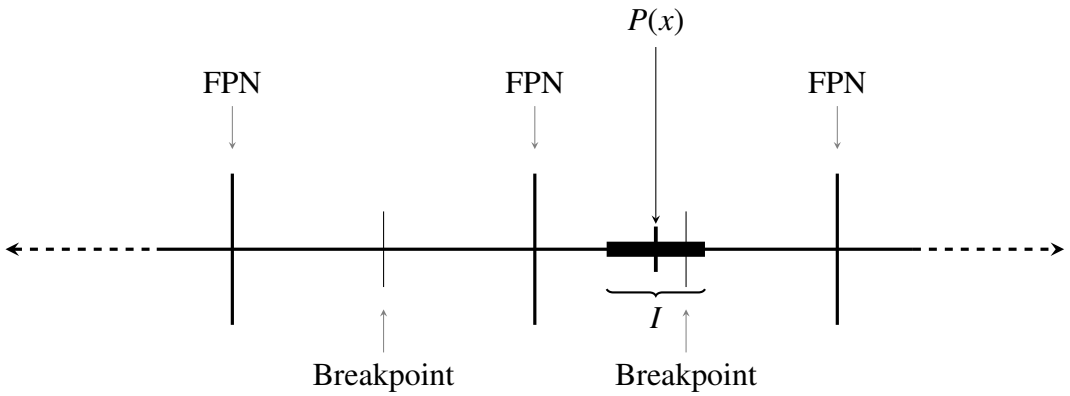


Figure 2.2: An example where TMD occurs

where $\mathrm{R} \in \{\mathrm{RN}, \mathrm{RZ}, \mathrm{RU}, \mathrm{RD}\}$ is the rounding operator to the precision p, $x \in C$ and $f_p(x) = \mathrm{R}_p(f(x))$.

However, since the known value is $P(x)$ not $f(x)$, we may inversely imagine that I is now centered around $P(x)$ and $f(x) \in I$. If m is not sufficiently large, the interval I may be wide enough to contain the breakpoint, which is defined as the mid-point between two consecutive floating-point representations for the case of RN and the floating-point representation itself for the other three rounding modes, and therefore Equation 2.3 may not be maintained and subsequently the TMD occurs.

Let us define the infinitely precise unsigned mantissa of the transcendental function $f(x)$ as $\sum_{i=0}^{\infty} m_i 2^{-i}$ with $m_0 = 1$. Figure 2.1 shows the cases of closeness of $f(x)$ to the breakpoint in the case of the nearest rounding mode (i.e. RN) and the directed rounding modes (i.e. RZ, RU and RD). Assume that L is length of the chain with consecutive ones or zeroes starting from the position $p+1$ and ending at $q-1$ with having $q \geq p+2$. Therefore, the higher the value of q, the higher the value of L, the closer the value of $f(x)$ to the breakpoint and subsequently the harder the case to be correctly rounded.

Let us discuss the case of RN. Having the pre-rounded value $P(x)$ computed, we only guarantee that $f(x) \in I$ where $I = [P(x) - 2^{-m}, P(x) + 2^{-m}]$. Figure 2.2 shows the case when m is not large enough to guarantee correct rounding. If $f(x)$ actually lies on the right of the breakpoint, we obtain an incorrectly rounded result. Thus, the Equation 2.3

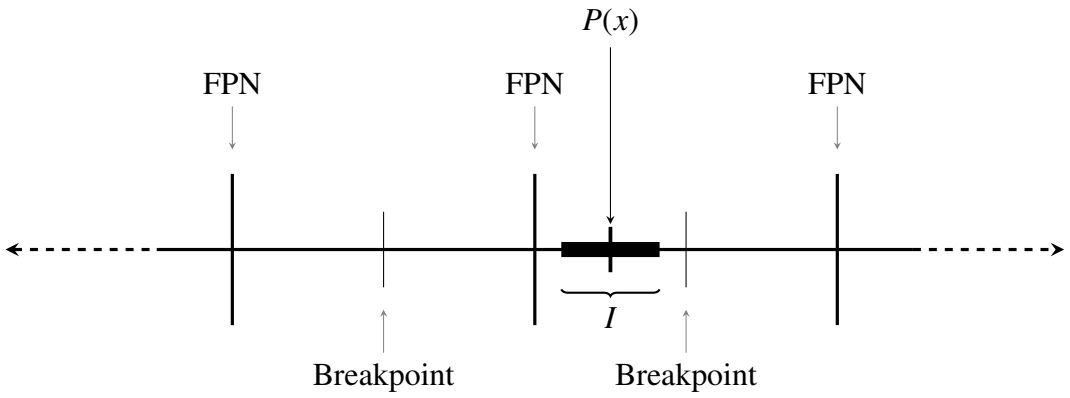


Figure 2.3: An example where TMD does not occur

is not guaranteed to be always achieved and subsequently the TMD occurs. Now, let us discuss the second case shown in Figure 2.3; no matter what the value of $f(x)$ is, Equation 2.3 is always maintained and we obtain a correctly rounded result. Clearly, the sufficient condition for the TMD to be solved is when the case shown in Figure 2.3 is achieved for every x within the designated interval. The real problem is how to determine and produce the required accuracy using feasible memory and logic cost to achieve such condition.

It should be noted that ϵ_{approx} is not the sole source of errors, another kind of errors called the *round-off* error stems from the fact that $P(x)$ is evaluated within finite precision. We call that precision, which may be much higher than the target precision p, the working precision w. Round-off errors may also involve the rounding of $P(x)$ coefficients in order to be stored within reasonable width.

2.3 Solving The TMD

Two intuitive methodologies for solving the TMD problem have been adopted in the literature. However, in real world implementations such as the CRlibm library (see Section 4.3), the two methodologies may be combined to improve the efficiency of the algorithms used.

2.3.1 Ziv's Multilevel Strategy

Ziv's multilevel strategy [36] is a progressive paradigm for producing a correctly rounded result. The method can be outlined as follows:

1. Set the index $i = 1$.
2. Evaluate the output $P^i(x)$.
3. Test if $P^i(x)$ is accurate enough such that Equation 2.3 is maintained. If so, return $\mathrm{R}_p\left(P^i(x)\right)$ and terminate the loop. If not, increment i and go to step 2.

Note that the accuracy of $P^i(x)$ increases as i gets incremented. Thus, Ziv's multilevel strategy aims at obtaining correct rounding for all arguments without penalizing the easy-to-round cases and subsequently producing a competent average execution latency.

Let us consider $\overline{T}$ as the average CPU time, T_i the CPU time for the iteration i, and p_i, the probability of x to be close enough to the breakpoint that it gets iterated until it reaches the i^{th} iteration. Therefore, $\overline{T}$ is calculated as follows:

$$\begin{aligned}\overline{T} &= \sum_{j=1}^{\infty}\left(1-\sum_{i=1}^{i=j-1} p_i\right)T_j \\ &= T_1 + (1-p_1)T_2 + (1-p_1-p_2)T_3 + \cdots\end{aligned} \tag{2.4}$$

Depending on the design and architecture decisions for each iteration which depend on the accuracy of $P^i(x)$, one can tune the values of p_i and T_i to reduce the value of $\overline{T}$. If $P^1(x)$ is designed such that p_1 is close to 1, then $\overline{T} \approx T_1$. Of course, as i increases, the cost of $P^i(x)$ implementation may greatly increase. Thus, we should make sure that almost all cases should satisfied within the first few iterations. Thus, even if i can grow to ∞ in theory, it can be dealt with some sort of exception whenever i becomes very large. However, this case remains very unlikely.

This methodology is somewhat inconvenient for hardware implementation as it may require multi-precision computations as well as high memory requirements for storing the minimax polynomial coefficients at all possible levels. Ziv's methodology was used in libultim library (now integrated in glibc [20]). A two-step evaluation technique is found in the CRlibm library (see Section 4.3).

2.3.2 Search Algorithms

The second methodology is to determine the worst cases through exhaustive search within the designated range a priori in order to estimate reasonable computing and memory requirements for design implementation. Exhaustive search is feasible for low and intermediate precisions (e.g SP or lower precisions). As for the case of SP (i.e. $p = 24$), the main target precision throughout the thesis, exhaustive testing can be achieved within less than an hour using a modern x86 processor for many elementary functions.

It should be noted that some intervals of floating-point representations do not require polynomial approximation to be evaluated and the correctly rounded output can be directly returned (e.g. very small arguments for sine, cosine and exponential functions). Search algorithms or numerical analysis can be used to determine such intervals. For example, we can, for the RN case, safely return $\sin(x) = x$ and $\cos(x) = 1$ when $|x| < 2^{-12}$ for SP and the same outputs when $|x| < 2^{-27}$ for DP.

Let us apply exhaustive search to find the hardest-to-round cases for some SP elementary functions within certain reduced intervals. Let L° be L with $\circ$ denoting the designated function. Assume that $\Gamma(L^\circ)$ is the number of occurrences of L° for the rounding mode R. Table 2.1 shows the relation between L° and $\Gamma(L^\circ)$ for $\circ \in \{\sin(x), \cos(x), \log_2(x), 2^x\}$ within certain intervals for the case of RN. One can simply note that Γ decreases exponentially as L° increases (Γ is roughly reduced by a factor of 2 when L° gets incremented). Assume that L°_* is the largest value of L° within the designated interval. Table 2.2 shows L°_* for $\circ \in \{\sin(x), \cos(x)\}$ with $[x \in 2^{-12}, 2^1]$.

One may then estimate that

$$\Gamma(L^\circ) \approx \lambda \times 2^{-(L^\circ+1)}$$

L°	$\Gamma\left(L^{\sin(x)}\right)$ $x \in [2^{-12}, 2^1]$	$\Gamma\left(L^{\cos(x)}\right)$ $x \in [2^{-12}, 2^1]$	$\Gamma\left(L^{\log_2(x)}\right)$ $x \in [2, 4]$	$\Gamma\left(L^{2^x}\right)$ $x \in [2^{-24}, 2^0]$
0	52863476	54411868	4191293	101863957
1	27318116	27275274	2099186	52732275
2	14454673	13669615	1048428	27562386
3	7207238	6842937	525044	13765947
4	3605298	3425081	262576	6895984
5	1801277	1713360	131080	3447106
6	901087	857143	65442	1723555
7	450415	428813	32970	861739
8	225280	213763	16482	431008
9	112412	107050	8177	215832
10	56309	53568	4015	107667
11	28090	26683	1969	53944
12	14090	13409	959	26882
13	7048	6698	487	13517
14	3492	3376	228	6629
15	1807	1614	126	3377
16	871	823	78	1697
17	473	420	41	846
18	230	195	16	442
19	113	109	6	207
20	57	51	1	103
21	29	25	0	55
22	11	15	3	19
23	5	4	0	14
24	4	5	0	6
25	1	2	0	2
26	**2**	1	**1**	2
27	0	1	0	1
28	0	**1**	0	0
29	0	0	0	**1**

Table 2.1: $\Gamma(L^\circ)$ vs L° for $\circ \in \{\sin(x), \cos(x), \log_2(x), 2^x\}$ within certain intervals

f	x	$f(x)$
sin	$1.10011110101010110010111\times2^{-4}$	$1.10011101111101011111000\underbrace{01\ldots26\,\text{zeroes}\ldots000}10\ldots\times2^{-4}$
sin	$1.11100111000001100001111\times2^{-2}$	$1.11010100110111101000101\underbrace{0111\ldots26\,\text{ones}\ldots111}01\ldots\times2^{-2}$
cos	$1.00100000111111111100110\times2^{-7}$	$1.11111111111110101110011\underbrace{1000\ldots28\,\text{zeroes}\ldots000}11\ldots\times2^{-1}$

Table 2.2: The values of x and $f^{\circ}(x)$ having L_*° for $\circ\in\{\sin(x),\cos(x)\}$ and $x\in\left[2^{-12},2^{1}\right]$

f	Interval	Rounding Type	L_*°
$\sin(x)$	$\left[2^{-25},\pi\right)$	Nearest	59
		Directed	72
$\cos(x)$	$\left[2^{-17},2^{2}\right)$	Nearest	58
		Directed	58
$\tan(x)$	$\left[2^{-25},\pi/2\right)$	Nearest	78
		Directed	72
$\log_2(x)$	$\left[2^{-1},2^{1024}\right)$	Nearest	54
		Directed	55
2^x	$(-\infty,+\infty)$	Nearest	59
		Directed	59
$\ln(x)$	$\left[2^{-1074},2^{-1}\right)$	Nearest	60
		Directed	60

Table 2.3: DP worst-case results for some elementary functions within certain intervals

where λ is the total number of FPNs within the designated interval. Thus, one can expect a rough estimation that $L_*^{\circ}\approx\log_2(\lambda)-1$ when we substitute $\Gamma(L_*^{\circ})=1$. Assume that m_*° is the largest value of m° within the designated interval, thus one can expect that $m_*^{\circ}\approx p+\log_2(\lambda)$. This rough estimation serves only as a guide for the required accuracy of a certain function.

However, finding the worst cases using exhaustive testing for high precisions such as DP is almost infeasible and for QP is entirely impossible for today's computational power. Advanced algorithms such as Lefèvre [15] and SLZ [31] algorithms were developed to find the worst cases for DP. Table 2.3 shows the results obtained by the algorithms of L_*° for some DP elementary functions within certain intervals [20].

2.4 Summary

A presentation of the TMD problem has been discussed. While a rigorous solution for the TMD problem does not exist, two intuitive methodologies were developed. The first methodology aims at gradually obtaining correct rounding and it is known as Ziv's multilevel strategy. The second methodology is to conduct search algorithms within intervals of interest in order to find the hardest-to-round cases and subsequently construct a reasonable implementation. While exhaustive search is feasible for small and intermediate precisions such as SP, more advanced algorithms such as Lefèvre's were developed to suit large precisions such as DP.

Chapter 3: Range Reduction and Polynomial Approximation

3.1 Range Reduction

In order to implement a realizable design, the working range within which the approximation is applied should be reasonably small. For example, we cannot use one polynomial to approximate the sine function over the entire range of SP FPNs. This will implicate incredibly huge memory and computing requirements. However, we can exploit the periodicity of the function and reduce the entire interval to much smaller interval (e.g. $[0, \pi]$).

Quite generally, an elementary function $f(x)$ is computed in three main steps:

1. *Range reduction*: The argument x may be transformed or reduced to another argument called the reduced argument x_r that lies within a smaller predetermined reduced interval.

2. *Evaluation*: $P(x_r)$ is evaluated using some approximation function valid within the reduced interval.

3. *Reconstruction*: The computed elementary function $\hat{f}(x)$ is obtained from $P(x_r)$.

The implementation of this abstract description is straightforward. For instance, the sine function can be computed as follows:

1. the argument is reduced to the interval $[0, \pi/4]$ using the relation $x_r = x - N\pi/4$.

2. Evaluate $\sin(x_r)$ and $\cos(x_r)$.

3. Obtain $\sin(x)$ from the relation $\sin(x) = \sin(x_r)\cos(N\pi/4) + \cos(x_r)\sin(N\pi/4)$. The values of $\sin(N\pi/4)$ and $\cos(N\pi/4)$ for $N = 0, 1, \ldots, 7$ are stored in a lookup table.

The choice of the reduced interval directly affects the evaluation and the reconstruction steps. For example, having determined the required accuracy, if we change the interval $[0, \pi/4]$ to $[0, \pi/8]$, we mitigate the requirements of the evaluation step (i.e. lower degrees for the case of polynomial or rational approximation or lower memory requirements for the case of table-based methods) yet at the expense of doubling the memory requirements for the reconstruction step as the maximum value of N now is 15 instead of 7. Subsequently, the range reduction influences the implementation decisions. Range reduction can be classified into two types as follows:

1. Additive reduction: where $x_r = x - NK$.

2. Multiplicative reduction: where $x_r = x/K^N$.

where N is a non-negative integer.

Multiplicative range reduction is pretty straightforward. K is usually chosen as the radix of the system; subsequently, x_r is computed with no loss in accuracy as only the exponent section is changed. For example, the computation of x_r for $\log_2(x)$ can be performed as $x_r = x/2^{E_x}$ and subsequently $\log_2(x) = E_x + \log_2(1.M_x)$.

However, accurate additive range reduction is surprisingly much more complex to realize. A straightforward additive range reduction can be done as follows:

$$N = \left\lfloor \frac{x}{K} \right\rfloor \tag{3.1}$$

$$x_r = x - N \times K \tag{3.2}$$

Thus, $x_r \in [0, K]$ or $[-K/2, K/2]$. Note that K, in many cases, is a transcendental number and subsequently NK cannot be exactly represented within floating-point systems. Therefore, we define K' as the FPN that represents K within the working precision w. A direct choice is to use $K' = \mathrm{RN}_w(K)$.

Unfortunately, the simple operation in Equation 3.2 may lead to inaccurate values of x_r and eventually even more inaccurate $\hat{f}(x)$ results. This loss of accuracy occurs when $x \approx NK'$ while there is insufficient working precision within the underlying architecture to obtain x_r correctly to enough least significant bits. This inaccuracy of x_r will be fed to the evaluation and then the reconstruction steps and the final output $\hat{f}(x)$ may be far from the correct $f(x)$. The loss of accuracy drastically worsens when x becomes very large as the massive cancellation grows inevitable. That is, when x and NK' are very large, the value of x_r is always expected to lie within the reduced interval, which is bounded to a much lower maximum value. This problem directly affects trigonometric functions. Fortunately, exponential functions, for example, overflow at much lower arguments and subsequently this problem is much more limited. Thus, more advanced range reduction algorithms, mainly targeted for trigonometric functions, were developed to produce accurate reduced arguments efficiently.

3.1.1 Cody and Waite's Redcution Algorithm

Cody and Waite's range reduction algorithm was introduced in [4] to increase the accuracy without increasing the working precision. The simple algorithm mitigates the massive cancellation problem by splitting K into two constants K'_1 and K'_2 as follows:

$$K \approx K'_1 + K'_2 \tag{3.3}$$

such that

$$\left| K - \left(K'_1 + K'_2 \right) \right| \ll \left| K - K' \right| \tag{3.4}$$

We can compute K'_1 and K'_2 as follows:

$$K'_1 = \mathrm{RN}_w(K) \tag{3.5}$$

$$K'_2 = \mathrm{RN}_w\left(K - K'_1 \right) \tag{3.6}$$

Hence, Equation 3.2 becomes as follows:

$$x_r = \left(x - NK_1'\right) - NK_2' \tag{3.7}$$

Thus, the operation represented by Equation 3.7 acts as a safeguard in case $(x - NK_1')$ suffers from massive cancellation. Hence, with a very low overhead a much higher accurate x_r can be obtained. The algorithm has been successfully implemented in software libraries. For instance, CRlibm uses different implementations of Cody and Waite's algorithm for low and intermediate arguments.

3.1.2 Payne and Hanek's Reduction Algorithm

Payne and Hanek's algorithm, introduced in [25], is extremely useful when the ordinary or the modified Cody and Waite algorithms are unable to produce a sufficiently accurate value of x_r; this usually happens at large arguments. Although the algorithm can be also used at intermediate and small arguments, the method is considered unfavorable due to the memory and computational penalty.

Let the FPN x within the target precision p be reformulated as $x = 2^{E_x - p + 1} \times X$, where X is integer where $2^{p-1} \leq X < 2^p$. Thus, Equation 3.2 can be rearranged as follows:

$$x_r = \frac{\pi}{2^M}\left(\frac{2^{M+E_x-p+1}}{\pi} \times X - N\right) \tag{3.8}$$

where M is a positive integer. Note that the quantity $(1/\pi)$ is a transcendental number. Thus, it is represented by an infinite series:

$$\frac{1}{\pi} = \sum_{i=0}^{\infty} W_i \times 2^{-i} \tag{3.9}$$

where $W_i \in \{0, 1\}$. Thus, the final form can be shown as follows:

$$x_r = \frac{\pi}{2^M}\left(2^{M+E_x-p+1} \times \left(\sum_{i=0}^{\infty} W_i \times 2^{-i}\right) \times X - N\right) \tag{3.10}$$

Since $x_r \in \left[0, \pi/2^M\right]$, the following condition should be fulfilled:

$$\left(2^{M+E_x-p+1} \times \left(\sum_{i=0}^{\infty} W_i \times 2^{-i}\right) \times X - N\right) \leq 1 \tag{3.11}$$

Thus, the subtraction operation in Equation 3.10, which may require a much higher precision than p depending on the argument value, need not be performed. Obtaining an accurate value of the fraction part of the quantity

$$2^{M+E_x-p+1} \times \left(\sum_{i=0}^{\infty} W_i \times 2^{-i}\right) \times X$$

is sufficient to compute x_r. We now only need to know the range of i values to obtain this quantity accurately. Note that when Equation 3.10 is unraveled, the component of

the series having $i \leq E_x - p$ are multiples of 2π and subsequently the components having $i > E_x - p$ should only be considered. We need also to find a higher bound on i such that we obtain an accurate value of x_r. If we need the fraction part with absolute error to become less than $2^{-\alpha}$, that implies that the last component should be

$$2\pi \times 2^{E_x - p - i} \times W_i \times X \approx 2^{3+p+E_x-p-i} < 2^{-\alpha}$$

where π is roughly approximated to 2^2 and X has a maximum of $2^p - 1 \approx 2^p$. Therefore, $E_x - p + 1 \leq i \leq \alpha + E_x + 4$.

In order to obtain the value of x_r, the fraction part must be multiplicated by $\pi/2^M$ as follows:

$$x_r \approx \frac{\pi}{2^M} \times \text{fraction}\left(2^{M+E_x-p+1} \times \left(\sum_{i=E_x-p+1}^{\alpha+E_x+4} W_i \times 2^{-i}\right) \times X\right) \tag{3.12}$$

Although this algorithm can compute efficiently reduced argument regardless of the value of x, the implementation of the method is very costly for both software and hardware designs. Payne and Hanek's algorithm was implemented for example in CRlibm and Sun Microsystem's libm, currently known as fdlibm [1].

3.2 Approximation polynomials

In order to evaluate a transcendental function, it should be approximated to some polynomial. Although the popular Taylor polynomials have great advantages due their easiness and maturity of analysis, they produce poor maximum error results compared to other approximation polynomials such as minimax, Chebyshev and Legendre approximation polynomials [21].

If we seek to approximate the continuous function f over a closed interval $C = [a, b]$ using an approximation polynomial P, there exist two types of approximations [20]:

1. *Least squares approximation* (also called L^2 approximation): L^2 approximation polynomials minimize the L^2 distance $\|f - P\|_{L^2}$ where

$$\|f - P\|_{L^2} = \left(\int_b^a \Big(f(x) - P(x)\Big)^2 \mathrm{d}x\right)^{\frac{1}{2}} \tag{3.13}$$

   Thus, L^2 approximation polynomials minimize the average error. Chebyshev series, Legendre and Jacobi orthogonal polynomials are examples of L^2 approximation polynomials.

2. *Least maximum approximation* (also called L^∞ and minimax approximation): L^∞ approximation polynomials minimize the L^∞ distance $\|f - P\|_{L^\infty}$ where

$$\|f - P\|_{L^\infty} = \max_{[a,b]} |f(x) - P(x)| \tag{3.14}$$

   Thus, L^∞ approximation polynomials minimize the maximum error.

[1] For more information, visit `http://www.netlib.org/fdlibm/readme`.

Approximation polynomials may be obliged to use large degrees to satisfy some required approximation error. Rational approximation was investigated in [12] to address this problem. Rational approximation can possibly produce much lower degree values for numerator and denominator polynomials compared to polynomial approximation assuming the same accuracy. Rational approximation is still unfavorable compared to polynomial approximation due to the known high cost of the division operation.

3.3 Minimax Approximation

A theorem by Karl Weierstrass in the late 19th century [17] shows that every continuous function f on a closed interval $[a,b]$ can be approximated by a polynomial P_n of degree n; assume that $\epsilon > 0$, therefore the following condition is fulfilled:

$$\|f - P\|_{L^\infty} < \epsilon \tag{3.15}$$

That is, there exists a polynomial that can approximate any continuous function over a closed interval with a whatever desired maximum error. Thus, L^∞ polynomials minimize the value of ϵ.

Chebyshev [21] showed that a unique minimax polynomial P^* exists if and only if there exist at least $n+2$ points within $[a,b]$, $(a \le x_0 < x_1 < x_2 < \ldots < x_{n+1} \le b)$ such that:

$$P_n(x_i) - f(x_i) = (-1)^i (P_n(x_0) - f(x_0)) = \pm \|f - P\|_{L^\infty}, \forall i \tag{3.16}$$

That is, the L^∞ distance equioscillates over at least $n+2$ points within $[a,b]$. Minimax polynomials can be realized by a numerical algorithm [27] by Remez introduced in 1934. Remez algorithm can be outlined as follows:

1. Set the index $k = 0$ and start with initial points $(a \le x_0 < x_1 < x_2 < \ldots < x_{n+1} \le b)$. Chebyshev nodes are preferred as initial points in order to obtain initially accurate approximation:

$$x_i = \frac{a+b}{2} + \frac{b-a}{2} \cos\left(\frac{(2i+1)\pi}{2n}\right) \qquad i = 0, 1, \ldots, n+1 \tag{3.17}$$

2. Solve the system of $(n+2)$ linear equations

$$\begin{aligned} P_{n,k}(x_0) - f(x_0) &= +\delta_k \\ P_{n,k}(x_1) - f(x_1) &= -\delta_k \\ &\vdots \\ P_{n,k}(x_{n+1}) - f(x_{n+1}) &= +\delta_k \end{aligned} \tag{3.18}$$

   where $P_{n,k}(x_i) = \sum_{j=0}^{n} a_{k,j} x_{k,i}^j$. Obtain $a_{k,j}, \forall j$ and δ_k.

3. Obtain the points having the extrema $y_{k,i}, \forall i$ of $f - P_{n,k}$ then use $x_{k+1,i} = y_{k,i}, \forall i$. Set $k \leftarrow k+1$ and then go to step 2.

As one can expect, the algorithm must be convergent. A theorem by de la Valée-Poussin [24] shows that

$$\upsilon_k = \frac{\left\| f - P_{n,k} \right\|_{L^\infty}}{\min\limits_{\forall i} |f(y_{k,i}) - P_{n,k}(y_{k,i})|} \geq 1 \tag{3.19}$$

and

$$\min_{\forall i} |f(y_{k,i}) - P_{n,k}(y_{k,i})| \geq \delta_k \tag{3.20}$$

Note that $\upsilon_k = 1$ and $P_{n,k} = P^*$ as $k \longrightarrow \infty$. Realistically, the iteration stops when

$$\max_{\forall i} |f(y_{k,i}) - P_k(y_{k,i})| - \min_{\forall i} |f(y_{k,i}) - P_k(y_{k,i})| < \hat{\epsilon} \tag{3.21}$$

were the value of $\hat{\epsilon}$ is predetermined.

There exist several commercial tools that can implement Remez algorithm. Unfortunately, we found them unable to produce large-degree/very accurate polynomials. Thankfully, we found a a free and open source tool called Sollya [3] that can produce minimax polynomials with whatever required accuracy. Sollya is generally useful for developing accurate elementary functions that can be implemented in software as well as hardware algorithms.

3.4 Polynomial Evaluation

Polynomial approximation provides us an efficient polynomial (i.e. the coefficients and the polynomial degree values) that minimizes the designated distance (i.e. L^2 or L^∞). Now, we need to know the efficient methods for evaluating the polynomial $P_n(x) = \sum_{i=0}^{n} a_i x^i$ in runtime. The method of evaluating the polynomial as well as the working precision and architecture decisions dictate the resulting accuracy and performance.

The widely known Horner's rule [22] is a simple method to evaluate P_n. It can be demonstrated as follows:

$$t_n = a_n \tag{3.22}$$

$$t_i = t_{i+1} x + a_i \quad \text{for } i = n-1, \ldots, 1, 0 \tag{3.23}$$

$$P_n = t_0 \tag{3.24}$$

or simply

$$P_n(x) = ((((a_n x + a_{n-1})\, x + a_{n-2}) \cdots)\, x + a_1)\, x + a_0 \tag{3.25}$$

Thus, n additions as well as n multiplications are needed. If FMA is used, only n operations are required.

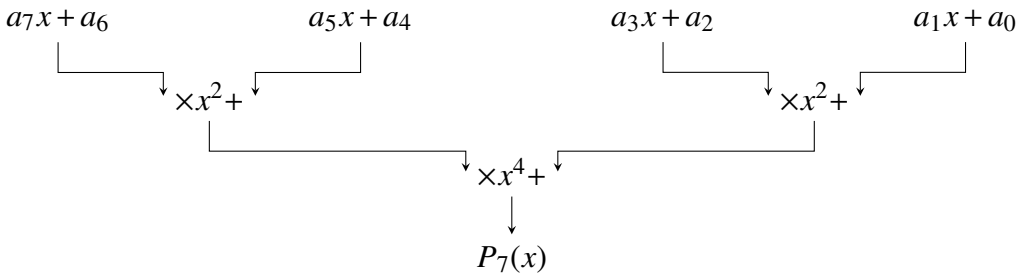
Estrin's scheme [8] is another method that improves Horner's method to exploit parallelism. Let $P_n(x)$ be split as follows:

$$P_n(x) = \sum_{0 \leq i < h} \left(a_{i+h} x^i \right) x^h + \sum_{0 \leq i < h} a_i x^i \tag{3.26}$$

where $h = (n+1)/2$ is power of 2. The two polynomials can be further split. For example, a polynomial of degree 7 can be evaluated as follows:

$$P_7(x) = \Big(((a_7x + a_6))\,x^2 + (a_5x + a_4)\Big)\,x^4 + (a_3x + a_2)\,x^2 + (a_1x + a_0)$$

where $P_7(x)$ is constructed as follows:



Therefore, the required steps grow logarithmically with the polynomial degree having the sufficient computational power.

3.5 Summary

In this chapter, range reduction algorithms have been discussed. As straightforward additive range reduction algorithms causes inaccurate results, more advanced algorithms such as Cody and Waite, and, Payne and Hanek, were developed to increase the accuracy of reduced arguments. Polynomial approximation types have been presented. Minimax approximation polynomials are realized using Remez algorithm. Horner's and Estrin's polynomial evaluation techniques have been presented.

Chapter 4: Table-Lookup Based Methods

4.1 Introduction

Table-based methods are the predominant hardware-based techniques for computing high-performance floating-point elementary functions. They are considered efficient in terms of speed as well as having the advantage of consuming reasonable amount of memory requirements especially when the desired accuracy is no more than the faithful rounding case.

In table-based methods, the working interval $C = [a, b]$ is divided into many smaller sub-intervals, usually of equal size, and a piece-wise polynomial of a low degree, mainly ranging from linear to cubic, is applied within the corresponding sub-interval. Dividing the original wide interval does not come for free, it obliges using relatively large look-up tables to store the sub-interval polynomial coefficients, ranging from several kbits to over 1 Mbits depending on the target precision, the polynomial degree and the required accuracy (e.g. faithful, correct, etc ...).

Generally speaking, there exists a trade-off between latency and memory to achieve a designated result accuracy. The wider the sub-interval, the larger the required degree to maintain a designated accuracy and subsequently a higher latency is incurred but at the advantage of having less memory due to having a fewer number of sub-intervals. Conversely, the narrower the sub-interval, the lower degree needed yet at the expense of using larger look-up tables.

Many notable achievements were made throughout the past 25 years to increase the accuracy of the table-based methods while reducing the memory and computing requirements. Pioneer works, such as [33, 10, 34, 35], paved the way to implement accurate floating-point elementary functions at high-speed and realizable lookup table sizes. Table-based methods can be divided into three paradigms as described in [19]:

1. Compute-bound methods: High-order polynomials with relatively small lookup-tables are used (e.g. the method by Tang [33]).

2. Table-bound methods: Large lookup-tables with very simple arithmetic circuitry are used. Bipartite [6] and Multipartite [19, 7] methods which use two or more lookup-tables with two or many-operand adders.

3. In-between methods: Mostly degree-1 to 3 piecewise polynomials are used with intermediate lookup-tables (e.g. The method given by Pineiro and Al. [26]).

In this chapter we present some memorable examples of hardware-oriented table-based methods. To our knowledge, all hardware-oriented methods assume a reduced range (e.g. $\sin(x)$ and 2^x for $x \in [0, 1]$ and $1/\sqrt{x}$ for $x \in [1, 2]$). Arguments out of such reduced intervals should be reduced using software routines or special-purpose circuitry.

4.2 Memorable Examples

4.2.1 Tang's Method

Tang's algorithm [33] is an archetype of the compute-bound methods. It uses large-degree polynomials with rather small lookup-tables. This paradigm is useful for high precisions (e.g. DP) where the required size of lookup-tables can be incredibly large for low-degree picewise polynomials to be implemented. The algorithm was developed for DP $\sin(x)$, 2^x and $\log_2(x)$ and it produces a theoretical bound of 0.57 ulps of accuracy.

4.2.1.1 $\sin(x)$ for $x \in [0, \pi/4]$

- Range reduction: if $|x| < 1/16$, compute $\sin(x)$ with some simple polynomial; otherwise, calculate the breakpoint c_{jk} where

$$c_{jk} = 2^{-j}(1 + k/8) \qquad j = 1,2,3,4 \text{ and } k = 0,1,\ldots,7$$

  Then define r as the reduced argument where $r = x - c_{jk}$ where $|r| \leq 1/32$

- Evaluation: compute $p(r) = \sin(r) - r$ and $q(r) = \cos(r) - 1$ as follows:

$$p(r) = p_1 r^3 + p_2 r^5 + \ldots + p_n r^{2n+1} \tag{4.1}$$
$$q(r) = q_1 r^2 + q_2 r^4 + \ldots + q_m r^{2m} \tag{4.2}$$

  where the coefficients $p_1, p_2, \ldots, p_n$ and $q_1, q_2, \ldots, q_m$ are computed using Remez algorithm.

- Reconstruction: $\sin(x) = \sin(c_{jk} + r)$ is reconstructed as follows:

$$\sin(x) = \sin(c_{jk})(\cos(r) - 1 + 1) + \cos(c_{jk})(\sin(r) - r + r) \tag{4.3}$$
$$\sin(x) \approx S_{jk} + rC_{jk} + p(r)C_{jk} + q(r)S_{jk} \tag{4.4}$$

  where S_{jk} and $C_{jk}, \forall j,k$ are the stored values of $\sin(c_{jk})$ and $\cos(c_{jk})$ respectively.

4.2.1.2 2^x for $x \in [-1, 1]$

- Range reduction: compute the reduced argument r where $r = |x - (m + c_k)| \log(2)$ where $|x - (m + c_k)| \leq 1/64$, $m = -1, 0, 1$, $c_k = k/32$ and $k = 0, 1, \ldots, 31$. Thus, $|r| \leq \log(2)/64$.

- Evaluation: compute the function $p(r) = e^r - 1$ where

$$p(r) = r + p_1 r^2 + \ldots + p_n r^{n+1} \tag{4.5}$$

  The coefficients $p_1, p_2, \ldots, p_n$ are computed using Remez algorithm.

Function	No. of table entries	No. of coefficients
$\sin(x)$	64	$n = 3$ and $m = 3$
2^x	32	$n = 5$

Table 4.1: The number of lookup-table entries and coefficients given by Tang

- Reconstruction: 2^x is reconstructed as follows:

$$2^x = 2^{m+c_k} e^r \tag{4.6}$$
$$2^x \approx 2^m (T_k + T_k p(r)) \tag{4.7}$$

where the values of $T_k, \forall k$ are the stored values of 2^{c_k}.

The size of the lookup tables is very attractive for the somewhat low-degree polynomial and the high accuracy obtained. For the case of $\sin(x)$, if the values of S_{jk} and C_{jk} are stored in DP, the size of the table-lookup is then 4 kbits of size. Applying the unmodified method to obtain correctly rounded results performs poorly (e.g. less than 1% having the polynomial evaluated using DP FMA for the sine case). We do not know of the existence of any variant to Tang's method developed to other designated accuracy and precision values. Therefore, we cannot verify the requirements that satisfy correct rounding using a convenient variant of the method.

4.2.2 Bipartite and Multipartite Methods

The bipartite method (also called the table-and-addition method) was first introduced in [6]. The method was used to compute $1/x$ and $1/\sqrt{x}$ in AMD-K7 [23] as initial approximations to eventually compute division and square root operations. The method is still favorable for rather small precisions (i.e. $p \leq 20$).

If we imagine that the output of every fixed-point argument x where $x \in 1 \leq x < 2$ of width p bits is stored in a lookup table, we will need then a lookup-table of $p2^p$ bits to store all outputs.

Now, assume that x is divided into three sections each of width k, i.e. $p = 3k$ such that x is arranged as follows:

$$x = x_0 + 2^{-k} x_1 + 2^{-2k} x_2$$

where $x_0 = 1.m_1 m_2 \ldots m_k$, $x_1 = 0.m_{k+1} m_{k+2} \ldots m_{2k}$ and $x_2 = 0.m_{2k+1} m_{2k+2} \ldots m_{p-1}$. Remember Taylor's form for x near $\bar{x}$ is calculated as follows:

$$f(x) = \sum_{i=0}^{n} \frac{f^{(i)}(\bar{x})(x-\bar{x})^i}{i!} + \frac{f^{(n+1)}(\zeta)(x-\bar{x})^{(n+1)}}{(n+1)!} \tag{4.8}$$

where the second term of the right side is the Lagrange remainder and $\zeta \in (\bar{x}, x)$. Thus, the Taylor expansion of $f(x)$ centered around the point $x_0 + 2^{-k} x_1$ up to the first order term is calculated as follows:

$$f(x) = f\left(x_0 + 2^{-k}x_1\right) + 2^{-2k}x_2 f'\left(x_0 + 2^{-k}x_1\right) + \epsilon_1 \tag{4.9}$$

where $\epsilon_1 \leq 2^{-4k-1}\max(|f''|)$. Now, we further approximate $f'\left(x_0 + 2^{-k}x_1\right)$ using Taylor's form around x_0 as follows:

$$f'\left(x_0 + 2^{-k}x_1\right) = f'(x_0) + \epsilon_2 \tag{4.10}$$

where $\epsilon_2 \leq 2^{-k}\max(|f''|)$. Thus, $f(x)$ is evaluated as follows:

$$f(x) = f\left(x_0 + 2^{-k}x_1\right) + 2^{-2k}x_2\left(f'(x_0) + \epsilon_2\right) + \epsilon_1 \tag{4.11}$$

$$f(x) = f\left(x_0 + 2^{-k}x_1\right) + 2^{-2k}x_2 f'(x_0) + \epsilon_1 + 2^{-2k}x_2\epsilon_2 \tag{4.12}$$

$$f(x) = a_0(x_0, x_1) + a_1(x_0, x_2) + \epsilon \tag{4.13}$$

where

$$\epsilon \leq \left(2^{-4k-1} + 2^{-3k}\right)\max(|f''|) \tag{4.14}$$

The values of a_0 and a_1 are stored in two lookup-tables each with address width of $2p/3$ bits. Thus, only an addition operation is required to obtain $\hat{f}(x) = a_0 + a_1$. If we assume that $p = 3k$, then the total memory required is $4p/3 \times 2^{2p/3}$ bits. This is remarkably smaller than the case of the primitive output storage of $p2^p$ bits for intermediate p values, i.e. $16 - 20$ bits. However, this advantage dwindles as the target precision increases for the memory grows exponentially in either case.

The multipartite method (or the $(j+1)$-partite method) was developed in [19] to generalize the bipartite method. Assume that x is divided into $(2j+1)$ k-bit parts as follows:

$$x = \sum_{i=0}^{2j} x_i 2^{-ik}$$

Thus, the Taylor polynomial for x near $x_0 = \sum_{i=0}^{j} x_i 2^{-ik}$ is calculated as follows:

$$f(x) = f\left(\sum_{i=0}^{j} x_i 2^{-ik}\right) + \left(\sum_{i=j+1}^{2j} x_i 2^{-ik}\right) f'\left(\sum_{i=0}^{j} x_i 2^{-ik}\right) \tag{4.15}$$

We further approximate $f'\left(\sum_{i=0}^{j} x_i 2^{-ik}\right)$ into j terms with corresponding error ϵ_{i+1} and

subsequently $f(x)$ is calculated as follows:

$$\begin{aligned} f(x) = f\left(\sum_{i=0}^{j} x_i 2^{-ik}\right) + \epsilon_0 \\ + x_{j+1} 2^{-(j+1)k} f'\left(\sum_{i=0}^{j-1} x_i 2^{-ik}\right) + \epsilon_1 \\ + x_{j+2} 2^{-(j+2)k} f'\left(\sum_{i=0}^{j-2} x_i 2^{-ik}\right) + \epsilon_2 \\ \vdots \\ + x_{2j} 2^{-2jk} f'(x_0) + \epsilon_j \end{aligned} \tag{4.16}$$

where $\epsilon_0 \leq 2^{-2(j+1)k-1} \max(|f''|)$ and $\epsilon_i \leq 2^{-(2j+1)k} \max(|f''|)$

$$\begin{aligned} f(x) = a_0\left(x_0, x_1, \ldots, x_j\right) + a_1\left(x_0, x_1, \ldots, x_{j-1}, x_{j+1}\right) \\ + a_2\left(x_0, x_1, \ldots, x_{j-2}, x_{j+2}\right) + \cdots + a_j\left(x_0, x_{2j}\right) + \epsilon \end{aligned} \tag{4.17}$$

where

$$\epsilon = \sum_{i=0}^{j} \epsilon_i \leq \left(2^{-2(j+1)k-1} + j2^{-(2j+1)k}\right) \max(|f''|) \approx j2^{-(2j+1)k} \max(|f''|) \tag{4.18}$$

Many variations of the method were developed to increase the accuracy and decrease the size of the lookup tables. In [35] a second-order Taylor polynomial is developed for the SP case. The symmetric bipartite table method (SBTM) introduced in [29], imposes an offset to the center of the approximation such that a symmetry is obtained around it. Thus, storing the first-order term of either the right side or the left of the center becomes sufficient while the other side is simply deducted from the symmetry. Subsequently, the size of the lookup table storing a_1 values is reduced by a factor of two. The bipartite and multipartite methods are not convenient to produce correct rounding because the total relative error is expected to be closer to the faithful case (Note that $p = 3k$ for Equation 4.14 and $p = (2j+1)k$ for Equation 4.18).

4.2.3 Pineiro's Method

The method by Pineiro et Al. [26] follows the in-between paradigm using degree-2 piecewise minimax polynomials in order to compute faithfully rounded IEEE 754 SP elementary functions. The method was applied for several elementary functions within accuracy of 1, 2 and 4 ulps.

Assume the argument x of width p is divided into two sub-words $x_1 = [x_1 x_2 \ldots x_m]$ and $x_2 = [x_{m+1} x_{m+2} \ldots x_p]$ where

$$x = x_1 + x_2 2^{-m}$$

$f(x)$	Table size in bits		
	1 ulp	2 ulps	4 ulps
$1/x$	$2^7 \times (25+16+10) = 6528$	$2^7 \times (24+15+9) = 6144$	$2^6 \times (25+17+12) = 3456$
$\sqrt{x}$	$2 \times 2^6 \times (25+15+9) = 6272$	$2 \times 2^5 \times (26+17+11) = 3456$	$2 \times 2^5 \times (23+14+9) = 2944$
$1/\sqrt{x}$	$2 \times 2^7 \times (25+15+9) = 12544$	$2 \times 2^6 \times (24+14+11) = 6272$	$2 \times 2^6 \times (23+13+9) = 5760$
2^x	$2^6 \times (25+16+10) = 3264$	$2^5 \times (25+18+13) = 1792$	$2^5 \times (24+15+10) = 1568$
$\log_2(x)$	$2^7 \times (26+16+10) = 6656$	$2^6 \times (27+19+14) = 3840$	$2^6 \times (24+16+10) = 3200$
$\sin(x)$	$2^6 \times (27+19+12) = 3712$	$2^6 \times (25+17+10) = 3328$	$2^6 \times (25+15+9) = 3136$

Table 4.2: The size of lookup-tables given by Pineiro and Al. for several SP elementary functions for the case of error within 1, 2 and 4 ulps

Thus, the degree-2 approximation piecewise polynomial $P_m(x)$ of $f(x)$ within $[x_1, x_1 + 2^{-m}]$ can be computed as follows

$$P_m(x) = a_0(x_1) + a_1(x_1)x_2 + a_2(x_1)x_2^2 \qquad x_2 \in \left[x_1, x_1 + 2^{-m}\right] \tag{4.19}$$

Thus, we have 2^m sub-intervals each has a corresponding tuple of a second degree minimax polynomial coefficients (a_0, a_1, a_2) with a corresponding tuple of coefficient lengths (t, p, q). The method uses what is called the enhanced minimax approximation that makes sure that the value of m is minimized as the size of the look-up table grows exponentially with m. Furthermore, the lengths of the coefficients should be minimized to reasonable wordlengths without violating the designated accuracy. The method can be summarized as follows:

1. Find m that gives a preliminary minimax polynomial with unrestricted wordlengths for the three coefficients that corresponds to some preliminary designated approximation error.

2. Set preliminary large values of (t, p, q) and check whether the bound of the approximation error is exceeded; if the error budget is still smaller than the approximation error, decrease (t, p, q) until the error budget meets the approximation error; otherwise, increment m and go to step 2.

3. Using exhaustive simulation, find the best rounding constant that can be used as a bias to minimize the overall error.

Table 4.2 shows the obtained results for error within 1 (i.e. faithfully rounded result), 2 and 4 ulps for six elementary functions.

The architecture uses a specialized squaring unit to generate x_2^2, a truncation is applied to the squarer as x_2^2 is guaranteed to be less than 2^{-2m} and thus there exists some room to neglect many of the least significant bits without sacrificing the result designated accuracy. We have 3 components to be summed: $a_2x_2^2$, a_1x_2 and a_0. The first two components are generated as partial products using the signed-digit radix-4 representation. This reduces the number of partial products to about half compared to the conventional binary approach

and subsequently a faster addition is achieved. The third term is added to the function-specific rounding constant. A fused accumulation tree is used to compress the partial products using 3:2 and 4:2 carry-save adders (CSAs) into a two conventional words that are fed to a fast carry-propagation adder (CPA).

The method is currently the preferred choice for SP elementary functions for GPUs and general-purpose processors. A related work in [32] uses also degree-2 minimax polynomials yet reduces the size of lookup-tables by around 40% by imposing same a_0 for two adjacent subintervals. In [14], degree-1 and 2 interpolations are investigated and compared to their corresponding approximation minimax polynomials for precisions up to SP.

4.2.4 Schulte's Method

Schulte's method [28, 30] is the main contribution for hardware-based correctly rounded floating-point elementary functions. The method was applied for four elementary functions, namely: $1/x$, $\sqrt{x}$, 2^x and $\log_2(x)$, using degree-1, 2 and 3 Chebyshev series piecewise polynomials for $p = 16$ and 24. This method serves as the best available guide of the estimated required size of lookup-tables for correctly rounded SP elementary functions using table-based methods.

Assume the argument $x \in [1,2)$ where $x = 1 + x_h + x_l 2^{-k}$. The Chebyshev series approximation polynomial $P_m(x) = \sum_{i=0}^{n-1} a_i(x_h) x_l^i$ of degree $n-1$ within the subinterval $\left[x_h, x_h + 2^{-k}\right]$ is computed as follows:

1. The Chebyshev nodes are obtained initially within $[-1,1]$ as follows:

$$t_i = \cos\left(\frac{(2i+1)\pi}{2n}\right) \qquad i = 0,1,\ldots,n-1 \tag{4.20}$$

2. The Chebyshev nodes are transformed from $[-1,1]$ to $\left[x_h, x_h + 2^{-k}\right]$ using the relation:

$$x_i = x_h + (t_i + 1)2^{-k-1} \qquad i = 0,1,\ldots,n-1 \tag{4.21}$$

3. The Lagrange polynomial interpolates the Chebyshev nodes within $\left[x_h, x_h + 2^{-k}\right]$ and is computed as follows:

$$P_m(x) = \sum_{i=0}^{n-1} f(x_i) L_i(x) \tag{4.22}$$

where

$$L_i(x) = \frac{\prod_{j=0, j\neq i}^{n-1} (x - x_j)}{\prod_{j=0, j\neq i}^{n-1} (x_i - x_j)} \tag{4.23}$$

4. The coefficients are rounded to lower precisions in order to explore the smallest arithmetic and memory requirements without violating the accuracy required for correct rounding.

Degree	Coefficient length				Table size		
	a_0	a_1	a_2	a_3	Words	Bits/word	Total bits
1	36	21	–	–	512K	57	28.5M
2	40	31	19	–	16K	90	1.4M
3	41	35	27	18	1K	121	121K

Table 4.3: Memory requirements for the four SP elementary functions for the case of degree 1, 2 and 3

The maximum approximation error δ for Chebyshev series polynomials is calculated as follows:

$$\delta = \frac{2^{-n(k+2)+1}\left|f^{(n)}(\xi)\right|}{n!} \tag{4.24}$$

where $\xi \in \left[x_h, x_h + 2^{-k}\right]$ gives the maximum value of $f^{(n)}$ over the interval. We have two factors to control the accuracy and the underlying architecture:

1. If k gets incremented, δ is reduced by a factor of 2^n but at the expense of doubling the lookup-table size.

2. If n gets incremented, δ is reduced by a factor of 2^{k+2} but the expense of extra delay and complexity for the arithmetic circuitry.

Let us assume $\delta \leq 2^{-q}$. Thus, k is calculated as follows:

$$k = \left\lceil \frac{q - 2n + 1 + \log_2\left(\left|f^{(n)}(\xi)\right|\right) - \log_2(n!)}{n} \right\rceil \tag{4.25}$$

Table (4.3) shows the lookup table size for the four implemented elementary functions for the case of degree-1, 2 and 3.

The architecture is very similar to Pineiro's method (see Figure 4.1). Ad-hoc powering units and partial-product generators are needed followed by a multi-operand adder.

4.3 CRlibm

After presenting several popular hardware-oriented table-based methods, we now discuss a popular software example that offers correct rounding.

CRlibm [5] is a portable, efficient and IEEE 754 compliant correctly rounded software math library for DP elementary functions. It uses a 2-step evaluation technique for computing the final result. A fast step that suffices for most arguments with having 2^{-60} to 2^{-80} of function-specific accuracy; a slower and more accurate step includes the rest of the arguments having 2^{-120} to 2^{-150} of function-specific accuracy relying on knowledge of the hardest-to-round cases given by Lefèvre's algorithm (see Section 2.3.2). The

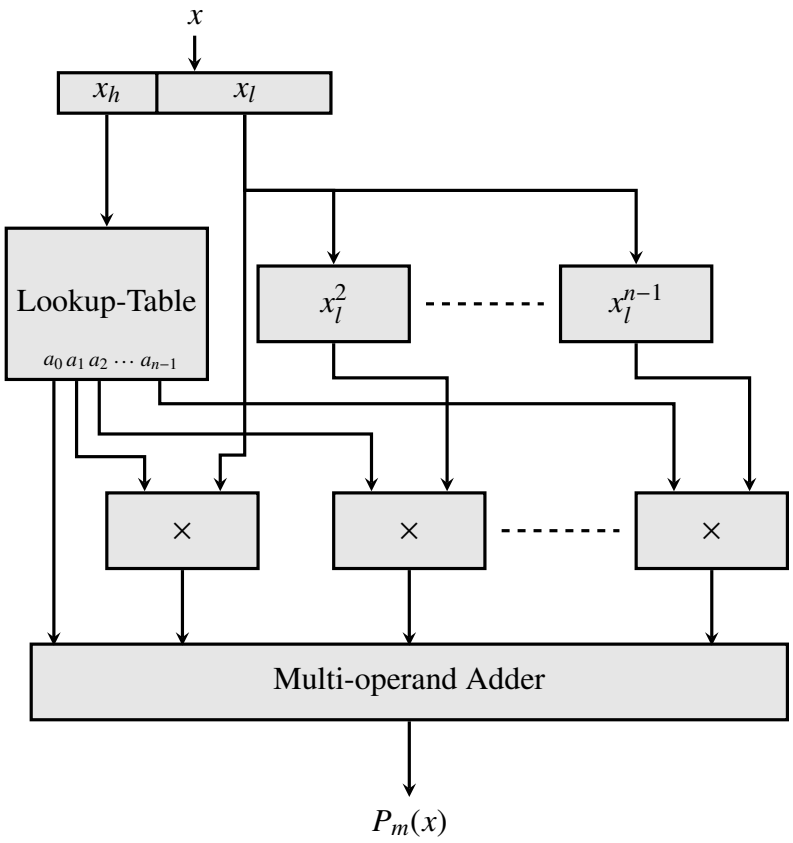


Figure 4.1: The architecture of Schulte's method

algorithms used greatly differ from a function to another. We give a brief description for the implementation of trigonometric functions. The fast step can be summarized as follows:

- Range reduction: Let $x \in [-\pi/4, \pi/4]$. Larger arguments are reduced to x using periodicity. Let $x = a + y$ where y is the reduced argument and

$$a = \frac{N\pi}{256}$$

  In order to return DP correctly rounded result, $y \in [-\pi/512, \pi/512]$ is considered a double-double variable represented by two DP FPNs y_h and y_l where $y = y_h + y_l$. The value of y is obtained using several implementations of Cody and Waite's method or Payne and Hanek's method depending on the value of x. Likewise, the value of $\sin(a)$ is represented by $sa_h + sa_l$ and $\cos(a)$ by $ca_h + ca_l$ where (sa_h, sa_l, ca_h, ca_l) are stored in lookup tables.

- Evaluation: compute $\sin(y) = (1 + t_s)(y_h + y_l)$ and $\cos(y) = 1 + t_c$ where

$$t_s = (y_h + y_l)^2 \left(s_3 + (y_h + y_l)^2 \left(s_5 + (y_h + y_l)^2 \left(s_7 + (y_h + y_l)^2\right)\right)\right) \tag{4.26}$$

$$t_c = (y_h + y_l)^2 \left(c_2 + (y_h + y_l)^2 \left(c_4 + (y_h + y_l)^2 \left(c_6 + (y_h + y_l)^2\right)\right)\right) \tag{4.27}$$

  where s_3, s_5, s_7 and c_2, c_4, c_6 are the coefficients of Taylor's polynomial.

- Reconstruction: Evaluate $\sin(x)$ as follows:

$$\sin(x) = (sa_h + sa_l)(1 + t_c) + (ca_h + ca_l)(y_h + y_l)(1 + t_s) \tag{4.28}$$

  Likewise, $\cos(x)$ is evaluated as follows:

$$\cos(x) = (ca_h + ca_l)(1 + t_c) - (sa_h + sa_l)(y_h + y_l)(1 + t_s) \tag{4.29}$$

  The multiplication of the least significant terms such as $sa_l y_l t_s$ are neglected as they do not contribute to the final DP result.

The accurate step uses Payne and Hanek's range reduction algorithm into $[-\pi/4, \pi/4]$ followed by a large-degree polynomial. A degree-25 polynomial is used for $\sin(x)$ with an approximation error of 2^{-125} for $x \in [-\pi/4, \pi/4]$ and 2^{-158} for $|x| < 2^{-17}$ while a degree-26 polynomial is used for $\cos(x)$ with approximation error of 2^{-132} for $x \in [-\pi/4, \pi/4]$ and 2^{-168} for $|x| < 2^{-18}$.

4.4 Summary

In this chapter, we have discussed the different paradigms of the table-based methods and presented memorable algorithms of each paradigm. Bipartite and Multipartite methods are preferred for low precisions, i.e. less than 16 bits. For SP, quadratic minimax polynomials

are preferred and have been successfully implemented in modern GPUs. To evaluate the polynomial, table-based methods use special multipliers, ad-hoc powering units and multi-operand adders.

The main contribution that explicitly addresses the case of correct rounding for SP was described by Schulte. Being a table-based method, it had to use large lookup tables to satisfy the correct rounding accuracy while keeping the polynomial degree either 1, 2 or 3 with memory requirements at 28.5 Mb, 1.44 Mb or 121 Kb respectively to evaluate four elementary functions (namely: 2^x, $\log_2(x)$, $\sqrt{x}$ and $1/x$) within reduced ranges. Moreover, a brief description of the CRlibm library implementation has been presented.

Chapter 5: A Table Lookup-less Method for Correctly Rounded Trigonometric Functions

5.1 Motivation

The method by Schulte discussed in the previous chapter shows the high memory cost required to obtain correctly rounded SP elementary functions using the table-based degree-1, 2 or 3 piecewise polynomials. The results of memory requirements given by Schulte's method imply that the lookup table sizes have to be increased from several kbits to hundreds of kbits per function to improve the result accuracy from the faithful case to the correct case in order to maintain the quadratic polynomial. One could also imagine how enormous and infeasible memory requirements it would cost if correctly rounded DP results are needed using the same, or any similar, low degree piecewise polynomials.

Rather than dividing the reduced interval into many sub-intervals, we can choose a single wide reduced interval within which all arguments lie. Larger arguments are transformed directly to that interval. Now, we should determine what that reduced interval is. More importantly, we should determine the polynomial with the required accuracy that achieves correct rounding for all arguments within such an interval using realizable memory and computing requirements.

As we seek to replace the table-based method with a table lookup-less one, a couple of questions should be raised:

1. To what extent can a single reduced interval decrease the memory requirements?

   Given that Schulte's degree-1 or 2 piecewise polynomials require several Mbits or kbits respectively, and knowing the fact that increasing the degree of the piecewise polynomial decreases the number of sub-intervals assuming fixed entire interval and required accuracy, how much memory size can a single polynomial for a single interval consume?

2. What is the increase of the computational cost?

   Using a single polynomial for the entire interval dictates increasing the degree of the approximation polynomial which mainly translates into a higher latency with a possibly higher logic cost. Moreover, the wordlengths of the computed coefficients should be reasonable in order to implement realizable memory and logic operands' wordlengths. Furthermore, the effect of round-off errors may considerably grow due to the large number of arithmetic operations and subsequently we may be obliged to increase the working precision which translates into higher latency and area requirements in order to mitigate the effect of round-off errors.

5.2 Overview of the Method

The main goal of this chapter is to introduce a table lookup-less method in order to compute correctly rounded IEEE-754 SP trigonometric functions. The method simply computes

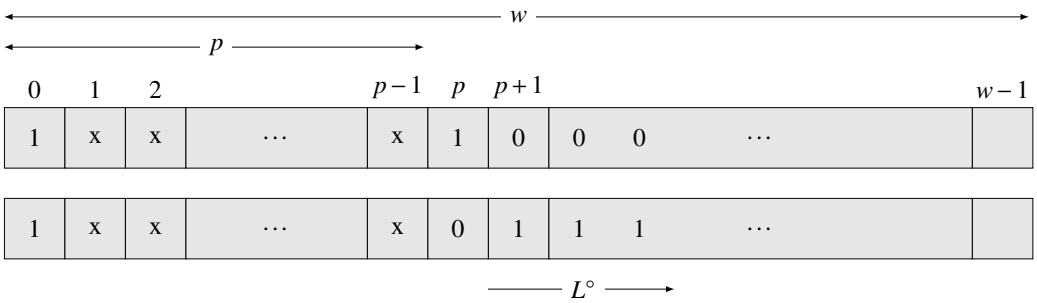


Figure 5.1: Mantissa of the working precision showing the target precision and the length of the chain L° assuming RN mode

correctly rounded $\sin(x)$ and $\cos(x)$ for $|x| < 2^{53}$. Although we initially address RN mode for the sake of priority and simplicity, we later show that the method is applicable for RZ, RU and RD.

The basic method uses a single minimax approximation polynomial of a large degree, degree-15 to 17, valid over a single relatively wide interval $[0,2]$. A standard DP FMA is assumed to carry out the polynomial evaluation using the straightforward Horner's method with the help of a simple control unit. The method uses the FMA with additional simple logic to implement a unified low-cost range reduction algorithm, a modified version of Cody and Waite's method, for all FPN arguments within the interval $[2,2^{53}]$ yet without penalizing low and intermediate argument values. The reconstruction is implicit and lossless; only a sign flip is needed.

However, the apparent disadvantage of our method is the requirement for a higher latency compared to table-based methods. This problem can be mitigated to some extent as will be shown later by using a set of a few piecewise polynomials, we propose a set of four piecewise polynomials, to reduce the average and worst-case latencies (the maximum piecewise polynomial degree becomes 11).

5.3 A Table Lookup-less Approach

Computing $\hat{f}_p^{\circ}(x)$, the designated result where $\circ \in \{s,c\}$ to denote the sine and cosine functions respectively within the target precision $p = 24$ for our case of SP, involves the three main steps discussed in Chapter 3, all of which are carried out within the working precision of DP, i.e. $w = 53$ (see Figure 5.1):

1. Range reduction: The argument x is transformed to the reduced argument x_r within a smaller range $C = [0,b]$ as follows:

$$N = \left\lfloor \frac{x}{b} \right\rfloor \tag{5.1}$$

$$x_r = x - b \times N \tag{5.2}$$

$$i = N \bmod 2^{M+1} \tag{5.3}$$

where the value of b is chosen as $b = \pi/2^M$, where $M = 0, 1, 2, \ldots$ and $i = 0, 1, \ldots, \left(2^{M+1} - 1\right)$ where the index i represents the $(i+1)^{\text{th}}$ sector out of a total of 2^{M+1} sectors within $[0, 2\pi]$ and it controls the reconstruction step.

2. Evaluation: The reduced argument x_r is fed to an approximation polynomial $P_n^\circ(x_r)$ of degree n where $\circ \in \{s, c\}$. We assume a single non-pipelined DP FMA. Thus, the polynomial evaluation is carried out using Horner's scheme in n cycles.

3. Reconstruction: The pre-rounded result $\hat{f}_w^\circ(x)$ is obtained from $P_n^\circ(x_r)$ using the trigonometric identities:

$$\hat{f}_w^s(x) = P_n^s(x_r)\cos(ib) + P_n^c(x_r)\sin(ib) \tag{5.4}$$
$$\hat{f}_w^c(x) = P_n^c(x_r)\cos(ib) - P_n^s(x_r)\sin(ib) \tag{5.5}$$

where the floating-point representations of $\sin(ib)$ and $\cos(ib)$ are stored in look-up tables.

The fourth step is called the *final rounding* where the designated final result $\hat{f}_p^\circ(x)$ is obtained by rounding $\hat{f}_w^\circ(x)$ to the target precision p as follows:

$$\hat{f}_p^\circ(x) = \mathrm{R}_p\left(\hat{f}_w^\circ(x)\right) \tag{5.6}$$

As discussed earlier in Chapter 3, the first three steps are obviously not entirely independent, they affect and depend on each other. It also should be clear that our ultimate goal is to make sure that $f_p^\circ(x) = \hat{f}_p^\circ(x)$ within the whole range.

As discussed in Chapter 2 (see Section 2.3.2), we can directly produce $\hat{f}_p^s(x) = x$ and $\hat{f}_p^c(x) = 1$ for $E_x < -12$. Thus, we unify the argument value comparison on the basis of the exponent section only. Therefore, the whole range of interest becomes $\Psi = [2^{-12}, 2^{53}]$.

5.3.1 Determining the value of M

There exists a trade-off between M and the degree n of the approximation polynomial P_n°. The higher the value of M, the narrower the interval C and subsequently the lower the degree n of P_n° required for a predetermined value of ϵ_{approx} and subsequently m°. This trade-off translates into another trade-off between latency and area; the higher the value of M, the lower the latency needed to evaluate P_n°, yet at the expense of higher memory required for the reconstruction step and therefore a higher area.

In order to obtain an optimal value of M, we should first find m_*° (see Section 2.3.2). As presented in Chapter 2, Table 2.2 shows that the values of L_*^s and L_*^c are 26 and 28 respectively for $x \in [2^{-12}, \pi/2]$ using exhaustive search. This means that our choice of $w = 53$ may be preliminarily sufficient for almost all argument values, i.e. including arguments requiring the range reduction step.

The value $m^\circ(x)$ can be calculated as follows:

$$m^\circ(x) = L^\circ(f^\circ(x)) + p - E_{f^\circ(x)} + 1 \tag{5.7}$$

M	n	$\hat{m}^s$	$\hat{m}^c$
1	14	59	59
1	15	65	65
2	11	57	56
2	12	62	63
3	9	56	54
3	10	61	63

Table 5.1: The polynomial degree n and the corresponding value of $\hat{m}$ for $f^s(x)$ and $f^c(x)$ where $M = 1, 2, 3$

where $E_{f_p^\circ(x)}$ is the binary exponent of $f_p^\circ(x)$. By applying a similar exhaustive testing within Ψ, we find that $m_*^\circ = 60$ and it happens when $x < \pi/2$. Note that, as opposed to the definition of m_*° in Chapter 2 which represents the relative required accuracy, we here intend to find the absolute required accuracy.

Table 5.1 shows the values of $\hat{m}$, the computed values of m using Sollya corresponding to different values of n and M. Thus, one can notice that $M = 1$ is the judicious choice when we try to implement the reconstruction represented by Equation 5.4 or 5.5 depending on the designated final result for two reasons:

1. The case of $M = 1$ requires evaluating either P_n^s or P_n^c. However, for $M > 1$, we must evaluate both P_n^s and P_n^c; i.e. we expect for our architecture a latency of $2n$ cycles for the evaluation plus two extra cycles to implement the reconstruction itself.

2. Memory requirements represented by $\sin(ib)$ and $\cos(ib)$ increase exponentially with M accompanied by the obligation of using look-up tables for $M > 1$ to implement the reconstruction.

Thus, the choice $M = 1$ enables us to implement an implicit and lossless reconstruction step as well as the advantage of having to evaluate one polynomial which translates into lower latency with no memory requirements for the reconstruction. Therefore, it is the convenient choice in terms of area and latency, and subsequently our working interval becomes $C[0, \pi/2]$. Thus, Table 5.1 indicates that a degree of 14 or 15 is theoretically sufficient for our case of $M = 1$ and $m_*^\circ = 60$.

5.3.2 Range Reduction

The range reduction step is only required when x does not lie within C. Its role is to transform x into another argument $x_r \in C$. As discussed in Chapter 3, we should not use the direct range reduction algorithm shown in Equations 5.1–5.3, where we store the floating-point representations of $2/\pi$ and $\pi/2$ within the working precision w, i.e. $b = \mathrm{RN}_w(\pi/2)$ and $(1/b) = \mathrm{RN}_w(2/\pi)$. Such a direct method causes incorrect $\hat{f}_p^\circ(x)$ results even at low values of x suffering from massive cancellation at the implementation of

Equation 5.2 due to the insufficient accuracy of representing b which stems from the transcendental nature of π. The situation worsens exponentially as E_x increases.

In order to obtain accurate x_r results, we increase the accuracy of b by using a modified version of Cody and Waite's method (see Section 3.1.1) as follows:

$$b = K_1' + K_2' \tag{5.8}$$

$$K_1' = \mathrm{RN}_w\left(\frac{\pi}{2}\right) \tag{5.9}$$

$$K_2' = \mathrm{RN}_w\left(\frac{\pi}{2} - K_1\right) \tag{5.10}$$

Therefore, Equation 5.2 is reformulated as $x_r = x - NK_1' - NK_2'$ and subsequently the range reduction step is implemented as follows:

$$t \Leftarrow \mathrm{FMA}\,(x,\, 1/b,\, 0) \tag{5.11}$$

$$N, i \Leftarrow \mathrm{FLR}\,(t) \tag{5.12}$$

$$t \Leftarrow \mathrm{FMA}\left(N,\, -K_1',\, x\right) \tag{5.13}$$

$$x_r \Leftarrow \mathrm{FMA}\left(N,\, -K_2',\, t\right) \tag{5.14}$$

where FLR operator involves the flooring operation as well as extracting i, which represents the two bits corresponding to the weights of 2^0 and 2^1 of N, and t is assumed to be a dummy register. All FMA operations use RN mode. We assume that FLR is executed in a dedicated cycle as the FMA is assumed to be unmodified for the sake of simplicity; subsequently, the entire range reduction step is implemented in 4 cycles. The implementation of the FLR operation, which can be easily realized, is not discussed here. Using this range reduction implementation, if Equation 5.13 suffers a massive cancellation, its output will be exact and therefore the output of Equation 5.14 will be still accurate. Note that as the value of E_x increases, the massive cancellation not only happens because the produced value of x_r is very small but also as x and NK_1' are both large values while we expect $x_r < \pi/2$.

The aforementioned range reduction method works well for argument values roughly up to 2^{29}. For $29 \leq E_x < 53$, x_r is not guaranteed to maintain that $x_r \in C$. The direction to where x_r tends to lie, depends on the value of $1/b$, or more precisely on the rounding mode of the stored representation of $2/\pi$. If RN or RU is used, $x_r \in [a', \pi/2]$ where $a' < 0$. However, if RZ or RD is used, $x_r \in [0, b']$ where $b' > \pi/2$. The values of $|a'|$ and b', using exhaustive testing, are guaranteed to grow to no more than $\pi/2$ and 2 respectively as long as $E_x < 53$. Figure 5.2 shows the change of the maximum values of a' and b' for $x = [2^{29}, 2^{53}]$. As one can expect, the values of $\log_2\left(\max_{[2,x]}(|a'|)\right)$ and $\log_2\left(\max_{[2,x]}(b' - \pi/2)\right)$ equal $-\infty$ for $x < 2^{29}$ as $a', b' \in [0, \pi/2]$ for such case.

Thus, we can simply solve this problem by extending the range within which P_n° is valid from C to $C' = [0, 2]$ while storing $1/b = \mathrm{RZ}_w(2/\pi)$. A second and even stronger reason to extend the valid interval of P_n° to C' is to avoid the costly comparison between x having $E_x = 0$ and $\pi/2$ to determine whether x requires range reduction and so as to unify the comparison on the basis of the value of E_x only. Thus, we have a unified range reduction algorithm for all arguments requiring range reduction, i.e. for $x \in [2^1, 2^{53}]$. It should be

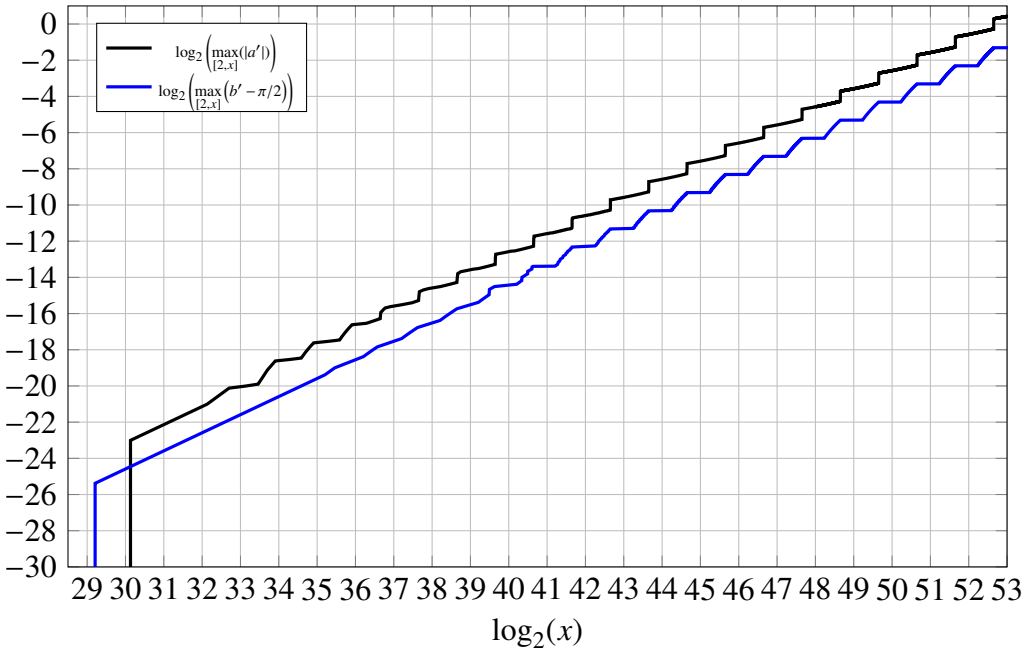


Figure 5.2: The change of a' and b' for $2^{29} \leq x < 2^{53}$

clear that the only change is that the polynomial P_n° is now valid within C' instead of C while the range reduction implementation is still the same. Thus, for $E_x < -12$, a direct handling for small values is carried out; for $-12 \leq E_x < 1$, the polynomial evaluation is directly carried out; for $1 \leq E_x < 53$, the range reduction is carried out followed by the polynomial evaluation.

Table 5.2 shows that this range extension usually costs an increase of n by 1 to roughly maintain the value of m shown in Table 5.1. Thus, it shows that $n = 15$ or $n = 16$ is now theoretically sufficient in order to adapt to $m_*^\circ = 60$.

n	$\hat{m}^s$	$\hat{m}^c$
13	49	50
14	55	54
15	59	60
16	65	64
17	69	70

Table 5.2: The polynomial degree n and the corresponding value of m for $f^s(x)$ and $f^c(x)$ where $x \in [0, 2]$ for $M = 1$

5.3.3 Polynomial Computation and Evaluation

Polynomial computation, that is the computation of the polynomial coefficients, was carried out using Sollya [3], a free and open source tool. We use Sollya to compute the minimax approximation polynomial using Remez algorithm and obtain the corresponding minimax approximation error.

Computing the polynomial $P_n^\circ(x_r) = \sum_{j=0}^{n} a_j x_r^j$ in Sollya interactive tool was accomplished using the remez() function. This function allows to obtain whatever required accuracy yet at the expense of a high resulting polynomial degree and large widths within which the coefficients are stored. Another function named fpminimax() can also compute approximation polynomials with accuracy close to remez() with the additional feature of imposing predefined widths on the coefficients at the expense that $\hat{m}^\circ \leq w$. Thus, fpminimax() is not convenient for our value of $m^\circ = 60$. Furthermore, the resulting coefficients are expected to be stored within higher precisions than $w = 53$ for our n values of interest. We chose the lowest precision that makes the remez() produce a valid output and performed a rounding $a_j = \mathrm{RN}_w(a'_j)$ for $j = 0, 1, \ldots, n$ where a'_j is the original pre-rounded coefficient produced by Sollya. Thus, this operation induces a round-off error that affects the total error.

The evaluation of P_n° is carried out using our assumption of the single non-pipelined FMA using the direct Horner's method (see Section 3.4) and is implemented as follows:

$$\begin{aligned} t &\Leftarrow \mathrm{FMA}\,(x_r, a_n, a_{n-1}) \\ t &\Leftarrow \mathrm{FMA}\,(x_r, t, a_{n-2}) \\ t &\Leftarrow \mathrm{FMA}\,(x_r, t, a_{n-3}) \\ &\vdots \\ P_n^\circ(x_r) &\Leftarrow \mathrm{FMA}\,(x_r, t, a_0) \end{aligned}$$

Therefore, P_n° requires n cycles to get evaluated. Also note that $x_r = x$ for $E_x < 1$. We use RN mode for all n cycles. Thus, we expect an additional round-off error ensued by these n FMA operations that affects the overall error.

As we seek to evaluate $P_n^\circ(x_r)$ depending on the value of i and the designated function $\hat{f}_p^\circ$, we do not need to store the coefficients of both functions. We can only store P_n^s coefficients and obtain $P_n^c(x_r)$ using the relation:

$$P_n^c(x_r) = P_n^s(x'_r) \tag{5.15}$$

where $x'_r = \pi/2 - x_r$ which is implemented as:

$$x'_r \Leftarrow \mathrm{FMA}(K'_1, 1, -x_r) \tag{5.16}$$

We should point out that Equation 5.16 may be extended to be more accurate as:

$$x'_r \Leftarrow \mathrm{FMA}(K'_2, 1, -x'_r) \tag{5.17}$$

This accuracy extension is actually useful for very rare cases as will be shown later using exhaustive testing.

However, for $x_r \in [\pi/2, 2]$ we obtain a negative value of x'_r, which happens not only for $\pi/2 < x < 2$ but also for $E_x \geq 30$ where we expect x_r to possibly exceed $\pi/2$. Therefore, Equation 5.15 is reformulated as follows:

$\circ$	Input	Output	Exception
s	$+0$	$+0$	None
	-0	-0	
	$\lvert x\rvert < 2^{E_{\min}}$	x	Underflow
c	± 0	1	None
both	$\lvert x\rvert = \text{Inf}$	NaN	Invalid
	sNaN		
	Otherwise	$\hat{f}_p^{\circ}(x)$	None [1]

Table 5.3: Outputs and flags for the different arguments

$$P_n^c(x_r) = (-1)^{S_{x'_r}} P_n^s(|x'_r|) \tag{5.18}$$

The negation and modulus operations just affect the sign section and therefore they can be handled by the control unit with no extra latency incurred. Thus, we can just evaluate $P_n^s(|x'_r|)$ and defer the multiplication by $(-1)^{S_{x'_r}}$ to the reconstruction step as will be shown.

5.3.4 Reconstruction and Final Rounding

The reconstruction step is implicit and lossless as only the sign bit of the designated result is affected. Therefore, this mere sign flipping operation can simply involve the quantity $(-1)^{S_{x'_r}}$ as follows:

$$\hat{f}_w^s(x) = \begin{cases} P_n^s(x_r) & \text{if } -12 \le E_x < 1 \\ P_n^s(x_r) & \text{if } 1 \le E_x < 53 \text{ and } i = 0 \\ (-1)^{S_{x'_r}} P_n^s(|x'_r|) & \text{if } 1 \le E_x < 53 \text{ and } i = 1 \\ -P_n^s(x_r) & \text{if } 1 \le E_x < 53 \text{ and } i = 2 \\ -(-1)^{S_{x'_r}} P_n^s(|x'_r|) & \text{if } 1 \le E_x < 53 \text{ and } i = 3 \end{cases} \tag{5.19}$$

$$\hat{f}_w^c(x) = \begin{cases} (-1)^{S_{x'_r}} P_n^s(|x'_r|) & \text{if } -12 \le E_x < 1 \\ (-1)^{S_{x'_r}} P_n^s(|x'_r|) & \text{if } 1 \le E_x < 53 \text{ and } i = 0 \\ -P_n^s(x_r) & \text{if } 1 \le E_x < 53 \text{ and } i = 1 \\ -(-1)^{S_{x'_r}} P_n^s(|x'_r|) & \text{if } 1 \le E_x < 53 \text{ and } i = 2 \\ P_n^s(x_r) & \text{if } 1 \le E_x < 53 \text{ and } i = 3 \end{cases} \tag{5.20}$$

We assume a separate final cycle dedicated for the reconstruction and final rounding shown in Equation 5.6 as well as signaling the convenient flags (see Table 5.3).

[1] IEEE-754-2008 recommends not to raise the inexact flag as it is the expected flag for almost all arguments.

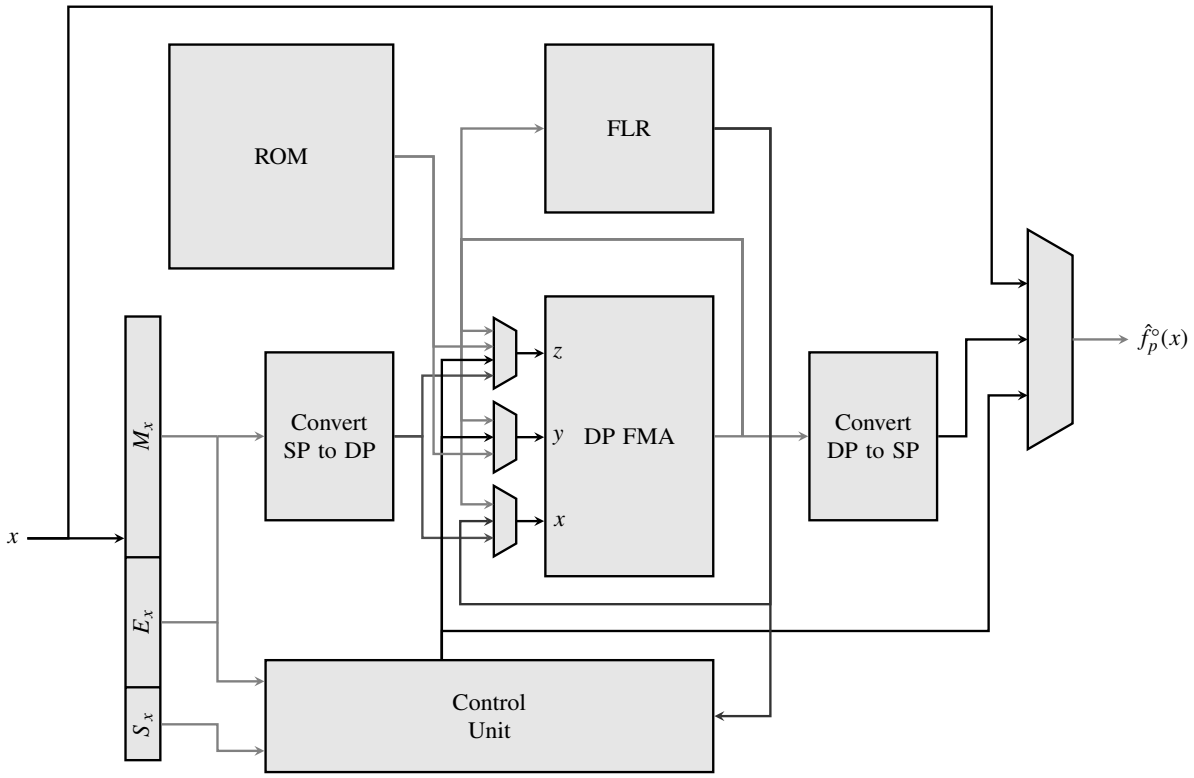


Figure 5.3: A description for the architecture

5.3.5 Memory and Latency Requirements

Figure 5.3 shows a description of the proposed architecture. Control signals and registers are omitted in order to simplify the illustration.

The total latency represented by the number of cycles τ^s and τ^c for computing $\hat{f}_p^s(x)$ and $\hat{f}_p^c(x)$ respectively can be calculated as the summation of $1+\tau_1+\tau_2+\tau_3$ where the first cycle is assumed to be mandatory and it is dedicated for converting x into double-precision as E_x needs to be encoded to DP, resolving E_x, fetching the first memory element as well as handling small arguments and special cases. The value $\tau_1 = 4$ represents the latency required by the range reduction, $\tau_2 = 1$ represents Equation 5.16) and may be extended to 2 cycles if Equation 5.17 is used, and $\tau_3 = n+1$ represents the n cycles required by the evaluation of P_n° plus the final cycle required by the reconstruction and final rounding.

$$\tau^s = \begin{cases} 1+\tau_3 & \text{if } -12 \le E_x < 1 \\ 1+\tau_1+\tau_2^*+\tau_3 & \text{if } 1 \le E_x < 53 \\ 1 & \text{Otherwise} \end{cases} \tag{5.21}$$

$$\tau^c = \begin{cases} 1+\tau_2+\tau_3 & \text{if } -12 \le E_x < 1 \\ 1+\tau_1+\tau_2^*+\tau_3 & \text{if } 1 \le E_x < 53 \\ 1 & \text{Otherwise} \end{cases} \tag{5.22}$$

Note that τ_2^* means that τ_2 exists only if P^c is required depending on the value of i and the designated function $\circ$. The "Otherwise" case involves $E_x < -12$ including denormals, $E_x \ge 53$, other special cases: zero, infinity, NaN arguments. For $53 \le E_x \le 127$,

n	$\mathrm{FT}^s_{\Psi_1}$	$\mathrm{FT}^s_{\Psi_2}$	$\mathrm{FT}^c_{\Psi_1}$	$\mathrm{FT}^c_{\Psi_2}$
14	15	2	0	5
15	1	2	1	3
16	0	2	0	2
17	0	1	0	2

Table 5.4: Number of faithful cases FT° at $n = 14, 15, 16, 17$

we recommend to raise an ad-hoc exception signal with NaN as a default output. The exception can be used to trigger a special-purpose circuitry or a software routine in order to reduce the argument using Payne and Hanek's algorithm (see Section 3.1.2) and then the values of x_r and i are fed back to the architecture. Also note that Equation 5.19 and 5.20 are valid for positive arguments; for negative arguments, the sign must be stripped at the first cycle for both functions and $f^s_w(x)$ is negated at the last cycle.

Preliminary and pessimistic memory requirements are directly calculated as $(n+1+3)\times 64$ bits divided as follows: $n+1$ coefficients and 3 for storing the values of K'_1, K'_2 and $1/b$. The floating-point representations of zero and one are required in SP and DP for the following situations: producing the output at zero inputs and $\hat{f}^c_p(x)$ when $E_x < -12$ for the SP case, and implementing Equation 5.11 and 5.16 for the DP case. We assume that these direct values can be signaled by the control unit. Thus, we have a memory size range of 1216 to 1344 bits for $n = 15$ to 17. Later, we will see that this preliminary memory calculation is much larger than we really need, and that the total memory size can be reduced to less than 700 bits using a simple heuristic memory reduction algorithm.

5.3.6 Exhaustive Testing Results

We test our method exhaustively by comparing $\hat{f}^\circ_p(x)$ with the expected $f^\circ_p(x)$ within Ψ. We define $\Psi_1 = [2^{-12}, 2^1]$ and $\Psi_2 = [2^1, 2^{53}]$. We also define FT° to represent the number of occurrences of faithful cases, where the output has an error of 1 ulp, within the corresponding interval. Table 5.4 shows the results when Equation 5.17 is applied. We have no cases having error more than 1 ulp within Ψ for our n values of interest. If Equation 5.17 is ignored we find an increase for the entire range Ψ of $8, 7, 2, 4$ for FT^s_Ψ and $1, 2, 0, 1$ for FT^c_Ψ for $n = 14, 15, 16, 17$ respectively. Therefore, the extension of Equation 5.17 may be considered unnecessary in order to reduce the entire latency by a cycle. It may be useful to point out that for the $n = 16$ case whose polynomial is shown in Table 5.5, the least of all those 4 faithful cases for both functions is larger than 2^{13}. Those 4 exceptions may be put in a table to be accessed in the first cycle in order to obtain 100% correct rounding within the whole range.

To understand how the final result $\hat{f}^\circ_p$ is obtained correctly rounded, we compare $\hat{f}^s_w(x)$ and $\hat{f}^c_w(x)$ with $f^s_w(x)$ and $f^c_w(x)$ (i.e. the expected results using MPFR) respectively for $x \in \Psi_1$. We define ζ°, a positive integer that represents the difference in ulps with respect to DP between $\hat{f}^\circ_w(x)$ and $f^\circ_w(x)$ where:

j	a_j
0	$1.10110000100110010111100101110100110111101111011111011\times2^{-66}$
1	$1.00000000000000000000000000000000000000000000000000000\times2^{0}$
2	$1.01110101001111001000111101101011010010000101010000101\times2^{-52}$
3	$-1.01010101010101010101010101010101010101010110001101 01\times2^{-3}$
4	$1.00011010100100000001100111000001000100010001010000011\times2^{-44}$
5	$1.00010001000100010001000100010000110110101110101011 01\times2^{-7}$
6	$1.10110110010001010100111101011101100111011111001110 01\times2^{-40}$
7	$-1.10100000000110100000001000111010011000010100010110 00\times2^{-13}$
8	$1.00111001000010001111100101110111110011110111100101 11\times2^{-37}$
9	$1.01110001110111011100010101010100110001111010110111 11\times2^{-19}$
10	$1.00000101001011011011000110001000010011101001001100 0\times2^{-36}$
11	$-1.10101110100110110000010001100010100011110010010111 01\times2^{-26}$
12	$1.00010010000001111010101001101011101011001101111011 00\times2^{-37}$
13	$1.01011001001101101101110111110011001101100001110000 01\times2^{-33}$
14	$1.01010010011010001110011001011011011001111100011 0011\times2^{-40}$
15	$-1.00100100111010100100011111100001110101100101000110000\times2^{-40}$
16	$1.01100100101010110101011011110000111111000100000000 10\times2^{-45}$

Table 5.5: coefficients of P_{16}

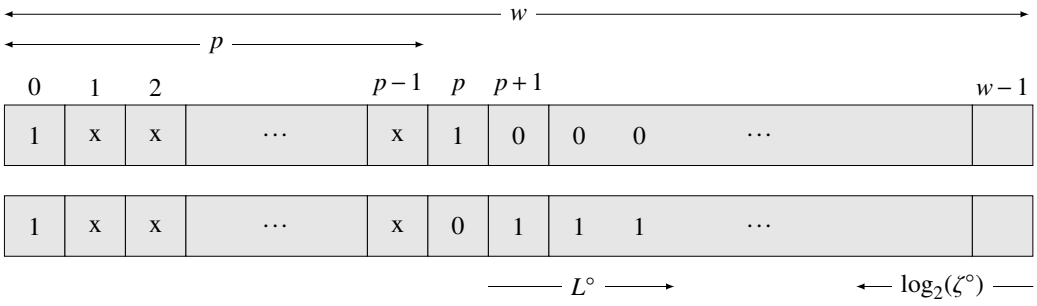


Figure 5.4: L° vs $\log_2(\zeta^\circ(x))$

$$\zeta^s = \frac{|f_w^s(x) - \hat{f}_w^s(x)|}{2^{-(w-1)+E_{f_w^s(x)}}} \tag{5.23}$$

and

$$\zeta^c = \frac{|f_w^c(x) - \hat{f}_w^c(x)|}{2^{-(w-1)+E_{f_w^c(x)}}} \tag{5.24}$$

Note that we assume that $E_{\hat{f}_w^\circ(x)} = E_{f_w^\circ(x)}$ which may not be guaranteed in very rare cases yet this does not affect our argument. While $\log_2(\zeta^\circ(x))$ can roughly represent the number of corrupt bits starting from the most significant position $w-1$ backwardly, our architecture decisions should maintain that $\log_2(\zeta^\circ(x))$ is not large enough so as to not affect L° for every argument within Ψ (see Figure 5.4).

Nevertheless, we are more interested in studying the change of the maximum of ζ over C', which indicates to a potential incorrectly rounded final result f_p°, faithfully rounded for our case. Therefore, we define ζ'°, a maximum value of all ζ° values over a fixed tight range of consecutive FPNs.

n	$\widetilde{\text{CR}}$	$\widetilde{\text{FT}}$	$\widetilde{\text{ER}}$
13	0.71%	1.42%	97.87%
14	11.33%	25.54%	63.13%
15	32.43%	36.47%	31.10%
16	76.58%	23.20%	0.22%
17	78.78%	20.74%	0.48%

Table 5.6: Accuracy of $\hat{f}_w^s(x)$ for $x \in \Psi_1$

For the case of f^s as shown in Figure 5.5, we can easily detect the relation between $\zeta'^s(x)$ and $\hat{m}^s$ for each n. Moreover, we can relate the difference between two different $\log_2(\zeta'^s(x))$ for two different n values at some fixed $\log_2(x)$ value. To understand the graphs more clearly, for example, Let us discuss the case for $n = 13$. For x values having $E_x = -12$ at $n = 13$ we expect that the mantissa components can be as small as $2^{-12-52} = 2^{-64}$ while our $|\epsilon_{\text{approx}}| < 2^{-49}$ and therefore we expect ζ^s can be up to 2^{15}. As $\hat{m}$ increases by 4 or 6 as n gets incremented, $\log_2(\zeta'^s)$ is reduced roughly by the same values at some x as we increment n. This is more apparent at low $\log_2(x)$ values. For $n = 16$ and 17 we see that $\log_2(\zeta'^s)$ is always 0; that is, we have faithful results with respect to $f_w^s(x)$, from $\log_2(x) = -12$ until almost the end of the interval, i.e. towards $\log_2(x) \approx 0$. As we may expect, almost all erroneous cases for $n = 13, 14, 15$ happen to lie at small values of x, i.e. roughly within $\log_2(x) \in [-12, -9]$. For $n = 16, 17$ we have 100% correctly rounded results for $f_p^s(x)$.

Actually, we found that a great share of the total cases of $\hat{f}_w^s$ for $x \in \Psi_1$ are correctly rounded with respect to $\hat{f}_w^s$ as n increases. Table 5.6 shows the share of correctly rounded ($\widetilde{\text{CR}}$), faithful ($\widetilde{\text{FT}}$) and erroneous or cases having error ≥ 2 ulps ($\widetilde{\text{ER}}$) for $n = 13, 14, 15, 16, 17$ for $\hat{f}_w^s$ within Ψ_1.

For the case of f^c shown in Figure 5.6, we find that due to the advantageous non-uniform distribution of x'_r, the value of $\log_2(\zeta'^c)$ is guaranteed to be small unless $x \approx \pi/2$. Thus, we expect the existence possibility of erroneous cases to lie when $\log_2(x) \in [0, 1]$ especially within the very narrow range around $x \approx \pi/2$.

The figures 5.5 and 5.6 indicate that the pre-rounded result is accurate enough to be easily rounded to any of the other three rounding modes within Ψ. Although using RN mode especially at the last cycle of the polynomial evaluation may not be the optimal choice for the directed rounding modes, FT occurrences are still only a few cases within Ψ. For $n = 16$ and using the same $\hat{f}_w^\circ(x)$ obtained for RN, we have $\text{FT}^\circ_{\Psi_1} = 0$ for RZ, RU and RD; while $\text{FT}^s_{\Psi_2}$ values are of $7, 4, 3$ and $\text{FT}^c_{\Psi_2}$ values of $5, 5, 1$ for RZ, RU and RD respectively when Equation 5.17 is applied.

5.3.7 A Heuristic Algorithm for Memory Reduction

This simple yet effective heuristic algorithm is based on the observation that the coefficients having lower values contribute less to the final result $\hat{f}_p^\circ(x)$. Note that while we have

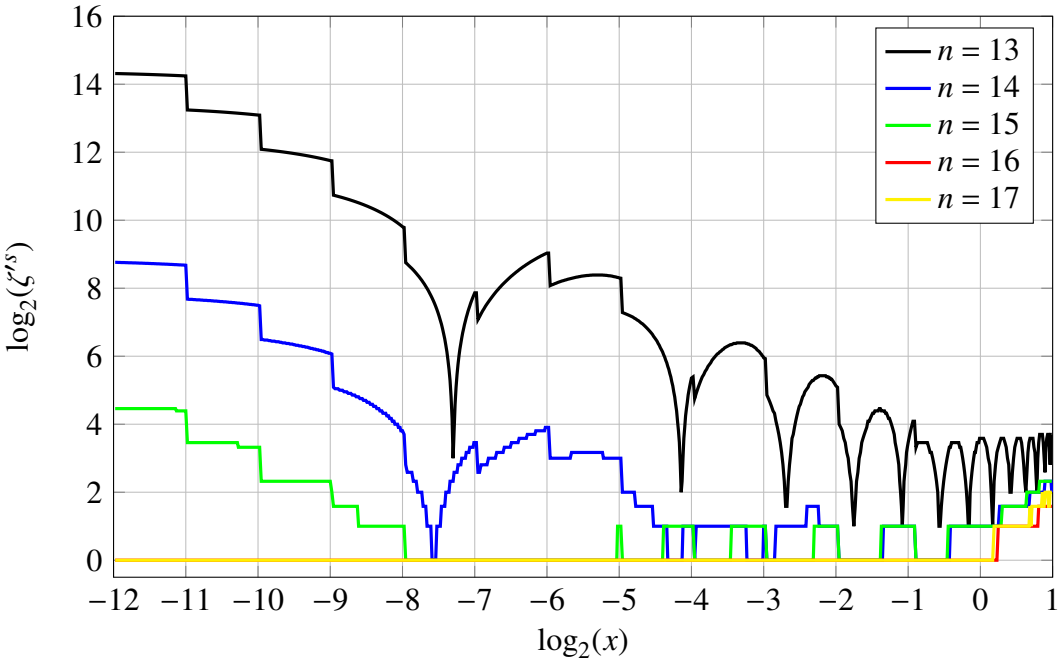


Figure 5.5: $\log_2(\zeta'^s(x))$ vs $\log_2(x)$ for $x \in \Psi_1$

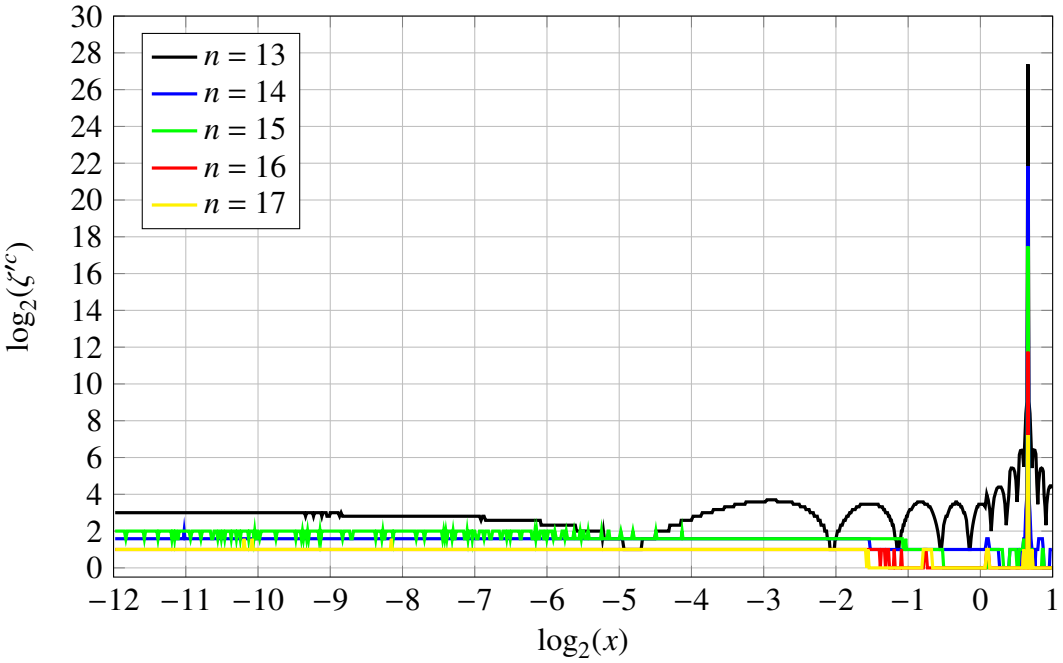


Figure 5.6: $\log_2(\zeta'^c(x))$ vs $\log_2(x)$ for $x \in \Psi_1$

$|\hat{f}_p^\circ(x)| \leq 1$ and P_n° is only accurate to $2^{-\hat{m}^s}$, the values of $E_{a'_j}$ can be as low as -66 for $n = 16$ and -70 for $n = 17$ and as high as 0 for all n values of interest. Therefore, having all coefficients exactly represented in DP may be unnecessary. We can then relate the width of the mantissa section to $|a'_j|$, or more precisely to $E_{a'_j}$.

Algorithm 1: Heuristic algorithm for memory reduction

```
for j = 0, 1, 2, ..., n do
    if E_{a'_j} ≤ −(w + θ − 1) then
        a_j ← 0;
    else if E_{a'_j} > −(w + θ − 1) and E_{a'_j} < −θ then
        a_j ← RN_{w'}(a'_j);
    else
        a_j ← RN_w(a'_j);
end
```

As shown in Algorithm 1, we define a positive integer bias θ, where if $E_{a'_j} \geq -\theta$ we round the coefficient normally as before to DP. For lower values of $E_{a'_j}$, a'_j is rounded to a precision w' as follows:

$$w'(E_{a'_j}) = w + \theta + E_{a'_j} \tag{5.25}$$

If $|a'_j|$ is small enough such that $E_{a'_j} \leq -(w + \theta - 1)$, we force it to equal zero. The algorithm simply intends to make the accuracy of each coefficient as small as $2^{-(w+\theta-1)}$. We can start with some small θ value and perform an exhaustive testing to decide whether it gives acceptable results with respect to the original rounding. If it does not, we increase the value of θ and re-perform the exhaustive testing and so forth until we obtain an acceptable result with respect to the original rounding.

The algorithm also indicates that the convenient values of θ, as will be shown in Table 5.7, imply that storing the exponent section in 11 bits according to the standard DP encoding is utterly useless as all coefficients, as well as K'_1, K'_2 and $1/b$, share a much narrower range of exponent representations which can fit to 6 bits for our n values of interest. Thus, we can only store the least 6 significant bits as follows: the number zero is represented by 000000 while E_{a_j} values from -62 to 0 are represented by 000001 to 111111. By appending the 5 most significant bits as 01111 for this set of E_{a_j} values, we obtain correctly represented exponents in the standard DP exponent encoding.

Table 5.7 shows some results using the algorithm while Equation 5.17 is applied. MEM_P and MEM_T represent the polynomial memory and total memory (including the stored values of K'_1, K'_2 and $1/b$) in bits respectively. It is not surprising to obtain a lower memory for $n = 16$ than $n = 15$ at the same θ value as we have more coefficients having lower absolute values when n is higher and subsequently the algorithm becomes able to round to lower values of w'. Moreover, it may happen that when we increment θ, we may find that FT° increases for the same n value.

5.3.8 Using Piece-wise Polynomials

Although the main philosophy of the thesis is to provide a single polynomial over a wide range to make the memory requirements feasible at the expense of using higher degrees,

n	θ	MEM_P	MEM_T	$\text{FT}^s_{\Psi_1}$	$\text{FT}^s_{\Psi_2}$	$\text{FT}^c_{\Psi_1}$	$\text{FT}^c_{\Psi_2}$
16	8	532	708	0	1	0	2
16	7	514	690	1	1	1	3
15	8	582	758	0	2	0	3
17	7	504	680	0	2	0	2

Table 5.7: Examples of faithful case occurrences FT° at different θ and n values

we can divide the original interval C' into a few sub-intervals, each requiring a lower degree than the original n value in order to reduce the average and worst case latencies while still keeping the memory at reasonable size.

While there is no restriction for choosing the number of piecewise polynomials and the width of each sub-interval, we can choose them so as to simplify the implementation without incurring considerable logic and memory overhead. We may use a set of polynomials depending on the value of E_x. Thus, we expect unequally sized sub-intervals and subsequently different required values of n for each. The resolution of E_x can be carried out fast due to having a very small number of segments.

For example, we may divide C' into 4 sub-intervals $\beta_1 = [0, 2^{-5}]$, $\beta_2 = [2^{-5}, 2^{-2}]$, $\beta_3 = [2^{-2}, 2^0]$ and $\beta_4 = [2^0, 2^1]$, where $n_{\beta_1} = 6$, $n_{\beta_2} = 8$, $n_{\beta_3} = 10$ and $n_{\beta_4} = 11$. That is, P° is interpreted as: $P^\circ_{\beta_1}$ for $E < -5$, $P^\circ_{\beta_2}$ for $-5 \le E < -2$, $P^\circ_{\beta_2 3}$ for $-2 \le E < 0$ and $P^\circ_{\beta_4}$ for $E = 0$. This configuration produces, when Equation 5.17 is applied, $\text{FT}^s_\Psi = 4$ and $\text{FT}^c_\Psi = 3$ with preliminary non-reduced 2688 bits of memory.

There exist many other possible but more expensive and complicated techniques to reduce the latency or increase the throughput such as using a pipelined FMA with Estrin's method (see Section 3.4), using cascaded FMAs or modifying the FMA internally to support the final rounding and the FLR operations.

5.3.9 Applying The Method for The Double-Precision Case

There have been already investigations, thanks to Lefèvre's algorithm, to find the hardest-to-round cases for several elementary functions within certain intervals (see Section 2.3.2). The results of L^s_* and L^c_* given in Table 2.3 indicate that the standard QP may not be preliminarily sufficient for all cases. However, for the DP case, we do not hope to produce only a few faithful cases out of the whole range as we have in SP, but one faithful case in a billion cases may be a satisfactory proportion.

We argue that this method may be extended to compute correctly rounded double-precision results. The only changes are: $p = 53$, $w = 113$; this translates into a QP FMA and using convenient degree such as $n = 28$ with corresponding $\hat{m}^s = 131$ to compute correctly rounded results for argument values $|x| < 2^{113}$. We generated 6 billion random cases within the range $[2^{-25}, 2^{113}]$ and found 100% correctly rounded cases for both functions. Although this number of random test cases is still very small compared to the total designated range, the test indicates that the method is feasible to be successfully applied to the DP case. Thus, we may be able to compute correctly rounded double-

precision trigonometric function for an enormous range of arguments with less than 4kbits of memory and within a latency of 30 to 36 cycles according to our architecture assumptions.

5.A Coefficients of Polynomials After Memory Reduction

In this appendix we present the 4 polynomials shown in Table 5.7 after applying the memory reduction algorithm.

j	a_j
0	0.0×2^{0}
1	1.0×2^{0}
2	1.01110101×2^{-52}
3	$-1.0101010101010101010101010101010101010101011000110101\times2^{-3}$
4	$1.000110101001\times2^{-44}$
5	$1.000100010001000100010001000100001101101011101010110 1\times2^{-7}$
6	$1.10110110010001010101\times2^{-40}$
7	$-1.10100000000110100000001000111010011000010100011\times2^{-13}$
8	$1.00111001000010001111101\times2^{-37}$
9	$1.01110001110111011100010101010100110001111\times2^{-19}$
10	$1.00000101001011011011100 1\times2^{-36}$
11	$-1.10101110100110110000010001100010 1\times2^{-26}$
12	$1.00010010000001111010101\times2^{-37}$
13	$1.01011001001101101101111\times2^{-33}$
14	$1.0101001001101000111\times2^{-40}$
15	$-1.00100100110101001001\times2^{-40}$
16	$1.01100100101011\times2^{-45}$

Table 5.8: Coefficients of P_{16} with $\theta = 8$

j	a_j
0	0.0×2^{0}
1	1.0×2^{0}
2	1.01×2^{-56}
3	$-1.0101010101010101010101010101010101010101010101010110001\times2^{-3}$
4	1.001001011×2^{-48}
5	$1.000100010001000100010001000100010000110100101001010111\times2^{-7}$
6	$1.0001100110000011\times2^{-43}$
7	$-1.10100000000110100000000110101101111110000000101\times2^{-13}$
8	$1.11110111110010010 1\times2^{-41}$
9	$1.01110001110111100010110100010001011001 01\times2^{-19}$
10	$1.00001100101101110011\times2^{-39}$
11	$-1.10101110011011000110000010001101\times2^{-26}$
12	$1.01110100000111101 11\times2^{-40}$
13	$1.01011111100100011010001001\times2^{-33}$
14	$1.001111110011\times2^{-42}$
15	$-1.11011010111011\times2^{-41}$
16	$1.111111101111\times2^{-47}$
17	1.101011111×2^{-50}

Table 5.9: Coefficients of P_{17} with $\theta = 8$

j	a_j
0	0.0×2^{0}
1	1.0×2^{0}
2	1.0111011×2^{-52}
3	$-1.0101010101010101010101010101010101010101011000110101\times 2^{-3}$
4	$1.000110101001\times 2^{-44}$
5	$1.0001000100010001000100010001000011011010111010101101\times 2^{-7}$
6	$1.101101100100010101\times 2^{-40}$
7	$-1.101000000001101000000010001110100110000101000 1\times 2^{-13}$
8	$1.001110010000100011111\times 2^{-37}$
9	$1.011100011101110111000101010101010011001\times 2^{-19}$
10	$1.000001010010110110111\times 2^{-36}$
11	$-1.101011101001101100000100011000101\times 2^{-26}$
12	$1.000100100000011110101 1\times 2^{-37}$
13	$1.010110010011011011011111\times 2^{-33}$
14	$1.01010010011010001 11\times 2^{-40}$
15	$-1.001001001101010 01\times 2^{-40}$
16	$1.011001001010 11\times 2^{-45}$

Table 5.10: Coefficients of P_{16} with $\theta = 7$

j	a_j
0	0.0×2^{0}
1	$1.000000000000000000000000000000000000000000000000000001\times 2^{0}$
2	$-1.1101010101001\times 2^{-47}$
3	$-1.0101010101010101010101010101010101010011011010110101\times 2^{-3}$
4	$-1.000011001001010011 11\times 2^{-39}$
5	$1.000100010001000100010001000101101010000000101 1101101\times 2^{-7}$
6	$-1.001101010000001100001001\times 2^{-35}$
7	$-1.101000000001100111110110000011011000111101 1101\times 2^{-13}$
8	$-1.001111000001011011010101 11\times 2^{-33}$
9	$1.011100011110010001011000011101000010 01\times 2^{-19}$
10	$-1.011000101000110011001000001\times 2^{-33}$
11	$-1.101011001001000010000000000110111\times 2^{-26}$
12	$-1.101110100011111001101010 1\times 2^{-35}$
13	$1.100001010010111101100111\times 2^{-33}$
14	$-1.110111100000110100 01\times 2^{-39}$
15	$-1.110010011000111 11\times 2^{-42}$

Table 5.11: Coefficients of P_{15} with $\theta = 8$

5.B Applying the Method for 2^x and $\log_2(x)$

As Chapter 2 tells us that $m_*^\circ \approx p + \log_2(\lambda)$ (see Section 2.3.2), we can expect having theoretically similar results to those obtained for trigonometric functions using the same architecture for other elementary functions. Note that our decision of using a working precision of DP cannot practically guarantee correct rounding for all arguments or similar results to those obtained for the case of trigonometric functions. The problem lies primarily within the round-off errors. The effect of the round-off errors may greatly differ from a function to another due to the values of the coefficients, the degree of the polynomial, the intervals of the arguments and the expected outputs.

In this appendix we give the results obtained for 2^x for $x \in [0,1]$ and $\log_2(x)$ for $x \in [2,4]$. As is the case with the trigonometric functions, we conduct an exhaustive search to find the hardest-to-round cases and then find the polynomial having the required accuracy with assuming the same architecture used for the case of trigonometric functions. Table 2.1 shows the relation between L° and $\Gamma(L^\circ)$ for $\log_2(x)$ and 2^x for the designated reduced intervals. We denote the largest value of L_*° for 2^x and $\log_2(x)$ as L_*^e and L_*^l respectively. As shown in Table 2.1, the values of L_*^e and L_*^l are 26 and 29 respectively. A similar exhaustive search is conducted to find m_*^e and m_*^l and they are found to be 51 and 54 respectively. Note as the output range for the two functions within the designated intervals is $[1,2]$, the values of m_*^e and m_*^l are exactly $L_*^e + p + 1$ and $L_*^l + p + 1$ respectively (see Equation 5.7) as $E_x = 0$ for either case.

For 2^x:

1. Range reduction: compute x_r as follows:

$$x_r = \begin{cases} x & \text{if } x \in [0,1] \\ x - \lfloor x \rfloor & \text{Otherwise} \end{cases} \tag{5.26}$$

2. Evaluation: Evaluate $P^e(x)$ for $x \in [0,1]$

3. Reconstruction:

$$2^x = \begin{cases} 1 & \text{if } |x| < 2^{-25} \\ P^e(x_r) & \text{if } x \in [2^{-25}, 1] \\ 2^{\lfloor x \rfloor} P^e(x_r) & \text{if } x \in \left[1, (2 - 2^{-23}) \times 2^6\right] \\ \infty & \text{if } x \geq 2^7 \end{cases} \tag{5.27}$$

For $\log_2(x)$:

1. Range reduction:

$$x_r = \begin{cases} x & \text{if } x \in [2,4] \\ 2 \times 1.M_x & \text{Otherwise} \end{cases} \tag{5.28}$$

2. Evaluation: Evaluate $P^l(x_r)$.

3. Reconstruction:

$$\log_2(x) = \begin{cases} P^l(x_r) & \text{if } x \in [2,4] \\ E_x - 1 + P^l(x_r) & \text{Otherwise} \end{cases} \tag{5.29}$$

$f(x)$	C	L°	m_*°	n	MEM_P	$\hat{m}^\circ$	FT_C°
2^x	$[0,1]$	29	54	10	492	51	1
$\log_2(x)$	$[2,4]$	26	51	15	858	43	10

Table 5.12: The values for certain elementary functions for the RN case

j	a_j
0	$1.00000000000000000000000000000000000000000000000000001 \times 2^{0}$
1	$1.01100010111001000010111111101111101000110111010 1 \times 2^{-1}$
2	$1.11101011111110111101111111111000010001100000100 11 \times 2^{-3}$
3	$1.11000110101100001000110101101010000110100 01 \times 2^{-5}$
4	$1.00111011001010101011011101111111011011011 \times 2^{-7}$
5	$1.01011101100001111110001000100110011101111111 \times 2^{-10}$
6	$1.01000011000010111001111100000110001101 01 \times 2^{-13}$
7	$1.11111111100000001101111111110001101 \times 2^{-17}$
8	$1.01100101100000111010101110001 01 \times 2^{-20}$
9	$1.10010101001011100101000101 11 \times 2^{-24}$
10	$1.01010111110100110111110001 \times 2^{-27}$

Table 5.13: Coefficients of P^e

The polynomials of both functions were obtained using Sollya as usual. However, the function fpminimax() was used to obtain both polynomials. Table 5.12 shows the values of $\hat{m}^\circ$, n and FT_C° for both functions within the designated intervals.

j	a_j
0	$-1.10100101000101101100111011101011001001001100111101 01\times 2^{1}$
1	$1.11101001100011000011100110100101001110110011111101101\times 2^{2}$
2	$-1.00101101100110001011100000101010011111000101011111\times 2^{3}$
3	$1.00110001011111010100100001100100100111010010111001111\times 2^{3}$
4	$-1.11100000000001111100110010110110110100001111001110111\times 2^{2}$
5	$1.00100101110111011101110011010101010001100110011111101\times 2^{2}$
6	$-1.00011010110001111011001111010111111010111111011000011\times 2^{1}$
7	$1.10101110000110101111111001011100000101111001110111111\times 2^{-1}$
8	$-1.00000010101011010101010000111100011101101000101111\times 2^{-2}$
9	$1.11101001110000100110100010101000011010110111101\times 2^{-5}$
10	$-1.01101000101101111010010110011000101001110001\times 2^{-7}$
11	$1.10010101000001101101111010001000001001111111\times 2^{-10}$
12	$-1.01001110111111010110111000111111101101101\times 2^{-13}$
13	$1.10000000110101100010011101101001101\times 2^{-17}$
14	$-1.00010010010001010100101011111011\times 2^{-21}$
15	$1.01101101011000010100111001\times 2^{-27}$

Table 5.14: Coefficients of P^l

Chapter 6: Conclusion and Future Work

A table lookup-less method for computing correctly rounded IEEE-754 SP elementary functions has been presented. The method uses a single large-degree minimax approximation polynomial valid over a wide single interval. A standard DP FMA is used to carry out the polynomial evaluation with the help of a simple control unit. Additional simple logic is integrated to support the range reduction and carry out the implicit reconstruction and final rounding steps. The method was applied for trigonometric functions within $|x| < 2^{53}$ with consuming memory of 708 bits using a degree-16 polynomial after reducing the initial memory using a simple heuristic algorithm. We have also argued that the method can be applied for DP trigonometric functions. Moreover, the method was applied for 2^x for $x \in [0,1]$ and $\log_2(x)$ for $x \in [2,4]$ having polynomial degrees of 10 and 15 with consuming memory of 492 and 858 bits respectively. The method can be also extended to compute other elementary functions with similar accuracy results using the same architecture.

The method requires very small memory requirements, usually hundreds of bits compared to hundreds of kbits for degree-2 table-based methods, and uses the general-purpose DP FMA to compute the final result, as opposed to the special-purpose circuitry (special multipliers, ad-hoc powering units and multi-operand adders) used by the table-based methods. Nevertheless, the latency of the method is much higher compared to the table-based methods due to the large-degree polynomials used. To mitigate this problem, we have proposed using a small number of piece-wise polynomials to reduce the average and worst-case latencies. For the case of trigonometric functions, we have applied unequally sized 4 sub-intervals for the initial reduced interval $[0,2]$ and have had a total non-reduced memory of 2688 bits with polynomial degrees of $6, 8, 10$ and 11.

The method needs a more rigorous error analysis in order to be generalized for arbitrary target precisions and elementary functions. As seen in Chapter 5, the round-off error was not the governing type of error for the case of trigonometric functions. However, this may not be the case for other elementary functions. Thus, the required working precision of the FMA may be larger or slightly smaller than DP for certain other elementary functions. The implementations of single wide polynomial vs several piece-wise polynomials need also to be compared in terms of area-latency trade-off. Furthermore, the practical implementations of FMA such as using pipelining can be explored with using more advanced polynomial evaluation techniques such as Estrin's method.

Although the method offers a substantial reduction of memory compared to the table-based methods, applying the method to obtain a lower accuracy such as the faithful accuracy is considered generally unwise depending on the designated function and reduced interval. While degree-2 piecewise polynomials usually produce faithfully rounded results with lookup-tables of several kbits, reducing the polynomial degree using our method to achieve the faithful accuracy may not be considerable. Thus, we may still end up having large latency due to the large-degree polynomial while having a not very significant memory reduction.

References

[1] Baker, A. *Transcendental number theory*. Cambridge University Press, 1990.

[2] Brent, R. P., and Zimmermann, P. *Modern computer arithmetic*, vol. 18. Cambridge University Press, 2010.

[3] Chevillard, S., Joldeş, M., and Lauter, C. Sollya: An environment for the development of numerical codes. 28–31.

[4] Cody, W. J., and Waite, W. M. *Software manual for the elementary functions*. Prentice-Hall, 1980.

[5] Daramy-Loirat, C., Defour, D., de Dinechin, F., Gallet, M., Gast, N., Lauter, C. Q., and Muller, J.-M. Cr-libm a library of correctly rounded elementary functions in double-precision, 2009.

[6] Das Sarma, D., and Matula, D. W. Faithful bipartite ROM reciprocal tables. In *Computer Arithmetic, 1995., Proceedings of the 12th Symposium on* (1995), IEEE, pp. 17–28.

[7] De Dinechin, F., and Tisserand, A. Multipartite table methods. *Computers, IEEE Transactions on 54*, 3 (2005), 319–330.

[8] Estrin, G. Organization of computer systems: the fixed plus variable structure computer. In *Papers presented at the May 3-5, 1960, western joint IRE-AIEE-ACM computer conference* (1960), ACM, pp. 33–40.

[9] Fousse, L., Hanrot, G., Lefèvre, V., Pélissier, P., and Zimmermann, P. MPFR: A multiple-precision binary floating-point library with correct rounding. *ACM Transactions on Mathematical Software (TOMS) 33*, 2 (2007), 13.

[10] Gal, S. An accurate elementary mathematical library for the IEEE floating point standard. *ACM Transactions on Mathematical Software (TOMS) 17*, 1 (1991), 26–45.

[11] Iordache, C., and Matula, D. W. On infinitely precise rounding for division, square root, reciprocal and square root reciprocal. In *Computer Arithmetic, 1999. Proceedings. 14th IEEE Symposium on* (1999), IEEE, pp. 233–240.

[12] Koren, I., and Zinaty, O. Evaluating elementary functions in a numerical coprocessor based on rational approximations. *Computers, IEEE Transactions on 39*, 8 (1990), 1030–1037.

[13] Lang, T., and Muller, J.-M. Bounds on runs of zeros and ones for algebraic functions. In *Computer Arithmetic, 2001. Proceedings. 15th IEEE Symposium on* (2001), IEEE, pp. 13–20.

[14] Lee, D.-U., Cheung, R. C., Luk, W., and Villasenor, J. D. Hardware implementation trade-offs of polynomial approximations and interpolations. *Computers, IEEE Transactions on 57*, 5 (2008), 686–701.

[15] Lefèvre, V., and Muller, J.-M. Worst cases for correct rounding of the elementary functions in double precision. In *Computer Arithmetic, 2001. Proceedings. 15th IEEE Symposium on* (2001), IEEE, pp. 111–118.

[16] Lefèvre, V., Muller, J.-M., and Tisserand, A. Toward correctly rounded transcendentals. *IEEE Transactions on Computers 47*, 11 (1998), 1235–1243.

[17] Lorentz, G. G. *Approximation of functions*, vol. 322. American Mathematical Soc., 2005.

[18] Muller, J.-M. *Elementary Functions: Algorithms and Implementation*. Birkhauser Boston, Inc., Secaucus, NJ, USA, 1997.

[19] Muller, J.-M. A few results on table-based methods. In *Developments in Reliable Computing*. Springer, 1999, pp. 279–288.

[20] Muller, J.-M., Brisebarre, N., de Dinechin, F., Jeannerod, C.-P., Lefèvre, V., Melquiond, G., Revol, N., Stehlé, D., and Torres, S. *Handbook of Floating-Point Arithmetic*. Birkhäuser Boston, 2010.

[21] Muller, J.-M., and Muller, J.-M. *Elementary functions*. Springer, 2006.

[22] Munro, I., and Paterson, M. Optimal algorithms for parallel polynomial evaluation. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science* (1971), IEEE, pp. 132–139.

[23] Oberman, S. F. Floating point division and square root algorithms and implementation in the AMD-K7 TM microprocessor. In *Computer Arithmetic, 1999. Proceedings. 14th IEEE Symposium on* (1999), IEEE, pp. 106–115.

[24] Pachón, R., and Trefethen, L. N. Barycentric-Remez algorithms for best polynomial approximation in the chebfun system. *BIT Numerical Mathematics 49*, 4 (2009), 721–741.

[25] Payne, M. H., and Hanek, R. N. Radian reduction for trigonometric functions. *ACM SIGNUM Newsletter 18*, 1 (1983), 19–24.

[26] Pineiro, J.-A., Oberman, S. F., Muller, J.-M., and Bruguera, J. D. High-speed function approximation using a minimax quadratic interpolator. *Computers, IEEE Transactions on 54*, 3 (2005), 304–318.

[27] Remez, E. Y. Sur la détermination des polynômes d'approximation de degré donnée. *Comm. Soc. Math. Kharkov 10* (1934), 41–63.

[28] Schulte, M., and Swartzlander, E. Exact rounding of certain elementary functions. In *Computer Arithmetic, 1993. Proceedings., 11th Symposium on* (1993), IEEE, pp. 138–145.

[29] Schulte, M. J., and Stine, J. E. Approximating elementary functions with symmetric bipartite tables. *Computers, IEEE Transactions on 48*, 8 (1999), 842–847.

[30] Schulte, M. J., and Swartzlander, E. E. Hardware designs for exactly rounded elementary functions. *Computers, IEEE Transactions on 43*, 8 (1994), 964–973.

[31] Stehlé, D., Lefèvre, V., and Zimmermann, P. Searching worst cases of a one-variable function using lattice reduction. *Computers, IEEE Transactions on 54*, 3 (2005), 340–346.

[32] Strollo, A. G., De Caro, D., and Petra, N. Elementary functions hardware implementation using constrained piecewise-polynomial approximations. *Computers, IEEE Transactions on 60*, 3 (2011), 418–432.

[33] Tang, P. T. P. Table-lookup algorithms for elementary functions and their error analysis. In *IEEE Symposium on Computer Arithmetic* (1991), pp. 232–236.

[34] Wong, W.-F., and Goto, E. Fast hardware-based algorithms for elementary function computations using rectangular multipliers. *Computers, IEEE Transactions on 43*, 3 (1994), 278–294.

[35] Wong, W.-F., and Goto, E. Fast evaluation of the elementary functions in single precision. *Computers, IEEE Transactions on 44*, 3 (1995), 453–457.

[36] Ziv, A. Fast evaluation of elementary mathematical functions with correctly rounded last bit. *ACM Trans. Math. Softw. 17*, 3 (Sept. 1991), 410–423.

الملخص

الرسالة تقدم طريقة مخصصة للهاردوير لحساب الدوال الأولية مُقربة تقريباً صحيحاً طبقاً لمعيار IEEE-754. الطريقة تقدم نهج غير مُجدول باستخدام دالة تقريبية مختارة لتقليل أقصى خطأ بدرجة عالية على نطاق واحد واسع بعكس الطرق المُجدولة السائدة حالياً و التى تستخدم كمية ضخمة من الذاكرة لتحقيق التقريب صحيح بدرجة منخفضة. الطريقة تم تطبيقها بشكل أساسى للدوال المثلثية لسعة الدقة المنفردة لقيم مدخلات حتى 2^{53}. الطريقة تطبق المرحلتان تصغير النطاق و تقدير الدالة التقريبية باستخدام وحدة جمع و ضرب من سعة الدقة المزدوجة. مرحلة تصغير النطاق تستخدم خوارزمية موحدة لجميع المدخلات و هى منبثقة من طريقة كودى ويت. نهج الطريقة يسمح بحساب نطاق ضخم من مدخلات الدوال المثلثية و باستخدام كمية منخفضة من الذاكرة (أقل من 700 بت) وهى كمية قد تصل الى أقل من خمس مرات من الكمية المستخدمة فى الطرق المُجدولة لحساب النتائج على نطاق ضيق بتقريب من النمط الأمين الأقل دقة من التقريب الصحيح. الطريقة جاهزة لدعم جميع أنواع التقريب المطلوبة من المعيار IEEE-754 و تم تجريبها على الدوال المثلثية لسعة الدقة المزدوجة كما تم استخدامها على الدوال الأسية و اللوغارتمية لسعة الدقة المنفردة.

طريقة غير مُجدولة لتقريب الدوال الأولية تقريباً صحيحاً طبقاً لمعيار IEEE-754

اعداد
جورج كمال كامل بدوى

رسالة مقدمة إلى كلية الهندسة - جامعة القاهرة
كجزء من متطلبات الحصول على درجة ماجستيرالعلوم
في
هندسة الالكترونيات و الاتصالات الكهربية

يعتمد من لجنة الممتحنين:

أ.د. حسام على حسن فهمى المشرف الرئيسى

أ.م.د. ابراهيم محمد قمر الممتحن الداخلي

أ.د. محمد واثق على كامل الخراشى الممتحن الخارجي
استاذ بقسم الحاسبات و النظم، كلية الهندسة – جامعة عين شمس

كليــة الهندســة - جامعــة القاهــرة
الجيـزة - جمهوريـة مصـر العربيــة

٢٠١٥

طريقة غير مُجدولة لتقريب الدوال الأولية تقريباً صحيحاً طبقاً لمعيار IEEE-754

اعداد
جورج كمال كامل بدوى

رسالة مقدمة إلى كلية الهندسة - جامعة القاهرة
كجزء من متطلبات الحصول على درجة ماجستيرالعلوم
في
هندسة الالكترونيات و الاتصالات الكهربية

تحت اشراف

أ. د. حسام على حسن فهمى

استاذ بقسم هندسة الالكترونيات و الاتصالات الكهربية
كلية الهندسة، جامعة القاهرة

كليــة الهندســة - جامعــة القاهــرة
الجيـزة - جمهوريـة مصـر العربيــة

٢٠١٥

طريقة غير مُجدولة لتقريب الدوال الأولية تقريباً صحيحاً طبقاً لمعيار IEEE-754

اعداد

جورج كمال كامل بدوى

رسالة مقدمة إلى كلية الهندسة - جامعة القاهرة
كجزء من متطلبات الحصول على درجة ماجستيرالعلوم
في
هندسة الالكترونيات و الاتصالات الكهربية

كليــة الهندســة - جامعــة القاهــرة
الجيـزة - جمهوريـة مصـر العربيــة

٢٠١٥