Hardware Development of a Rational Number Arithmetic Computation Unit Using Binary Continued Logarithms

Joshua Williams
Computer Science Department
Tufts University
Medford, United States
joshua.williams645895@tufts.edu

Ege Ozgul
Electrical Engineering Department
Tufts University
Medford, United States
Ege.Ozgul@Tufts.edu

Mark Hempstead
Electrical Engineering Department
Tufts University
Medford, United States
Mark.Hempstead@Tufts.edu

***Abstract*—In this paper, we develop an Arithmetic Computation Unit for operating on Binary Continued Logarithms (BCLs) a novel decimal number representation That can represent any rational number losslessly given a large enough width. The paper first presents the mathematical theory that allows storing high-precision decimal numbers, then it outlines the theory for performing mathematical operations on the type. After presenting the mathematical theory, the custom type and operations are implemented in VHDL and a hardware accelerator is built which is dedicated to performing high-precision decimal arithmetic on the type.**

***Index Terms*—Binary Continued Logarithms, Continued Fractions, Error Free Arithmatic**

I. Introduction

In processors, decimal numbers need to be stored in a different format than standard integers, and also custom decimal arithmetic algorithms need to be designed in order to perform decimal arithmetic computations. Although floating point representation provides a decent level of accuracy and precision to represent decimal values, it introduces significant error which can accumulate and cause inaccurate computations where precision is required. So floating point arithmetic units fail to meet the precision requirements in certain applications. In this paper, a novel decimal number representation is presented using binary continued fractions. This new type can represent rational numbers with exact accuracy given enough bits, and it can also represent irrational numbers with high accuracy where accuracy depends on the number of bits available. In this project, the mathematical representation and arithmetic operations are defined and presented. Also in this project, the defined type and operations are implemented in VHDL to build a hardware accelerator dedicated to performing decimal number computations.

A. Limitations of floating point format

Floating point format is widely used in almost all processors to handle decimal arithmetic. While this format is very useful for many applications, it has some accuracy limitations: For example, 1.2 - 1.1 is computed as 0.099999999 when floating point is used. Imagine using the results of such computations to do more such arithmetic in a for loop. The small error would accumulate and eventually become a very big error, especially for chaotic systems. In certain applications where accuracy and precision is vital, floating point format is not sufficient.

B. Use Cases of Binary Continued Logarithms Computation Unit

In scientific computing, high-precision arithmetic is becoming more and more important as it allows for more accurate solutions to scientific problems. In recent years, the use of high precision arithmetic has been applied to a wide variety of scientific problems. In this section, numerous applications which have recently arisen for high-precision arithmetic in scientific computing are summarized.

One main application of high precision arithmetic is in solving optimization problems[7,8]. Quadruple precision has been proven to be effective in generating numerically stable solutions to numerous problems in fields like mathematical biology. Additionally, high precision computation has been utilized in the computer aided solution for the Smale's 14th problem [9], as well as in finding numerically reliable eigenvalues for anharmonic oscillators[10].

In planetary orbit dynamics, higher precision arithmetic (typically IEEE extended or double-double) is used to reduce the numerical error, in simulations of the solar system [11,12]. High-precision arithmetic has also been utilized to study n-body Coulomb atomic systems, where large and nearly singular linear systems need to be solved[13].

Furthermore, high precision computations have been used to find accurate solutions to the Schrödinger equation of the lithium atom [14]. The use of high-precision arithmetic is also important in computing scattering amplitudes of collisions involving fundamental particles, such as quarks, gluons and bosons[16-19].

Finally, very high precision (up to 10,000 digit) has been utilized to compute determinants and minors of extremely large matrices that arise in the analysis of the Zeroes of the Riemann Zeta function [20].

II. Mathematical Theory

A. Continued Fractions Overview

A Continued fraction is a mathematical form that is capable of representing decimal numbers. The figure below displays the Generalized Continued Fractions (GCF) form:

$$x = b_0 + \cfrac{a_0}{b_1 + \cfrac{a_1}{b_2 + \cfrac{a_2}{b_3 + \cfrac{a_3}{\ddots + \cfrac{a_{n-1}}{b_n}}}}} \tag{1}$$

Below is an alternative representation of x:

$$x = GCF \begin{bmatrix} b_0 & b_1 & b_2 & b_3 & \dots & \dots & b_n \\ a_0 & a_1 & a_2 & a_3 & \dots & a_{n-1} & \end{bmatrix} \tag{2}$$

The example below displays a continued fraction representation of a rational number:

$$\frac{181}{101} = 1 + \cfrac{1}{1 + \cfrac{1}{3 + \cfrac{1}{1 + \cfrac{1}{4 + \cfrac{1}{4}}}}} \tag{3}$$

Below are the terms needed to represent $\frac{181}{101}$

$$\frac{181}{101} = GCF \begin{bmatrix} 1 & 1 & 3 & 1 & 4 & 4 \\ 1 & 1 & 1 & 1 & 1 & \end{bmatrix} \tag{4}$$

Irrational numbers can also be approximated using the continued fractions format. In (5), π is represented using the continued fractions representation. Note that since π is irrational, the form is infinite, so we would need an infinite number of terms to exactly represent the number π.

$$\pi = 0 + \cfrac{4}{1 + \cfrac{1^2}{3 + \cfrac{3^2}{5 + \cfrac{5^2}{\dots}}}} \tag{5}$$

Again represented in the alternative form.

$$\pi = GCF \begin{bmatrix} 0 & 1 & 3 & 5 & 7 & \cdots \\ 4 & 1^2 & 3^2 & 5^2 & \cdots & \end{bmatrix} \tag{6}$$

B. Arithmetic and the BLFT

While continued fractions have been formalized since 1613 [21] and algorithms using and relating to them have existed since Euclid's Algorithm [22], it was believed that precise native arithmetic on them was impossible. However, in 1977 this was shown to be false by Gosper [1] who introduced his own methodology for computing Linear (7) and Bilinear (8) transforms.

The Linear Fractional Transformation (LFT) is the unitary operation of continued fractional arithmetic, and if a function can be put in the form shown below (7) it can be evaluated using Gosper's arithmetic.

$$LFT\,(x) \rightarrow \frac{Ax + B}{Cx + D} \tag{7}$$

Similarly, the Bilinear Fractional Transformation (BLFT) is the binary operation of continued fractional arithmetic and is used when there are two inputs x and y.

$$BLFT\,(x, y) \rightarrow \frac{Axy + Bx + Cy + D}{Exy + Fx + Gy + H} \tag{8}$$

To evaluate an LFT with an input x (9) and a result z we can ingest terms from x in (b, a) pairs and emit terms to z also in (b, a) pairs.

$$x = GCF \begin{bmatrix} b_{x,0} & b_{x,1} & \dots & \dots & b_{x,n} \\ a_{x,0} & a_{x,1} & \dots & a_{x,n-1} & \end{bmatrix} \tag{9}$$

When a pair of terms are ingested or emitted, the information they contain is transferred to or from the coefficients of the LFT respectively. We use i to indicate the number of term pairs ingested from x, and k to indicate the number of pairs emitted to z, then we can notate the LFT at a given point as ϵ_i^k (10).

$$\epsilon_i^k = \frac{A_i^k x_i + B_i^k}{C_i^k x_i + D_i^k} \tag{10}$$

$$x_i = GCF \begin{bmatrix} b_{x,i} & b_{x,i+1} & \dots & \dots & b_{x,n} \\ a_{x,i} & a_{x,i+1} & \dots & a_{x,n-1} & \end{bmatrix} \tag{11}$$

This LFT ϵ_i^k encapsulates all of the information of the calculation that is yet to be transferred into the result z. In (12) this is shown by the inclusion of ϵ_i^k as the final b term of z_k, and once the value of $\epsilon_i^k = \infty$ then $z_k = z$ and the computation is complete.

$$z_k = GCF \begin{bmatrix} b_{z,0} & b_{z,1} & \dots & b_{z,k-1} & \epsilon_i^k \\ a_{z,0} & a_{z,1} & \dots & a_{z,k-1} & \end{bmatrix} \tag{12}$$

To ingest a pair of terms from x we replace all occurrences of x_i in ϵ_i^k with $\left(b_{x,i} + \frac{a_{x,i}}{x_{i+1}}\right)$ (13) and then rearrange back into the LFT form (14). The terms are collected and form the new LFT coefficients (15).

$$\epsilon_{i+1}^k = \frac{A_i^k \left(b_{x,i} + \frac{a_{x,i}}{x_{i+1}}\right) + B_i^k}{C_i^k \left(b_{x,i} + \frac{a_{x,i}}{x_{i+1}}\right) + D_i^k} \tag{13}$$

$$= \frac{\left(A_i^k b_{x,i} + B_i^k\right) x_{i+1} + A_i^k a_{x,i}}{\left(C_i^k b_{x,i} + D_i^k\right) x_{i+1} + C_i^k a_{x,i}} \tag{14}$$

$$= \frac{A_{i+1}^k x_{i+1} + B_{i+1}^k}{C_{i+1}^k x_{i+1} + D_{i+1}^k} \tag{15}$$

A similar process can be done for emission. But first, we take the continued fraction form of ϵ_i^k (16) and rearrange it in order to get ϵ_i^{k+1} (17). Substituting the previous LFT in for ϵ_i^k (18) we can then rearrange into the LFT form (19) and collect terms for the new LFT coefficients (20).

$$\epsilon_i^k = b_{z,k} + \frac{a_{z,k}}{\epsilon_i^{k+1}} \tag{16}$$

$$\epsilon_i^{k+1} = \frac{a_{z,k}}{\epsilon_i^k - b_{z,k}} \tag{17}$$

$$= \frac{a_{z,k}}{\dfrac{A_i^k x_i + B_i^k}{C_i^k x_i + D_i^k} - b_{z,k}} \tag{18}$$

$$= \frac{C_i^k a_{z,k} x_i + D_i^k a_{z,k}}{\left(A_i^k - C_i^k b_{z,k}\right) x_i + \left(B_i^k - D_i^k b_{z,k}\right)} \tag{19}$$

$$= \frac{A_i^{k+1} x_i + B_i^{k+1}}{C_i^{k+1} x_i + D_i^{k+1}} \tag{20}$$

Both ingestion and emission can be framed as matrix operations on LFT and BLFT coefficient vectors and we will use this form later, when we show that for the BCL these matrix operations can be further decomposed into fixed operations.

LFT Matrix operations

$$\text{(x-ingest)} \begin{bmatrix} b_{x,i} & 1 & 0 & 0 \\ a_{x,i} & 0 & 0 & 0 \\ 0 & 0 & b_{x,i} & 1 \\ 0 & 0 & a_{x,i} & 0 \end{bmatrix} \begin{bmatrix} A_i^k \\ B_i^k \\ C_i^k \\ D_i^k \end{bmatrix} = \begin{bmatrix} A_{i+1}^k \\ B_{i+1}^k \\ C_{i+1}^k \\ D_{i+1}^k \end{bmatrix} \tag{21}$$

$$\text{(z-emmit)} \begin{bmatrix} 0 & 0 & a_{z,k} & 0 \\ 0 & 0 & 0 & a_{z,k} \\ 1 & 0 & -b_{z,k} & 0 \\ 0 & 1 & 0 & -b_{z,k} \end{bmatrix} \begin{bmatrix} A_i^k \\ B_i^k \\ C_i^k \\ D_i^k \end{bmatrix} = \begin{bmatrix} A_i^{k+1} \\ B_i^{k+1} \\ C_i^{k+1} \\ D_i^{k+1} \end{bmatrix} \tag{22}$$

We have not included the derivation of BLFT operations. They follow the exact same method as for the LFT and should be easily derivable by the reader.

BLFT Matrix operations

$$\text{(x-ingest)} \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ b_{x,i} & b_{x,i} & 0 & 1 & 0 & 0 & 0 & 0 \\ a_{x,i} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & a_{x,i} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & b_{x,i} & b_{x,i} & 0 & 1 \\ 0 & 0 & 0 & 0 & a_{x,i} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & a_{x,i} & 0 & 0 \end{bmatrix} \begin{bmatrix} A_{i,j}^k \\ B_{i,j}^k \\ C_{i,j}^k \\ D_{i,j}^k \\ E_{i,j}^k \\ F_{i,j}^k \\ G_{i,j}^k \\ H_{i,j}^k \end{bmatrix} = \begin{bmatrix} A_{i+1,j}^k \\ B_{i+1,j}^k \\ C_{i+1,j}^k \\ D_{i+1,j}^k \\ E_{i+1,j}^k \\ F_{i+1,j}^k \\ G_{i+1,j}^k \\ H_{i+1,j}^k \end{bmatrix} \tag{23}$$

$$\text{(y-ingest)} \begin{bmatrix} b_{y,j} & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ a_{y,j} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & b_{y,j} & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & a_{y,j} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & b_{y,j} & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & a_{y,j} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & b_{y,j} & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & a_{y,j} & 0 \end{bmatrix} \begin{bmatrix} A_{i,j}^k \\ B_{i,j}^k \\ C_{i,j}^k \\ D_{i,j}^k \\ E_{i,j}^k \\ F_{i,j}^k \\ G_{i,j}^k \\ H_{i,j}^k \end{bmatrix} = \begin{bmatrix} A_{i,j+1}^k \\ B_{i,j+1}^k \\ C_{i,j+1}^k \\ D_{i,j+1}^k \\ E_{i,j+1}^k \\ F_{i,j+1}^k \\ G_{i,j+1}^k \\ H_{i,j+1}^k \end{bmatrix} \tag{24}$$

$$\text{(z-emmit)} \begin{bmatrix} 0 & 0 & 0 & 0 & a_{z,k} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & a_{z,k} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & a_{z,k} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & a_{z,k} \\ 1 & 0 & 0 & 0 & -b_{z,k} & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & -b_{z,k} & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & -b_{z,k} & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & -b_{z,k} \end{bmatrix} \begin{bmatrix} A_{i,j}^k \\ B_{i,j}^k \\ C_{i,j}^k \\ D_{i,j}^k \\ E_{i,j}^k \\ F_{i,j}^k \\ G_{i,j}^k \\ H_{i,j}^k \end{bmatrix} = \begin{bmatrix} A_{i,j}^{k+1} \\ B_{i,j}^{k+1} \\ C_{i,j}^{k+1} \\ D_{i,j}^{k+1} \\ E_{i,j}^{k+1} \\ F_{i,j}^{k+1} \\ G_{i,j}^{k+1} \\ H_{i,j}^{k+1} \end{bmatrix} \tag{25}$$

C. Problems with Binary Implementation

There are a number of challenges in designing an accelerator using the continued fractions to perform decimal arithmetic.

1) Equivilancy:

$$\frac{465}{83} = 4 + \cfrac{7}{3 + \cfrac{13}{8 + \cfrac{3}{2}}} = 5 + \cfrac{2}{2 + \cfrac{8}{4 + \cfrac{20}{5 + \cfrac{15}{3 + \cfrac{3}{16}}}}} \tag{26}$$

Within GCFs there are uncountably infinite ways to represent a single value. This redundancy introduces a high level of complexity to any proposed hardware design and reduces the number of bits of information stored per hardware bit. There are some subsets of GCFs such as Regular Continued Fractions (RCFs have all a terms set to 1) (27) that improve this, but none that bring the information density to parity.

$$\frac{465}{83} = 5 + \cfrac{1}{1 + \cfrac{1}{1 + \cfrac{1}{1 + \cfrac{1}{1 + \cfrac{1}{16}}}}} = 5 + \cfrac{1}{1 + \cfrac{1}{1 + \cfrac{1}{1 + \cfrac{1}{1 + \cfrac{1}{15 + \cfrac{1}{1}}}}}} \tag{27}$$

2) Term Width and Organization: The number of bits required to represent a term is unbounded (28), as a result, it is difficult to organize them efficiently in memory while maintaining a high information density. A dynamic delimiting scheme would be required for terms, or a fixed width could be used per term, but this creates an artificial limit on term size and causes issues with Gosper arithmetic.

$$\frac{2^{33}-1}{2^{32}} = 1 + \cfrac{1}{1+\cfrac{1}{2^{32}-1}} \tag{28}$$

3) Linear Type: The range of a GCF and RCF are both linearly proportional to the size of their terms. This means that in comparison to floating point numbers, a much larger number of bits is necessary to represent a number with a large order of magnitude.

4) Term Emission: GCFs do not necessarily converge with each subsequent pair of terms and as a result the choice of when and what to emit to z is somewhat arbitrary. However, certain subsets of GCFs, including RCFs and BCLs, do converge. As a result, we are able to decide when and what to emit by evaluating the LFT or BLFT as its inputs approach the limits of their representable range(s), emitting if there is a consensus on the value of the next term. In specific cases these limits fail to converge to a consensus within a finite number of ingestions. Seidensticker [5] and Brabec [3] have both proposed using speculative term emission in order to force convergence, either reverting if an error is found [3] or using GCF equivalency to cancel out the error with subsequent terms [5].

D. BCLs, a Subset of GCFs

Following on from work by Brabec [2] our implementation uses a subset of GCFs known as Binary Continued Logarithms (BCLs). These BCLs are an extension of Canonical Continued Logarithms (CCLs) introduced by Gosper [1]. CCLs have a limited range of $[1, \infty]$ and are therefore not usable as a generalized numerical type. One advantage over other forms of Continued Fractions is that they scale exponentially with their terms, similar to floating point numbers, and therefore can represent the full range of rational numbers larger than 1 with a limited number of bits, with more bits increasing the granularity rather than range.

$$x = 2^{\alpha_0} + \cfrac{2^{\alpha_0}}{2^{\alpha_1} + \cfrac{2^{\alpha_1}}{\ddots + \cfrac{2^{\alpha_{n-1}}}{2^{\alpha_n} + \cfrac{2^{\alpha_n}}{\infty}}}} \geq 1 \tag{29}$$

$$= CCL\left[\alpha_0 \quad \alpha_1 \quad \alpha_2 \quad \ldots \quad \alpha_{n-1} \quad \alpha_n\right] \geq 1 \tag{30}$$

BCLs introduce two extra terms α_s and α_r, each of which is either 0 or 1. Between these two terms the range of the representation is increased to include all rational numbers $[-\infty, \infty]$

$$x = (-1)^{\alpha_s} \times \left(2^{\alpha_0} + \cfrac{2^{\alpha_0}}{2^{\alpha_1} + \cfrac{2^{\alpha_1}}{\ddots + \cfrac{2^{\alpha_{n-1}}}{2^{\alpha_n} + \cfrac{2^{\alpha_n}}{\infty}}}}\right)^{(-1)^{\alpha_r}} \tag{31}$$

$$= BCL\left[\alpha_s \quad \alpha_r; \quad \alpha_0 \quad \alpha_1 \quad \alpha_2 \quad \ldots \quad \alpha_{n-1} \quad \alpha_n\right] \tag{32}$$

The BCL also lends itself to easy representation in hardware. The two special terms α_s and α_r are treated in the same fashion as the sign bit of floating point numbers and take up the two most significant bits. For each CCL term α_i we use $\alpha_i + 1$ bits, α_i 1 bits followed by a delimiting 0 bit. We continue this until we either run out of bits in the chosen word width, or we have no more terms, at which point all remaining bits are set to 1. This way there are no wasted bits between terms, and if there are not enough bits to represent all terms, the truncation is the closest representable value. There are also an infinite number of implied 1 bits after the end, which represent the final infinite term and allow for the same value to be represented at multiple widths, as long as there are more than the minimum number of bits.

$$-\frac{53}{97} = (-1)^1 \times \left(2^0 + \cfrac{2^0}{2^0 + \cfrac{2^0}{2^2 + \cfrac{2^2}{2^2 + \cfrac{2^2}{2^3 + \cfrac{2^3}{2^\infty}}}}}\right)^{(-1)^1} \tag{33}$$

$$= [1\ 1\ ;\ 0\ 0\ 2\ 2\ 3\ \infty] \tag{34}$$

$$\text{Binary (16-bit)} \rightarrow 11\ 00110110111011(11111\ldots) \tag{35}$$

$$\text{Min \# of Bits} = 2 + n + \sum_{i=0}^{n-1} \alpha_i \tag{36}$$

One drawback of this binary representation is since terms are effectively encoded in the unary number system our terms have a limited range. This range issue cancels out with the type's exponential scale giving a max non ∞ value of $2^{\text{\# bits - 3}}$.

As mentioned in Sec II.A. the matrix transforms associated with each term of GCL arithmetic can be decomposed into fixed operations for BCLs. These fixed operations are not only much simpler to implement in hardware, but they also correspond one-to-one with the bits of the input BCL(s) and the output BCL.

The Decomposition of matrix operations of the LFT are demonstrated below. (37) shows the GCF LFT ingestion matrix with its a and b terms replaced with the terms of the BCL. (38) decomposes this matrix into two matrices, one with skewing terms and the other a diagonal matrix raised to the power of $\alpha_{x,i}$. Each of the diagonal matrices that make up the exponentiated matrix corresponds to a 1 bit in the $\alpha_{x,i}$ term of x and the skewing matrix corresponds to the corresponding delimiting 0 bit. The same process and logic holds for bits of z as shown in (39) and (40).

$$\text{(x-ingest)} \begin{bmatrix} 2^{\alpha_{x,i}} & 1 & 0 & 0 \\ 2^{\alpha_{x,i}} & 0 & 0 & 0 \\ 0 & 0 & 2^{\alpha_{x,i}} & 1 \\ 0 & 0 & 2^{\alpha_{x,i}} & 0 \end{bmatrix} \begin{bmatrix} A_i^k \\ B_i^k \\ C_i^k \\ D_i^k \end{bmatrix} = \begin{bmatrix} A_{i+1}^k \\ B_{i+1}^k \\ C_{i+1}^k \\ D_{i+1}^k \end{bmatrix} \tag{37}$$

$$\begin{bmatrix} 2^{\alpha_{x,i}} & 1 & 0 & 0 \\ 2^{\alpha_{x,i}} & 0 & 0 & 0 \\ 0 & 0 & 2^{\alpha_{x,i}} & 1 \\ 0 & 0 & 2^{\alpha_{x,i}} & 0 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}^{\alpha_{x,i}} \tag{38}$$

$$\text{(z-emmit)} \begin{bmatrix} 0 & 0 & 2^{\alpha_{z,k}} & 0 \\ 0 & 0 & 0 & 2^{\alpha_{z,k}} \\ 1 & 0 & -2^{\alpha_{z,k}} & 0 \\ 0 & 1 & 0 & -2^{\alpha_{z,k}} \end{bmatrix} \begin{bmatrix} A_i^k \\ B_i^k \\ C_i^k \\ D_i^k \end{bmatrix} = \begin{bmatrix} A_i^{k+1} \\ B_i^{k+1} \\ C_i^{k+1} \\ D_i^{k+1} \end{bmatrix} \tag{39}$$

$$\begin{bmatrix} 0 & 0 & 2^{\alpha_{z,k}} & 0 \\ 0 & 0 & 0 & 2^{\alpha_{z,k}} \\ 1 & 0 & -2^{\alpha_{z,k}} & 0 \\ 0 & 1 & 0 & -2^{\alpha_{z,k}} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 2 \end{bmatrix}^{\alpha_{z,k}} \tag{40}$$

Furthermore, we can show via the equality between results of (41) and (42) that the order in which bits are ingested from x and emitted to z is irrelevant as long as the order is maintained within the bit operations of x and within the bit operations of z.

$$\begin{bmatrix} 2^{\alpha_{x,i}} & 1 & 0 & 0 \\ 2^{\alpha_{x,i}} & 0 & 0 & 0 \\ 0 & 0 & 2^{\alpha_{x,i}} & 1 \\ 0 & 0 & 2^{\alpha_{x,i}} & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & 2^{\alpha_{z,k}} & 0 \\ 0 & 0 & 0 & 2^{\alpha_{z,k}} \\ 1 & 0 & -2^{\alpha_{z,k}} & 0 \\ 0 & 1 & 0 & -2^{\alpha_{z,k}} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 2^{\alpha_{x,i}+\alpha_{z,k}} & 2^{\alpha_{z,k}} \\ 0 & 0 & 2^{\alpha_{x,i}+\alpha_{z,k}} & 0 \\ 2^{\alpha_{x,i}} & 1 & -2^{\alpha_{x,i}+\alpha_{z,k}} & -2^{\alpha_{z,k}} \\ 2^{\alpha_{x,i}} & 0 & -2^{\alpha_{x,i}+\alpha_{z,k}} & 0 \end{bmatrix} \tag{41}$$

$$\begin{bmatrix} 0 & 0 & 2^{\alpha_{z,k}} & 0 \\ 0 & 0 & 0 & 2^{\alpha_{z,k}} \\ 1 & 0 & -2^{\alpha_{z,k}} & 0 \\ 0 & 1 & 0 & -2^{\alpha_{z,k}} \end{bmatrix} \begin{bmatrix} 2^{\alpha_{x,i}} & 1 & 0 & 0 \\ 2^{\alpha_{x,i}} & 0 & 0 & 0 \\ 0 & 0 & 2^{\alpha_{x,i}} & 1 \\ 0 & 0 & 2^{\alpha_{x,i}} & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 2^{\alpha_{x,i}+\alpha_{z,k}} & 2^{\alpha_{z,k}} \\ 0 & 0 & 2^{\alpha_{x,i}+\alpha_{z,k}} & 0 \\ 2^{\alpha_{x,i}} & 1 & -2^{\alpha_{x,i}+\alpha_{z,k}} & -2^{\alpha_{z,k}} \\ 2^{\alpha_{x,i}} & 0 & -2^{\alpha_{x,i}+\alpha_{z,k}} & 0 \end{bmatrix} \tag{42}$$

All of these principles hold true for matrix operations of the BLFT with BCLs also, however we shall not include derivations or proofs in this paper.

E. Ingestion Operations

Ingestion operation consists of reading a single bit from one of the input registers x or y, then evaluating the bit to update the values of the terms A-H.

The first 2 bits of each input are the sign bit and the reciprocal bit. The regular bits are the rest of the bits in the BCL type. Note that if the emitted bit is zero and if the emitted bit either the sign bit or the reciprocal bit, then the registers are left unchanged.

The following table displays the updated values of the terms A-H for all different ingestion conditions.

Bit Type	Ingestion from x	Ingestion from y
sign-bit(1)	$\begin{bmatrix} -A & -B & C & D \\ -E & -F & G & H \end{bmatrix}$	$\begin{bmatrix} -A & B & -C & D \\ -E & F & -G & H \end{bmatrix}$
reciprocal-bit(1)	$\begin{bmatrix} C & D & A & B \\ G & H & E & F \end{bmatrix}$	$\begin{bmatrix} B & A & D & C \\ F & E & H & G \end{bmatrix}$
standard-bit(0)	$\begin{bmatrix} A+C & B+D & A & B \\ E+G & F+H & E & F \end{bmatrix}$	$\begin{bmatrix} A+B & A & C+D & C \\ E+F & E & G+H & G \end{bmatrix}$
standard-bit(1)	$\begin{bmatrix} 2A & 2B & C & D \\ 2E & 2F & G & H \end{bmatrix}$	$\begin{bmatrix} 2A & B & 2C & D \\ 2E & F & 2G & H \end{bmatrix}$

F. Emission Operations

Once the bit to be emitted is computed, the output registers are updated depending on the value of the emitted bit and the type of the emitted bit. Note that if the emitted bit is zero and if the emitted bit either the sign bit or the reciprocal bit, then the registers are left unchanged.

Bit Type	Emission to z
sign-bit(1)	$\begin{bmatrix} -A & -B & -C & -D \\ E & F & G & H \end{bmatrix}$
reciprocal-bit(1)	$\begin{bmatrix} E & F & G & H \\ A & B & C & D \end{bmatrix}$
standard-bit(0)	$\begin{bmatrix} E & F & G & H \\ A-E & B-F & C-G & D-H \end{bmatrix}$
standard-bit(1)	$\begin{bmatrix} A & B & C & D \\ 2E & 2F & 2G & 2H \end{bmatrix}$

G. Limits and Computation Overview for BLFT

To determine whether a bit can be emitted, and its value, first the β terms need to be computed. These β terms represent the limits of ϵ and are computed by evaluating ϵ as x and y approach the limits of their representable range. There are 3 different cases to determine the values of the β terms:

1) Reciprocal bits not ingested: If the reciprocal bits are not consumed, then x and y ranges are $[0, \infty]$, and the β terms converge to the following fractions.

$$\beta_0 = \lim_{\substack{x\to\infty \\ y\to\infty}} (\epsilon) = \frac{A}{E} \tag{43}$$

$$\beta_1 = \lim_{\substack{x\to\infty \\ y\to 0}} (\epsilon) = \frac{B}{F} \tag{44}$$

$$\beta_2 = \lim_{\substack{x\to 0 \\ y\to\infty}} (\epsilon) = \frac{C}{G} \tag{45}$$

$$\beta_3 = \lim_{\substack{x\to 0 \\ y\to 0}} (\epsilon) = \frac{D}{H} \tag{46}$$

2) Reciprocal bits are ingested: If the reciprocal bits are consumed, then β terms converge to following fractions.

$$\beta_0 = \lim_{\substack{x\to\infty \\ y\to\infty}} (\epsilon) = \frac{A}{E} \tag{47}$$

$$\beta_1 = \lim_{\substack{x\to\infty \\ y\to 1}} (\epsilon) = \frac{A+B}{E+F} \tag{48}$$

$$\beta_2 = \lim_{\substack{x\to 1 \\ y\to\infty}} (\epsilon) = \frac{A+C}{E+G} \tag{49}$$

$$\beta_3 = \lim_{\substack{x\to 1 \\ y\to 1}} (\epsilon) = \frac{A+B+C+D}{E+F+G+H} \tag{50}$$

3) All the input bits are ingested: If all of the input bits are ingested, the range representable by x and y is $[\infty, \infty]$. In this case, all β terms converge to $\frac{A}{E}$.

$$\beta_0 = \beta_1 = \beta_2 = \beta_3 = \lim_{\substack{x\to\infty \\ y\to\infty}} (\epsilon) \ = \frac{A}{E} \tag{51}$$

Once all of the β terms are computed, we can check if they agree on the emission value, if so we can emit. When emitting the sign bit all terms must have the same sign. When emitting the reciprocal bit all terms must either be in the range $[0, 1]$ or $[1, \infty]$ in order to agree on the output bit. For all other bit emissions, if all β terms are in the range [1,2], 0 bit is emitted. If all β terms are in the range [2,∞], 1 bit is emitted. If some of the β terms are in the range [1,2], and some are in the range [2,∞], then we continue to ingest from x and y until a consensus is reached and we can emit again. In the case where all β terms are on the boundary, either bit value can be emitted, this corresponds to the approximately 1 bit of redundancy that exists in the BCL.

H. Binary Continued Logarithm Operation Examples

Here is the standard form of the BLFT:

$$z = \frac{Axy + Bx + Cy + D}{Exy + Fx + Gy + H} \tag{52}$$

By setting the values of the coefficients (A - H), we can perform different arithmetic operations at the same computation cost. Below are some examples of initial BLFT coefficients for computing simple arithmetic operations:

Addition $(x + y)$:

$$z = \frac{(0)xy + (1)x + (1)y + (0)}{(0)xy + (0)x + (0)y + (1)} \tag{53}$$

Subtraction $(x - y)$:

$$z = \frac{(0)xy + (1)x + (-1)y + (0)}{(0)xy + (0)x + (0)y + (1)} \tag{54}$$

Multiplications $(x \times y)$:

$$z = \frac{(1)xy + (0)x + (0)y + (0)}{(0)xy + (0)x + (0)y + (1)} \tag{55}$$

Division (x/y):

$$z = \frac{(0)xy + (1)x + (0)y + (0)}{(0)xy + (0)x + (1)y + (0)} \tag{56}$$

III. Python Implementation

Initially, the entire accelerator was built using Python with approximately the same architecture as our RTL design. This allowed for rapid development and debugging of the unit. Once the Python implementation was improved, debugged, and optimized, it was then used to generate test data for each part of the hardware design. The test data consisted of an array of inputs and outputs. Each input from the array was given to the Python program, then it generated an output for each of the inputs. By running the program for each different input, we were able to generate a large array of test data.

We started converting the Python implementation into VHDL code to build the accelerator hardware. We built the functional blocks one by one, and we tested each of them against the Python code by using the test data that was generated from the Python functions.

IV. Hardware Design of Accelerator

A. Inputs and Outputs

There are 2 input and 2 output ports that are each 64-bit wide, and they are used for receiving and storing the input and output values. There are 3 types of values that these registers can take and produce:

- The denominator, numerator pair of a fraction, either provided to the input registers or stored in the output registers when the accelerator is used for performing conversion operations between a BCL and a fraction.
- 2 BCLs can be stored in the input and/or output registers for LFT or BLFT operations.
- The output 1 register can produce a floating-point number when converting a BCL to a floating point number.

B. Operations

This accelerator currently supports multiple arithmetic operations which are defined below:

- NOP: No operation code
- BCL: Given a denominator and numerator of a fraction, this BCL operation converts the fraction into the Binary continued Logarithms format.
- FRAC: Given an input encoded in Binary continued logarithms format, FRAC operations converts the given number into a fraction format by outputting the denominator and numerator of the fraction and stores them in out1 and out2 registers.
- DBL: This operation converts the input number encoded in binary continued logarithms format into double format.
- ADD: This operation adds 2 binary continued logarithms inputs and stores the output in out1 register.
- SUB: This operation subtracts 2 binary continued logarithms inputs and stores the output in out1 register.
- MUL: This operation multiplies 2 binary continued logarithms inputs and stores the output in out1 register.
- DIV: This operation divides 2 binary continued logarithms inputs and stores the output in out1 register.

C. Overview of the Accelerator Architecture

1) Initialization of Registers A-H: The starting values of the coefficients A-H are stored in the ROM, and those values are copied into the Registers A-H at the beginning of the computation.

2) Comparison Block - ingesting and emitting bits: This accelerator uses multiple cycles to perform arithmetic computations. At each cycle, one bit is either ingested or emitted. When ingesting, we ingest 1 bit from x or y. After ingesting from y, the comparison block computes the values of the $\beta_0, \beta_1, \beta_2, \beta_3$ terms. If all these terms lie in the same range, then a z bit is emitted and stored in the z register. We continue

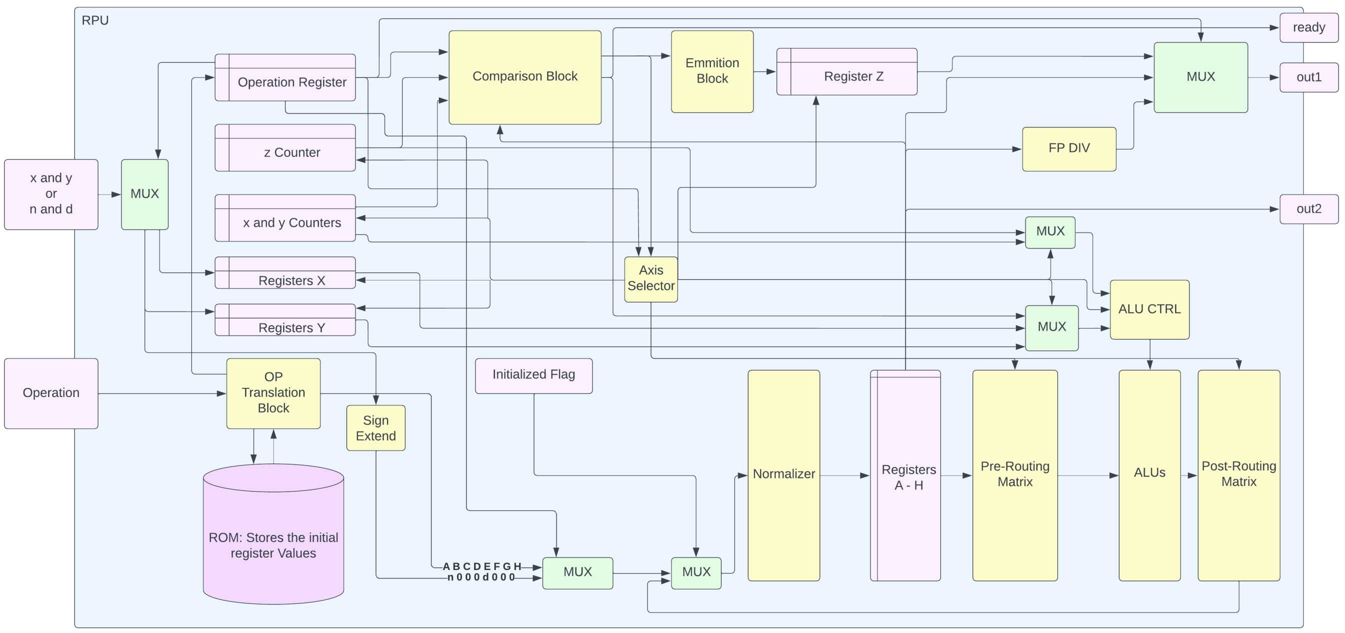

Fig. 1. Block Diagram Of the Hardware Design

to emit bits to z until we can no longer at which point we go back to ingesting, starting with x.

So the comparison block is responsible for determining the computations explained in section G (”Limits and Computation Overview for BLFT”).

3) Normalizer: The normalizer computes the greatest common power of 2 factor of registers A-H and divides each by this value. It may also negate the A-H registers in certain circumstances. Normalizing the register values in this way reduces overflows and it also enables correct bound evaluations in the comparison block.

4) Pre-Routing Matrix: The Pre-routing Matrix is made up of multiple muxes which change the order of registers A-H and provide the muxed values to the ALU layer. The order of the pre-routing matrix output depends on the state of the control wires which are controlled by the axis selector which is a unit responsible for choosing between x, y, and z registers.

5) ALU layer: The ALU layer receives the reordered values, and then it applies negation, swapping, bit shifting, and/or addition operations. The type of operation ALU layer performs is managed by the ALU controller (ALU CTRL).

6) Post Pre-Routing Matrix: The computed values are then routed into the Post routing matrix where the order of the results is updated, and then the new values are sent to the normalizer. This cycle repeats until all bits are emitted to the z register.

V. Results

At this stage, we still have a long way to go with our implementation. In an attempt to gather some results, we have created a couple of benchmarks in Python, matrix inversion and polynomial evaluation, where basic arithmetic and casting operators are provided as lambda arguments. This has allowed us to wrap Python’s inbuilt Fraction class, float primitive, and our BCL implementation in order to calculate the accuracy of BCLs vs floating point numbers one-to-one across a broad range of inputs. While the benchmarks are working, it appears that our BCL implementation is failings after an extended set of operations. This results in many of our results being $\pm\infty$ or 0. We have tried increasing the BCL width incase this is due to overflow or underflow however it has not fixed the issue. We believe that this is an error in our implementation rather than a limitation of BCLs since in [2], Brabec was able to evaluate several of our test cases, such as Rump’s polynomial without any apparent issue. We believe that the issue may be in the normalization stage as we fixed an issue earlier in development that presented in the same fashion as our current results.

VI. Limitation

A. *Irrational Functions Evaluations*

The binary continued logarithms type cannot represent irrational numbers and functions with exact accuracy; the irrational value can be approximated and stored in BCL format.

B. *1-bit of redundancy*

In the following numbers below, the BCL encodings would have 1 bit redundancy:

- 1 bit would be used to indicate the sign of the number, but zero does not have a sign:

$$-0 = 0$$

- 1 bit would be used to indicate the reciprocal of 1 which is also equal to itself:

$$1 = 1^{-1}$$

$$(-1) = (-1)^{-1}$$

The following general forms of BCL a and b are also equivalent.

$$a = (-1)^{\alpha_s} \times \left(2^{\alpha_0} + \cfrac{2^{\alpha_0}}{2^{\alpha_1} + \cfrac{2^{\alpha_1}}{\ddots + \cfrac{2^{\alpha_{n-1}}}{2^{\alpha_n} + \cfrac{2^{\alpha_n}}{\infty}}}} \right)^{(-1)^{\alpha_r}} \tag{57}$$

$$b = (-1)^{\alpha_s} \times \left(2^{\alpha_0} + \cfrac{2^{\alpha_0}}{2^{\alpha_1} + \cfrac{2^{\alpha_1}}{\ddots + \cfrac{2^{\alpha_{n-1}}}{2^{\alpha_n - 1} + \cfrac{2^{\alpha_n - 1}}{2^0 + \cfrac{2^0}{\infty}}}}} \right)^{(-1)^{\alpha_r}} \tag{58}$$

The only two numbers without equivalents are those for $\pm\infty$. As a result the approximate information density of the BCL is $\frac{\text{\# bits - 1}}{\text{\# bits}}$.

C. Distribution is non-uniform (fractal)

In Figure 2, all 129 unique 8-bit BCL values are generated and sorted in increasing order. The percentage change between subsequent numbers is plotted on the y axis. This demonstrates the non-uniform (fractal) precision of the BCL. This unpredictability in precision makes it difficult to accurately guarantee the minimum precision of a given width without enumerating all values, a task that is infeasible at larger widths.

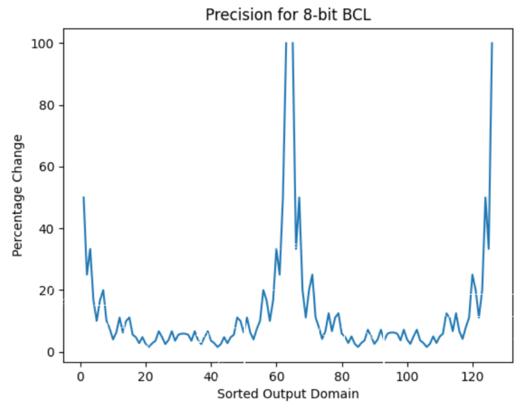

Fig. 2. Block Diagram Of the Hardware Design

VII. Conclusion

In conclusion, this paper presents a novel alternative decimal number format with high precision. By leveraging Binary Continued Logarithms and their innate binary nature we can accommodate both rational and irrational values with little to no rounding error. The paper also presents the theory for performing mathematical transformations on this decimal type.

Furthermore, in this project, a hardware accelerator was designed which is dedicated to performing arithmetic on the BCL format natively, as well as providing options for converting to and from fraction format. The designed BCL accelerator has some limitations discussed in the previous section. T here is also much room for refinement, improvement, and optimization of the accelerator. But this paper establishes the first steps toward the integration of higher precision arithmetic accelerators in processors and embedded systems. Using this accelerator, higher precision computation could be performed all in hardware which has the potential to build more efficient, faster, and more accurate embedded systems and technologies.

VIII. Future Work

- The existing VHDL implementation of the accelerator will be optimized for edge cases to produce more accurate results.
- The accelerator will be optimized to perform faster computations by introducing techniques like buffering, pipelining and parallelizm.
- The accelerator could be expanded into a Binary Continued Logarithms matrix computation accelerator which will leverage parallelism to perform high precision and fast matrix computations.
- Theoretically, the unit can be expanded to natively compute the results of nonlinear fractional transforms. We propose that this could be used to approximate irrational functions using Pade approximates, however, future research is necessary to determine whether this is feasible for a hardware implementation.
- The unit will be improved to capture error conditions and raise exceptions.
- The BCL accelerator will be improved to support multi-word evaluation. Because results are computed most significant bit first, results converge to the correct value. In the case where a result does not fit into a single BCL the remaining information remains in the A-H registers. We can leverage this but raising an incomplete computation flag, allowing the user to trap on it and continue to compute the result to a higher precision. To complement this we would also add a method to operate on multiword inputs.

References

[1] R. W. Gosper. Continued Fraction Arithmetic. Unpublished manuscript, 1977.
[2] T. Brabec, "Hardware Implementation of Continued Logarithm Arithmetic," Proc. 12th GAMM IMACS Int'l Symp. Scientific Computing, Computer Arithmetic and Validated Numerics (SCAN '06) Conf. Post-Proc., 2006

[3] T. Brabec, ”Speculatively Redundant Continued Logarithm Representation,” in IEEE Transactions on Computers, vol. 59, no. 11, pp. 1441-1454, Nov. 2010, doi: 10.1109/TC.2010.110.

[4] Jonathan M. Borwein, Neil J. Calkin, Scott B. Lindstrom & Andrew Mattingly (2017)Continued Logarithms and Associated Continued Fractions, Experimental Mathematics, 26:4, 412-429, DOI: 10.1080/10586458.2016.1195307

[5] Seidensticker, Bob. (1983). Continued fractions for high-speed and high-accuracy computer arithmetic. 10.1109/ARITH.1983.6158099.

[6] Bailey, D.H.;Borwein, J.M. High-Precision Arithmetic in Mathematical Physics. Mathematics 2015, 3, 337-367. https://doi.org/10.3390/math3020337

[7] Ma, D.; Saunders, M. Solving multiscale linear programs using the simplex method in quadruple precision. 2014. Available online: http://stanford.edu/group/SOL/multiscale/papers/quadLP3.pdf accessed on 30 April 2015.

[8] Ma, D.; Saunders, M. Experiments with linear and nonlinear optimization using quad precision. 2014. Available online: http://stanford.edu/group/SOL/multiscale/talks/14informsQuadMINOS.pdf accessed on 30 April 2015.

[9] Tucker, W. A. rigorous ODE solver and Smale’s 14th problem. Found. Comput. Math. 2002, 2, 53–117. [Google Scholar]

[10] Macfarlane, M. H. A high-precision study of anharmonic-oscillator spectra. Ann. Phys. 1999, 271, 159–202. [Google Scholar]

[11] Lake, G.; Quinn, T.; Richardson, D. C. From Sir Isaac to the Sloan survey: Calculating the structure and chaos due to gravity in the universe, Proceedings of the 8th ACM-SIAM Symposium on Discrete Algorithms, New Orleans, LA, USA, November 1997; pp. 1–10.

[12] Farrés, A.; l Laskar, J.; Blanes, S.; Casas, F.; Makazaga, J.; Murua, A. High precision symplectic integrators for the Solar System. Celest. Mech. Dyn. Astron. 2013, 116, 141–174. [Google Scholar]

[13] Frolov, A. M.; Bailey, D. H. Highly accurate evaluation of the few-body auxiliary functions and four-body integrals. J. Phys. B 2003, 36, 1857–1867. [Google Scholar]

[14] Yan, Z.-C.; Drake, G. W. F. Bethe logarithm and QED shift for Lithium. Phys. Rev. Lett. 2003, 81, 774–777. [Google Scholar]

[15] Zhang, T.; Yan, Z.-C.; Drake, G. W. F. QED corrections of O(mc2a7 ln a) to the fine structure splittings of Helium and He-Like ions. Phys. Rev. Lett. 1994, 77, 1715–1718. [Google Scholar]

[16] Ellis, R. K.; Giele, W. T.; Kunszt, Z.; Melnikov, K.; Zanderighi, G. One-loop amplitudes for W+3 jet production in hadron collisions. 2008. Available online: http://arXiv.org/abs/0810.2762 accessed on 30 April 2015.

[17] Berger, C. F.; Bern, Z.; Dixon, L. J.; Febres Cordero, F.; Forde, D.; Ita, H.; Kosower, D.A.; Maitre, D. An automated implementation of on-shell methods for one-loop amplitudes. Phys. Rev. D 2008, 78, 036003. [Google Scholar]

[18] Ossola, G.; Papadopoulos, C. G.; Pittau, R. CutTools: A program implementing the OPP reduction method to compute one-loop amplitudes. J. High-Energy Phys. 2008, 0803, 042. [Google Scholar]

[19] Czakon, M. Tops from light quarks: Full mass dependence at two-Loops in QCD. Phys. Lett. B 2008, 664, 307. [Google Scholar]

[20] Beliakov, G.; Matiyasevich, Y. A parallel algorithm for calculation of large determinants with high accuracy for GPUs and MPI clusters. 2013. Available online: http://arxiv.org/abs/1308.1536 accessed on 30 April 2015.

[21] Cataldi, Pietro. Trattato Del Modo Brevissimo Di Trovar La Radice Quadra Delli Numeri. Cochi, Bartolomeo, 1613.

[22] Euclid Author, Of Alexandria Contributor Hypsicles, and Giovanni Mocenigo. Euclid’s ”Elements”. Venice: Erhard Ratdolt, -05-25, 1482. Pdf. Retrieved from the Library of Congress, www.loc.gov/item/2021667076.