VERIFICATION OF DECIMAL FLOATING-POINT OPERATIONS

by

Amr Abdel-Fatah Ramdan Sayed-Ahmed

A Thesis Submitted to the
Faculty Of Engineering at Cairo University
in Partial Fulfillment of the
Requirements for the Degree of
MASTER OF SCIENCE
in
ELECTRONICS AND COMMUNICATIONS

FACULTY OF ENGINEERING, CAIRO UNIVERSITY,
GIZA, EGYPT
2011.

VERIFICATION OF DECIMAL FLOATING-POINT OPERATIONS

by

Amr Abdel-Fatah Ramdan Sayed-Ahmed

A Thesis Submitted to the
Faculty Of Engineering at Cairo University
in Partial Fulfillment of the
Requirements for the Degree of
MASTER OF SCIENCE
in
ELECTRONICS AND COMMUNICATIONS

Under the Supervision of

Hossam. A. H. Fahmy

Associate Professor
Elec. And Com. Dept.

FACULTY OF ENGINEERING, CAIRO UNIVERSITY,
GIZA, EGYPT
2011.

VERIFICATION OF DECIMAL FLOATING-POINT OPERATIONS

by

Amr Abdel-Fatah Ramdan Sayed-Ahmed

A Thesis Submitted to the
Faculty Of Engineering at Cairo University
in Partial Fulfillment of the
Requirements for the Degree of
MASTER OF SCIENCE
in
ELECTRONICS AND COMMUNICATIONS

Approved by the
Examining Committee:

Associate. Prof. Dr: Hossam A. H. Fahmy, Thesis Main Adviser.

--

Prof. Dr : Ashraf M. Salem, Member.

Associate. Prof. Dr: Ibrahim M. Qamar, Member.

FACULTY OF ENGINEERING, CAIRO UNIVERSITY,
GIZA, EGYPT
2011.

Contents

List of Figures

List of tables

Acknowledgments

The thesis is a part from the project “Promoting Egypt as the First Decimal Arithmetic Intellectual Property Cores Provider for Financial Applications in the World” (grant number C2/S1/163) funded by the RDI programme through the EU Egypt Innovation Fund (EEIF). The RDI programme is a program of the Egyptian Ministry of Higher Education and Scientific Research funded by the European Union.

I want to thank Dr. Hossam. A. H. Fahmy for his favors in all what I did. Many Thanks to Dr. Mike Cowlishaw and SilMinds engineers for their cooperation and their acceptance to publish the verification results.

Abstract

Decimal floating-point designs require a verification process to prove that the design is in compliance with the IEEE Standard for Floating-Point Arithmetic (IEEE Std 754-2008). Our work is a decimal floating-point verification using simulation based verification, which a simulation method based on coverage models to cover corner cases of a certain decimal floating-point operation. Our work represents five engines, the first engine for the verification of decimal addition-subtraction operation, the second for the verification of decimal multiplication operation, the third for the verification of decimal fused-multiply-add operation, the fourth for the verification of decimal square root operation, and the fifth for the verification of decimal division operation. Each engine solves constraints describing corner cases of the operation, and generates test vectors to verify these corner cases in the tested design. We also represent the coverage models of each operation solved by the engines. The generated test vectors have discovered bugs in commercial hardware designs reported and in commercial software designs reported. The verification of decimal fused-multiply-add operation and the verification of decimal square root operation are the first published work.

Chapter 1

Introduction

Decimal floating-point implementations perform the arithmetic operation using the numbers in base ten. Decimal floating-point implementations as software or hardware based designs have many advantages over binary floating-point especially in the financial and commercial applications. Simple decimal fractions such as 1/10 which might represent a tax amount or a sales discount yield an infinitely recurring number if converted to a binary representation. Hence, a binary number system with a finite number of bits cannot accurately represent such fractions. When an approximated representation is used in a series of computations, the final result may deviate from the correct result expected by a human. In a large billing application such an error may be up to $5 million per Year[7].

As decimal floating-point is newly defined in the IEEE Standard for Floating-Point Arithmetic (IEEE Std754-2008)[21], new verification technologies are needed to verify the compliance of the decimal floating-point designs with the standard.

As most applications (from aircraft control systems to weather forecasting) use floating-point approximation, and these applications are often used in monitoring and controlling physical systems, the consequence of bugs in the result of these applications can be catastrophic. An example is the destruction of Ariane 5 rocket after the take off in 1996, owing to uncaught floating-point exception. Also, the costly and embarrassing error of Intel in the floating-point division instruction of some early Intel Pentium processors in 1994. Intel set aside approximately $475M to cover costs arising from this issue [10].

An amount of effort has been applied on the formal verification of binary floating-point, in Intel[12], AMD[14], and IBM[17], and on the simulation based verification of binary floating-point in IBM [2,3,9,19,20].

The verification of decimal floating-point using simulation based verification [1,8] was recently presented but the proposed algorithms do not guarantee to find the solution of certain cases. They cannot solve simultaneous constraints on inputs and the intermediate result, and cannot solve constraints on an unbounded intermediate result. Also there are no algorithms before our own research to solve constraints of the FMA and the square root operations. Furthermore, there is no previous work in the formal verification of decimal floating-point.

1.1 Formal Verification

The hardware design starts with high-level specifications, formal verification uses mathematical methods to verify that the design meets all or parts of its specification. The main idea of formal hardware verification is to prove the function correctness of the design which the design simulation using test vectors cannot do.

There are two formal verification scenarios: (1) Equivalence Checking to make sure the equivalence of two given circuit descriptions by translating both of them to an internal format and establishing the correspondence between both of them in a matching phase, (2) Model checking (property checking) where a given circuit and its properties are formulated to a given verification language, then it is proven that all properties hold under all circumstances.

Formal verification has a lot of difficulties with arithmetic circuits using normal techniques like Binary Decision Diagram (BDD) or Boolean Satisfiability Problem (SAT) [5]. Word-level approaches (such as Binary Moment Diagram (BMD), Hybrid Decision Digram (HDD), etc.) have been used, but it is often difficult to integrate in a fully arithmetic tool [5]. The normal techniques represent the circuit in binary states which cause the state explosion problem with the arithmetic circuits while the word approaches represent the circuit in high level states.

1.2 Simulation based Verification

Another approach to the verification is simulation based verification, which is a simulation method based on coverage models to verify corner cases of decimal or binary floating-point operations.

The approach represents the specifications of a certain floating-point operation in terms of constraints on the inputs, the output, and some internal signals of the operation. Each specification has a coverage model, the coverage model consists of tasks, each task represents the constraints of a certain case from the cases that test this specification. These constraints are solved by an engine that generates a test vector to verify the case in a decimal floating-point design using simulation. The coverage model is a set of related tasks targeting a certain floating point area or features of the floating-point operation, and it is defined using a Cartesian product between two lists or among more lists of constraints while ignoring the impossible combinations.

Simulation based technique can be applied regardless of the state space size, and can be quite effectively in discovering bugs, but it cannot prove the absence of bugs, because it expresses the specifications in terms of some signals of the implementation. On other hand, Formal techniques can prove the absence of bugs in an implementation, because they prove that all the specification properties hold under all circumstances of the implementation states. However, they require a significant investment in the machines and manual work time, and are limited to small defined implementation fragments.

In verification of decimal floating-point, IBM has developed its verification tool FPgen [3] to verify the decimal FP implemented in millicode in IBM System Z9 [6] and in the verification of decimal FP hardware in IBM power6. It uses the simulation based verification in the verification of decimal and binary floating-point unites.

FPgen uses multiple engines in solving constraints. It has two types of engines, (1) Analytical engines, which are based on mathematical algorithms and guaranteed to find the solution in a reasonable amount of time. (2) Search engines, which are based on search methods and do not guarantee to find the solution in a reasonable amount of time. Since the search engines may not find the solution, although one may exit. The search engines are used when the analytical engines cannot solve the constraints and generate test vectors.

According to [1], FPgen decimal mathematical algorithms (1) may not be suitable for some corner cases (eg. When the inputs are subnormal numbers), (2) they cannot solve simultaneous constraints on inputs and the intermediate result, and cannot solve constraints on the unbounded intermediate result, (3) there are no algorithms to solve constraints of the FMA and the square root operations. FPgen coverage models are described in [22].

1.3 Our Verification Work

Our decimal floating-point verification method is simulation based verification, which a simulation method based on coverage models to cover all corner cases of a certain decimal floating-point operation. The method guarantees that the simulation covers the interesting cases of the operation. On the other hand the random simulation does not guarantee a good coverage due to the large space of the inputs that is equal to $10^{(n*p)}$. Where $(p=16 \vee p=34)$ is the maximum number of digits in each operand for IEEE 745-2008 decimal FP formats, and n is the number of the operation operands.

We represent the standard specifications of each operation(eg: Overflow, Underflow, Rounding, ...) as coverage models using the models generation block as shown in Figure 1, which is a C++ code that generates the tasks of each model. The behavior of the models generation block of each operation is explained in the next chapters under the title “The main ideas of the operation models”.

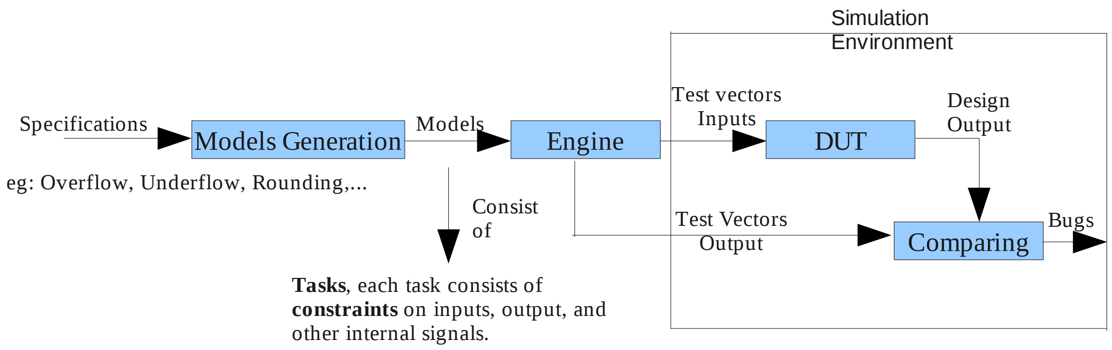

Figure 1. Our Verification Work Environment for DUT(Design Under Test)

The constraints of each task is solved using a software engine that takes a task as input and generates a test vector as output. The test vector consists of value of the input operands of the operation and the output of the operation compliant with the standard.

The test vectors are used to verify the different implementations of the operation using simulation. The simulation environment is determined according to the type of the design implementation, as shown in Figure 1, it enters the test vector inputs to the design implementation and compares the output of the design implementation with the output of the test vector, if there is a mismatching, it is a bug in the design implementation.

The test vectors are represented as ASCII characters, the syntax of the test vectors is the IBM syntax which is explained in Appendix A. The simulation tools of system on chip designs read the test vectors encoded based on DPD (Densely Packed Decimal) decimal floating-point, or based on BID (Binary Integer Decimal) decimal floating-point [21]. Therefore, free software tools like the tool in [7] are needed to encode the test vectors. While, we test the software implementation designs of the decimal floaing-point libraries, using the generated test vectors directly, without encoding.

The Addition-Subtraction, Multiplication, Fused-Multiply-Add (FMA), Square root, and Division engines are our software engines to solve constraints on inputs, intermediate result, and specific features related to the operation. Each engine uses algorithms allowing the engines to solve all the constraints numerically including simultaneous constraints on inputs and the intermediate result, and constraints on the unbounded intermediate result. The engines find the solution of most cases if the solution exits, the cases that the engines may not solve it, will be explained in the next chapters.

The fives engines are used for the verification of SilMinds decimal floating-point hardware implementations[7,13,15], and research decimal floating-point designs at Cairo university[18]. The generated test vectors have proven the efficiency of the engines in discovering bugs in the different operations. The generated test vectors also have discovered bugs in the FMA and the square root operations of the DecNumber library from IBM (Decimal floating-point library used in gcc)[23].

1.4 Main Definitions

The FP standard [21] defines, the precision p as the maximum number of digits in the significand. $emax$ is the maximum exponent, and $emin=1-emax$ is the minimum exponent.

In our work, decimal floating-point numbers are represented in the unnormalized format. A number is defined as $(-1)^s(d_{P-1}d_{P-2}d_{P-3}...d_0)10^q$ where s is the sign, $d_{P-1}d_{P-2}\cdots d_0$ is the significand where $d_i\in\{0,1,\cdots,9\}$, and the exponent is bounded by $qmin\leq q\leq qmax$, where $qmax=emax-p+1$ and $qmin=emin-p+1$.

We define a “mask” for a number of digits as all the possible values that such digits may take. For the minimum values we use the subscript N while the maximum values have The subscript M. For example, the mask of p digits significand $d_{P-1}d_{P-2}\cdots d_0$ represents the minimum and the maximum of each digit in the significand. If $0\leq d_i\leq 9$ then the mask consists of two numbers, the first number represents the minimum absolute values of each digit in the significand $d_{N_{P-1}}d_{N_{P-2}}\cdots d_{N_0}=00\cdots 0$ and the second number represents the maximum absolute values of each digit in the significand $d_{M_{P-1}} d_{M_{P-2}} \cdots d_{M_0} = 99\cdots9$. If in another case there is a constraint on d_0 to be exactly *5* then $d_{N_0} = d_{M_0} = 5$ and the remaining digits may take any values from *0* to *9*, then the mask is $d_{N_{P-1}} \cdots d_{N_1} d_{N_0} = 0\cdots05$ to $d_{M_{P-1}} \cdots d_{M_1} d_{M_0} = 9\cdots95$.

The intermediate result is the result of the operation when the precision of the significand or the exponent is unbounded; i.e. the result before the rounding or the normalization processes.

The Rounding mode is one from five modes defined in the standard : Round ties to even, Round ties to away, Round toward zero, Round toward positive, and Round toward negative. We do the rounding process to all the digits that follow a point called fractional point, to the right of the digit d_0.

The fused-multiply-add (FMA) operation is a multiplication operation followed by an addition-subtraction operation. The addition intermediate result is the result of the addition-subtraction operation when the precision of the significand or the exponent is unbounded, and the multiplication intermediate result is the result of the multiplication operation when the precision of the significand or the exponent is unbounded.

All input types list is a list from the standard types [21], which are Normal numbers, Zeros, Subnormal numbers, Infinities, quiet NaN (qNaN), and signaling NaN (sNaN).

1.5 Thesis layout

In each of the following chapters, we represent the main steps of the engine for one operation and the coverage models that have been solved by that engine.

Chapter 2 discusses the addition-subtraction while chapter 3 explains the multiplication. The engines and the models presented for these two operations are compared to the previous research.

Chapter 4 presents the main steps of the FMA, and chapter 5 deals with the square root. To our knowledge this the first published work on these two operations.

Finally, chapter 6 describes the division, and chapter 7 concludes the work.

1.6 Publications out of This Work

1. A. Sayed-Ahmed, H. A. H. Fahmy, M. Y. Hassan, "Three Engines to Solve Verification Constraints of Decimal Floating-Point operations," in Forty-Four Asilomar Conference on Signals, Systems, and Computers, Nov 2010.

2. A. Sayed-Ahmed, Hossam. A. H. Fahmy, R. Samy "Verification of Decimal Floating-Point Fused-Multiply-Add Operation," in The ACS/IEEE International Conference on Computer Systems and Applications (AICCSA), Egypt, 2011.

Chapter 2

Engine and Models of Decimal Addition-Subtraction Operation

The addition-subtraction engine is a software tool, generates addition -subtraction test vectors to cover corner cases that verify the compliance of software or hardware implementations of the decimal floating-point addition-subtraction operation with the IEEE standard (754-2008) for Floating Point Arithmetic, it takes coverage models as inputs and generates test vectors as outputs.

The addition-subtraction engine solved the coverage models one time and generated about 136000 test vectors in Decimal64, the test vectors have proved their efficiency by discovering bugs in Silminds design[7].

The generated test vector is a decimal vector that has five sets. The first set is type of the operation (add or subtract), number of the precision (64 or 128), and the rounding mode. The second set is sign, significand, and exponent of the first input. The third set is sign, significand, and exponent of the second input. The fourth set is sign, significand, and exponent of the output. Finally the fifth set is one or two of three flags(invalid, inexact, and overflow). The simulation enviroment enters the first three sets to the implementation and verifies the implementation output against the last two sets.

The task given to the addition-subtraction engine is the set of constraints on six elements, (1) the significand of the first input Sx that is set as the smaller exponent input, (2) the significand of the second input Sy that is set as the larger exponent input, (3) the significand of the intermediate result Sz, (4) the right shift value to significand of the smaller exponent input, (5) the intermediate result exponent at which the addition_subtraction operation occurs, and (6) the rounding mode.

The constraint on Sx is a mask starting from the minimum number Nx to the maximum number Mx. The constraint on Sy is a mask starting from the minimum number Ny to the maximum number My. Each number in the previous masks has p digits. Similarly, the mask on Sz consists of two numbers Nz and Mz, each number has $2p+1$ digits, $p+1$ digits before the fractional point and p digits after it. The addition intermediate result exponent and the rounding direction are either given explicitly in the task or left to the engine to choose randomly.

The ability of the engine to choose randomly within the range of the mask or to choose the intermediate result exponent and the rounding direction empowers the engine to generate test vectors discovering more bugs.

An example to explain the format of the decimal addition-subtraction task at $p=16$ is as follows:

$$64+T: \quad -1 \;\; -9999999999999999 \quad -1000000000000000 \;\; -9999999999999999$$
$$-9999999999999999\text{p}9000000000000000 \quad -9999999999999999\text{p}9999999999999999$$
$$R \quad R \quad 4$$

This multiplication task means that $Nx=-1$, $Mx=-9999999999999999$, $Ny=-1000000000000000$, $My=-9999999999999999$, $Nz=-9999999999999999\text{p}9000000000000000$ $Mz=-9999999999999999\text{p}9999999999999999$. Also, it means that the engine chooses randomly the right shift value, and the exponent of the intermediate result, while the rounding mode is(Round to Negative).

One of the solutions of this task is the test vector d64- < $-2837171276486938\text{E}137$ $+9997162828723513\text{E}140$ -> $-1000000000000000\text{E}141$ X.

The *d64* means decimal64, the - means subtraction operation, the following < means that the rounding mode is Round to Negative, the first input is $x=-2837171276486938*10^{137}$, the second input is $y=+9997162828723513*10^{140}$, the rounded result is $z=-1000000000000000*10^{141}$, and the following X indicates that the inexact flag is high, because the exact result is

$-9999999999999999.938 * 10^{140}$. The rounding mode causes a carry in the intermediate result and increases the exponent by one.

2.1 The Addition-Subtraction Engine

The engine determines the number of digits of the first input significand p_x from the interval $[no\ of\ digits\ of\ Nx, no\ of\ digits\ of\ Mx]$, and number of digits of the second input significand p_y from the interval $[no\ of\ digits\ of\ Ny, no\ of\ digits\ of\ My]$.

The engine chooses randomly the right-shift value to the significand of the smaller exponent input sr_x either from the interval $[1, p]$ or from the interval $[p+1, qmax-qmin]$. If sr_x is equal zero, it will choose randomly left-shift value to the significand of the larger exponent input sl_y from the interval $[0, p-p_y]$, otherwise if sr_x is larger than zero, sl_y is equal to $p-p_y$. Then, it shifts to left both Ny and My, with the value of sl_y, and shifts to right both Nx and Mx, with the value of sr_x.

After the shifting process, the engine uses the Addition Algorithm to get the first input significand Sx, the second input significand Sy, and the intermediate result significand Sz. After getting the signifigands, the engine shifts Sx to the left with value of sr_x, and shifts to right Sy with a value of sl_y.

The engine gets the input exponents and the result exponent that achieve the right shift sr_x and the left shift sl_y. The intermediate result exponent Ez either has explicit value or is chosen using $q_{min}+sr_x \le Ez \le q_{max}-sl_y$. The first input exponent is calculated using $Ex=Ez-sr_x$, and the second input exponent is calculated using $Ey=Ez+sl_y$.

In the case that, the intermediate result significand has cancellation digits and sr_x is larger than zero, the engine shifts Sz to left and decreases Ez with the value $scn=min(sr_x, p-no\ of\ digits\ before\ point)$.

In the case that, the intermediate result significand has a carry digit, the engine shifts Sz one digit to the right and increases Ez by one.

The engine rounds the intermediate result according to the standard. The rounding process may generate a carry, which forces the engine to shift Sz one digit to the right and increase Ez by one.

In the case that, Ez is larger than $qmax$, it is an overflow case, its result is according to the rounding mode.

2.1.1 The Addition Algorithm

The algorithm is based on solving the linear equations that can be estimated from Figure 2, where each column represents one linear equation. The figure shows the addition of the two input significands at $p=8$, where Sx_i denotes the first input significand digit of weight 10^i, Sy_i denotes the second input significand digit of weight 10^i, and Sz_i denotes the intermediate result significand digit of weight 10^i.

$$
\begin{array}{ccccccccccccccc}
+ & Sx_7 & Sx_6 & Sx_5 & Sx_4 & Sx_3 & Sx_2 & Sx_1 & Sx_0 & Sx_{-1} & Sx_{-2} & Sx_{-3} & Sx_{-4} & Sx_{-5} & Sx_{-6}\cdots \\
 & Sy_7 & Sy_6 & Sy_5 & Sy_4 & Sy_3 & Sy_2 & Sy_1 & Sy_0 & Sy_{-1} & Sy_{-2} & Sy_{-3} & Sy_{-4} & Sy_{-5} & Sy_{-6}\cdots \\
\hline
Sz_8 & Sz_7 & Sz_6 & Sz_5 & Sz_4 & Sz_3 & Sz_2 & Sz_1 & Sz_0 & Sz_{-1} & Sz_{-2} & Sz_{-3} & Sz_{-4} & Sz_{-5} & Sz_{-6}\cdots
\end{array}
$$

Figure 2. The Addition of two Input Significands assuming Precision 8

The algorithm iterates to solve the linear equations from left to right. As shown in Figure 2, the first linear equation from left is $Sz_7-Sx_7-Sy_7=br_7$ where br_7 is the value of carries that transfer from the previous weights to the weight of 10^7, or the borrow generated from this weight to lower weights. The second and the third linear equations are $Sz_6+10*br_7-Sx_6-Sy_6=br_6$ and $Sz_5+10*br_6-Sx_5-Sy_5=br_5$. In general the linear equation for the column of index n is:

$$br_n=Sz_n+10*br_{n+1}-Sx_n-Sy_n. \tag{2.1}$$

To start the solution, the algorithm attempts to solve the first three linear equations (representing columns 7 to 5) together based on the range of carries that may transfer from the next lower significant column. The algorithm chooses the digits Sz_8, Sz_7, and Sz_6 randomly from their intervals, and replaces Sz_7 with $Sz_7+10*Sz_8$. Then since the ranges of borrow digit br_5, the digit Sx_5, and the digit Sy_5 are known as $(Nx_4+Ny_4-Mz_4)/10 \le br_5 \le (Mx_4+My_4-Nz_4)/10$, $Nx_5 \le Sx_5 \le Mx_5$, and $Ny_5 \le Sy_5 \le My_5$. The algorithm transforms the third linear equation to the inequality condition:

$$\frac{(Nx_4+Ny_4-Mz_4)}{10}+Nx_5+Ny_5 \le Sz_5+10*br_6 \le Mx_5+My_5+\frac{(Mx_4+My_4-Nz_4)}{10}. \qquad (2.2)$$

Finally, it searches randomly on the combination values of Sx_7, Sx_6, Sy_7, Sy_6, Sz_5 that satisfy the first linear equation, the second linear equation and the Inequality 2.2 . The steps taken so far constitute the first outer iteration that gets the final values of Sx_7, Sy_7, Sz_8, Sz_7, Sz_6, Sz_5 and estimates the values of Sx_6, Sy_6 that may be refined in the following iteration. In the second iteration, the algorithm transforms the fourth linear equation $Sz_4+10*br_5-Sx_4-Sy_4=br_4$ to the inequality:

$$\frac{(Nx_3+Ny_3-Mz_3)}{10}+Nx_4+Ny_4 \le Sz_4+10*br_5 \le Mx_4+My_4+\frac{(Mx_3+My_3-Nz_3)}{10},$$

and searches randomly on the values of Sx_6, Sx_5, Sy_6, Sy_5, Sz_4 that achieve the second linear equation, the third linear equation and the inequality condition, where the digits Sx_7, Sy_7, br_6, Sz_7, Sz_6, Sz_5 are known from the previous iteration. The algorithm does this procedure in all the iterations and gets all digits of Sx, Sy, and Sz.

In general, the algorithm gets randomly the digits Sz_p, Sz_{p-1}, and Sz_{p-2}, from their intervals, and replaces Sz_{p-1} with $Sz_{p-1}+10*Sz_p$. It does several iterations of index i, from $i=p-1$ to $i=-p$, to get in each iteration the digits Sx_i, Sy_i, Sz_{i-2}, and estimates the digits Sx_{i-1}, Sy_{i-1}, such that the combination values of these digits achieves the general two linear equations and the inequality condition. The general form of the two linear equations and the inequality condition are:

$$br_i=Sz_i-Sx_i-Sy_i \qquad (2.3)$$

$$br_{i-1}=Sz_{i-1}+10*br_i-Sx_{i-1}-Sy_{i-1} \qquad (2.4)$$

$$\frac{(Nx_{i-3}+Ny_{i-3}-Mz_{i-3})}{10}+Nx_{i-2}+Ny_{i-2}\leq Sz_{i-2}+10*br_{i-1}\leq Mx_{i-2}+My_{i-2}+\frac{(Mx_{i-3}+My_{i-3}-Nz_{i-3})}{10}. \quad (2.5)$$

2.2 The Main Ideas of the Addition-Subtraction Models

The models are defined using a Cartesian product between two or more lists of constraints with ignoring the impossible combinations, and allowing the other constraints to be chosen randomly.

All the model proposal ideas are in [22], except the ideas of the carry and borrow model. However we describe all the ideas in the form of our engine constraints.

A) Inputs Types Model

The model aims to verify all possible combinations of the input types. The proposal ideas of the model are in [22]. We separate the model into three sub-models as follows:

1. It verifies the design when one of the inputs is Zero using, (1) a list of the first input significand is equal to zero, (2) a list of the first input exponent from the interval $[qmin, qmax]$, (2) all input types list of the second input.

2. It verifies the design when one of the inputs is Infinity, sNaN, or qNaN using, (1) a list of the first or the second input from the Infinities, sNaN, and qNaN, (2) all input types list of the other input.

3. It verifies the design in solving the other input types using, (1) a list of the first or the second input from the minimum Subnormal, the maximum Subnormal, the minimum Normal, and the maximum Normal, (2) a list of the other input exponent from the interval $[qmin, qmax]$.

B) Result Types Model

The model aims to verify the ability of the design to generate different types of the final result. The proposal ideas of the model are in [22]. We separate the model into five sub-models as follows:

1.It verifies all the result exponents using, (1) a list of the intermediate result exponent from the interval $[qmin, qmax]$, (2) a list of right shift from the intervals $\{0, [1, p], [p+1, qmax-qmin]\}$.

2.It verifies the generation of the first hundred Subnormal numbers, the last hundred Subnormal numbers, and the first hundred Normal numbers using, (1) the intermediate result exponent is equal to $qmin$, (2) a list of the intermediate result significands from the intervals $\{[2,100], [10^{p-1}-100, 10^{p-1}+100]\}$.

3.It verifies the generation of numbers from One to 100 using, (1) the intermediate result exponent is equal to zero, (2) a list of the intermediate result significands from the interval $[1,100]$.

4. It verifies the last hundred Normal numbers using, (1) the intermediate result exponent is equal to $qmax$, (2) a list of the intermediate result significand from the interval $[10^{p}-100, 10^{p}-1]$.

5. It verifies the generation of Zero result due to cancellation at the effective subtraction operation using, (1) the intermediate result significand is equal to zero due to cancellation, (2) a list of the intermediate result exponent from the interval $[qmin, qmax]$.

C) Rounding Model

The model aims to verify the rounding process. The proposal ideas of the model are in [22]. We separate the model into three sub-models as follows:

1. It verifies the rounding process using, (1) a list from the five rounding modes, (2) a list of intermediate result significand that consists of the cross product of the guard digit interval $[0,9]$, the least significand digit interval $[0,9]$, the sticky bit interval $[0,1]$.

2.It verifies the possible carry propagation due to rounding process using, (1) a list from the five rounding modes, (2) a list of intermediate result significand from the cross product of the guard digit interval $[0,9]$, the sticky bit interval $[0,1]$, and the patterns $\{\overbrace{99\cdots9}^{p}, \overbrace{\{0-8\}9\cdots9}^{p}, \overbrace{X\{0-8\}9\cdots9}^{p}, \cdots, \overbrace{XX\cdots X\{0-8\}}^{p}\}$. (3) a list of the intermediate result exponent that consists of $\{qmax, emin, random\,number\}$.

3. It verifies the sticky bit calculations using, (1) a list of right shift from the interval $[1, qmax-qmin]$, (2) number of digits list of the smallest exponent input significand that consists of $\{1, random\,number\}$.

D)Shift Model

The model aims to verify all the possible shifting of the input significands. The proposal ideas of the model are also in [22].

1. It verifies the possible shifting to the input significands using, (1) a list of left shift values of the largest exponent input from the interval $[0, p-1]$, (2) a list of right shift values to the smallest exponent input from the interval $[0, qmax-qmin]$.

E) Trailing and Leading Zeros Model

The model verifies all the possible trailing and leading zeros in the input siginficands and the intermediate result significand. The proposal ideas of the model are also in [22]. We separate the model into three sub-models as follows:

1.It verifies all possible trailing and leading zeros in the input significands using, (1) a list of the first input significand, (2) a list of the second input significand same like previous list, that consists of the patterns

$$\begin{gathered}
\overbrace{\{1-9\}00\cdots00}^{P}, \overbrace{0\{1-9\}00\cdots00}^{P}, \cdots, \overbrace{00\cdots0\{1-9\}}^{P} \\
\overbrace{\{1-9\}\{1-9\}0\cdots00}^{P}, \overbrace{0\{1-9\}\{1-9\}0\cdots00}^{P}, \cdots, \overbrace{00\cdots0\{1-9\}\{1-9\}}^{P} \\
\overbrace{\{1-9\}X\{1-9\}0\cdots00}^{P}, \overbrace{0\{1-9\}X\{1-9\}0\cdots00}^{P}, \cdots, \overbrace{00\cdots0\{1-9\}X\{1-9\}}^{P} \\
\vdots \\
\overbrace{\{1-9\}XX\cdots X\{1-9\}}^{P}
\end{gathered}$$

2.It verifies all possible trailing and leading zeros in the intermediate result significand using, (1) a list of the intermediate result sigificand similar to the previous list, (2) right shift value is equal to zero.

3.It verifies the last carry in the intermediate result significand using, (1) the right shift from the interval $[0, p-1]$, (2) a list of the intermediate result sigificand from the patterns

$$
\begin{gathered}
\overbrace{1\{1-9\}00\cdots00}^{p+1},\overbrace{10\{1-9\}00\cdots00}^{p+1},\cdots,\overbrace{100\cdots0\{1-9\}}^{p+1},\overbrace{100\cdots00}^{p+1} \\
\overbrace{1\{1-9\}\{1-9\}0\cdots00}^{p+1},\overbrace{10\{1-9\}\{1-9\}0\cdots00}^{p+1},\cdots,\overbrace{100\cdots0\{1-9\}\{1-9\}}^{p+1} \\
\overbrace{1\{1-9\}X\{1-9\}0\cdots00}^{p+1},\overbrace{10\{1-9\}X\{1-9\}0\cdots00}^{p+1},\cdots,\overbrace{100\cdots0\{1-9\}X\{1-9\}}^{p+1} \\
\vdots \\
\overbrace{1XX\cdots X\{1-9\}}^{p+1}
\end{gathered}
$$

F) Cancellation Model

The model verifies the cancellation digits in the intermediate result significand when the operation is effective subtraction. The proposal ideas of the model are also in [22]. We separate the model into three sub-models as followss:

1. It verifies all possible number of the cancellation digits using, (1) a list of number of digits of the intermediate result significand from the interval $[1, p]$, (2) a list of right shift from the interval $[0,1]$, (3) a list of left shift from the interval $[0, p-1]$.

2. It verifies the cancellation case at the other values of right shift using, (1) One cancellation digit in the intermediate result significand, (2) a list of the right shift from the interval $[2, qmax-qmin]$, (3) a list of left shift from the interval $[0, p-1]$.

3.It verifies the cases of Subnormal result due to cancellation using, (1) a list of number of digits of the intermediate result significand from the interval $[1, p]$, (2) a list of right shift from interval $[0, intermediate\,result\,exponent - qmin]$, (3) a list of left shift from the interval $[0, p-1]$, (4) a list of the intermediate result exponent from the interval $[qmin, emin]$.

G) Overflow Model

The model verifies the overflow cases. The proposal ideas of the model are also in [22]. We separate the model into three sub-models as follows:

1. It verifies the overflow cases due to the final carry at the effective addition operation using, (1) the intermediate result exponent is equal to $qmax$, (2)the intermediate result significand has a carry digit that is equal to one, (3) a list of right shift from the interval $[0, p-1]$, (4) a list of left shift from the interval

$[0, p-1]$.

2. It verifies the overflow cases due to the rounding process using, (1) the intermediate result exponent is equal to $qmax$, (2) the right shift value is equal to p, (3) a list of the intermediate result significand that consists of the guard digit interval $[5,9]$, (4) a list from two rounding modes Round ties to even and Round ties to away, (5) the significand of the largest exponent input is equal to $10^p - 1$.

3. It verifies also the overflow cases due to the rounding process using, (1) the intermediate result exponent is equal to $qmax$, (2) a list of right shift from the interval $[p+1, qmax-qmin]$, (3) a list from two rounding modes, Round toward positive and Round toward negative, (4) the significand of the largest exponent input is equal to $10^p - 1$.

H) Carry and Borrow Model

The model verifies all the possible propagations of carries and borrows that occur during the effective addition or effective subtraction operations. The proposal ideas of the model are all new. We separate the model into two sub-models as follows:

1. It verifies all patterns of the borrow propagation at the effective subtraction operation using, (1) a list of right shift values from the interval $[1, p]$, (2) a list of the largest exponent input significand that consists of the patterns

$$\begin{gathered}
\overbrace{\{1-9\}00\cdots0\,X}^{p}, \overbrace{\{1-9\}00\cdots0\mathrm{XX}}^{p}, \cdots, \overbrace{\{1-9\}X\cdots XX}^{p} \\
\overbrace{X\{1-9\}0\cdots0\,X}^{p}, \overbrace{X\{1-9\}0\cdots0\mathrm{XX}}^{p}, \cdots, \overbrace{X\{1-9\}X\cdots XX}^{p} \\
\overbrace{X\,X\{1-9\}0\cdots0\,X}^{p}, \overbrace{X\,X\{1-9\}0\cdots0\mathrm{XX}}^{p}, \cdots, \overbrace{X\,X\{1-9\}X\cdots XX}^{p} \\
\vdots \\
\overbrace{XXX\cdots X\{1-9\}}^{p}
\end{gathered}$$

2. It verifies all patterns of the carry propagation at the effective addition operation using, (1) a list of right shift values from the interval $[1, p]$, (2) a list of the largest exponent input significand that consists of the patterns

$$
\overbrace{\{1-9\}99\cdots99}^{p},\overbrace{\{1-9\}99\cdots99\mathrm{X}}^{p},\overbrace{\{1-9\}99\cdots9\mathrm{XX}}^{p},\cdots,\overbrace{\{1-9\}X\cdots XX}^{p}
$$

$$
\overbrace{X\{1-9\}99\cdots99}^{p},\overbrace{X\{1-9\}99\cdots99\mathrm{X}}^{p},\overbrace{X\{1-9\}99\cdots9\mathrm{XX}}^{p},\cdots,\overbrace{X\{1-9\}X\cdots XX}^{p}
$$

$$
\overbrace{XX\{1-9\}99\cdots99}^{p},\overbrace{\{1-9\}99\cdots99\mathrm{X}}^{p},\overbrace{XX\{1-9\}99\cdots9\mathrm{XX}}^{p},\cdots,\overbrace{XX\{1-9\}X\cdots XX}^{p}
$$

$$
\vdots
$$

$$
\overbrace{XXX\cdots X\{1-9\}}^{p}
$$

2.3 Previous work

The Fpgen addition-subtraction algorithm by IBM [1] is given a specific intermediate result and the difference d between the actual and the preferred exponents, to provide two inputs that yield the specified result. The algorithm denotes the addend significand with the smaller exponent by S_x and the addend significand with the larger exponent by S_y, and the significand of the intermediate result is denoted by S_z.

The algorithm divides the problem into four sub cases :

Case 1: The result is exact and the actual exponent is equal to the preferred exponent, the algorithm selects random S_x less than S_z and calculates $S_y = S_z - S_x$, where the exponents of them same like the intermediate result exponent. Next, it selects the operand that has possible shift right or left according to the leading or the trailing zeros of the operand, and select one of possible shifting.

Case 2: The result is exact and the actual exponent differs from the preferred exponent, the algorithm tests, if there is carry or not, where carry is possible if $10^{p-1} \le S_z/10 \le 10^{p-1} + 10^{p-d} - 2$.

If there is no carry, it chooses $S_x/10^d \le S_z - 10^{p-1}$ that has d trailing zeros, and subtracts it from $\bar{S}_z$ to get $\bar{S}_y$, that has p digits. If there is a carry, it chooses S_x using $10S_z - 10^p \le S_x/10^{d-1} \le min(10^{p-d+1} - 1, 10S_z - 10^{p-1})$ that has at least $d-1$ trailing zeros, then computes $\bar{S}_y = \bar{S}_z - S_x$, such that $\bar{S}_z$ has $p+d$ digits and $\bar{S}_y$ has $p+d-1$ digits.

Case 3: The result is inexact but the sticky bit is zero, and $d > 0$. In this case,

$\bar{S}_z$ has $p+d$ digits including $d-1$ digits. According to the carry condition, if there is no carry, $\bar{S}_y$ has at least d trailing zero, the algorithm chooses S_x using $S_z-10^{p+1}\le S_x/10^{d-1}<10^{p-d}$, and computes $\bar{S}_y=\bar{S}_z-S_x$. Otherwise, if there is a carry, $\bar{S}_y$ has at least $d-1$ trailing zeros, and the algorithm gets S_x, and $\bar{S}_y$ as before.

Case 4: The result is inexact, the sticky bit is one, and $d\ge 2$,there are three sub-cases:

1. At $d>p$ and the guard digit is equal to zero, the algorithm separates $\bar{S}_z$ that has $p+d$ digits into three substrings, the head of digits of $\bar{S}_z$ is assigned to S_y, the tail of digits is assigned to S_x, and in middle there are zero digits.

2. At $d=p$, if $\bar{S}_y$ has the same digits as $\bar{S}_z$, the algorithm solves this case as the previous case. Otherwise, the addition operation has a carry which occurs at $S_y=\overbrace{99\cdots 9\text{y}}^{p}$, $S_z=\overbrace{100\cdots 0}^{p+1}z\cdots z$, and the most significant digit of S_x is greater than the guard digit.

3.At $d<p$, if $\bar{S}_y$ has the same digits as $\bar{S}_z$, the algorithm chooses S_x using $\bar{S}_z-10^{p+d}<S_x\le min(10^p-1,\bar{S}_z-10^{p+d-1})$. Otherwise it chooses S_x using $\bar{S}_z-10^{p+d-1}<S_x\le 10^p$, and computes $\bar{S}_y=\bar{S}_z-S_x$.

2.4 Comparison

The Fpgen addition-subtraction algorithm divides the operation into cases and sub-cases and uses different inequalities to each one. Our engine uses one procedure to solve all the cases which are based on the values of right shift to the smaller exponent input significand and the values of left shift to the larger exponent input significand. Our engine can solve all the simultaneous constraints on the inputs and the unbounded intermediate result using the Addition Algorithm, while Fpgen addition-subtraction algorithms solve the simultaneous constraints on the inputs and the final result, also they solve constraints on the intermediate result.

The value of the Addition Algorithm will appear clearly in the fused-multiply-add(FMA) as shown in chapter 4.

2.5 Summary

This chapter represents the main steps that the addition_subtraction engine uses to solve all the constraints numerically. It also describes the main ideas of the coverage models that have been solved by the engine to generate test vectors can verify all the corner cases in the hardware or software implementations of the decimal floating-point addition-subtraction operation.

The engine solved the coverage models one time and generated about 136000 test vectors in Decimal64, the test vectors have proved their efficiency by discovering bugs in Silminds design, most of the bugs appear from the cancellation model and the overflow model.

Chapter 3

Engine and Models of Decimal Multiplication Operation

The multiplication engine is a software tool, it generates multiplication test vectors to cover corner cases that verify the compliance of software or hardware implementations of the decimal floating-point multiplication operation with the IEEE standard (754-2008) for Floating Point Arithmetic.

The multiplication engine solved the coverage models one time and generated about 96000 test vectors in Decimal64, the test vectors have proved efficiency by discovering bugs in Silminds design[13].

The generated test vector is a decimal vector that has five sets. The first set is the operation type (multiplication), number of the precision (64 or 128), and the rounding mode. The second set is sign, significand, and exponent of the first input. The third set is sign, significand, and exponent of the second input. The fourth set is sign, significand, and exponent of the result. Finally the fifth set is one or two of four flags (invalid, inexact, underflow and overflow). The designer enters first three sets to his implementation and verifies the implementations output against last two sets.

The task given to the multiplication engine is the set of constraints on six elements: (1) the significand of the first input Sx, (2) the significand of the second input Sy, (3) the significand of the intermediate result Sz, (4) the exponent of the first input, (5) the intermediate result exponent which is the sum of the two inputs exponents, and (6) the rounding mode.

The constraint on Sx is a mask starting from the minimum number Nx to the maximum number Mx. The constraint on Sy is a mask starting from the minimum number Ny to the maximum number My. Each number in the previous two masks has p digits. Similarly, the mask on Sz consists of two numbers Nz and Mz, each number consists of $2p$ digits. The first input exponent, intermediate result exponent and the rounding direction are either given explicitly in the task or left to the engine to choose randomly.

An example to explain the format of the decimal multiplication task at $p=16$ is as follows:

$$64*T: \quad +1 \quad +9999999999999999 \quad -1 \quad -9999999999999999$$
$$-0\text{p}2000000000000000 \quad -9999999999999990\text{p}2999999999999999$$
$$R \quad R \quad 0$$

This multiplication task means that $Nx=+1$, $Mx=+9999999999999999$, $Ny=-1$, $My=-9999999999999999$, $Nz=-0\text{p}2000000000000000$ $Mz=-9999999999999990\text{p}2999999999999999$. Also, it means that the engine chooses randomly the exponent of the first input, the exponent of the intermediate result, and the rounding mode is(Round Ties to Even).

One of the solutions of this task is the test vector $d64*$ $=0$ $+377203339734945\text{E}41$ $-7473476140447729\text{E-}358$ -> $-2819020159606310\text{E-}302$ X.

The $d64$ means decimal64, the $*$ means multiplication operation, the following $=0$ means that the rounding mode is Round Ties to Even, the first input is $x=+377203339734945*10^{41}$, the second input is $y=-7473476140447729*10^{-358}$, the rounded result is $z=-2819020159606310*10^{-302}$, and the following X indicates that the inexact flag is high, because the exact result is $-2819020159606310.255808487189905*10^{-302}$.

3.1 The Multiplication Engine

The engine uses the Multiplication Algorithm to get, the first input significand Sx, the second input significand Sy, and the intermediate result significand Sz. Then, it gets the input exponents and the intermediate result exponent. The intermediate result exponent Ez either is chosen from the interval $[qmin, qmax]$, or is given explicitly. The first input exponent is chosen using

$max(qmin, Ez-qmax) \le E_x \le min(qmax, Ez-qmin)$, or is given explicitly. The second input exponent is calculated using $Ey=Ez-Ex$.

The engine shifts the intermediate result significand to right with a value $srz=max(0, p_z-p)$, and the intermediate result exponent Ez is replaced with $Ez+srz$.

In case of clamping, where $Ez>qmax \wedge Ez+p_z \le qmax+p$, the engine shifts to left Sz with a value that is equal to $Ez-qmax$, and replaces Ez with $qmax$.

At special case of under flow, where $Ez<qmin$ and $Ez+p_z \ge qmin$, it shifts to right Sz with a value that is equal to $qmin-Ez$, and replaces Ez with $qmin$.

The engine rounds the intermediate result according to the standard. The rounding process may generate a carry, which forces the engine to shift Sz one digit to right and increase Ez by one.

Finally, if Ez is larger than $qmax$, it is an overflow case. If Ez is smaller than $qmin$, it is an underflow case Sz. The cases result is according to the rounding mode.

3.1.1 The Multiplication Algorithm

The algorithm is based on solving the nonlinear equations that can be estimated from Figure 3, where each column represents one nonlinear equation. The figure shows the multiplication of two inputs significands at $p=8$, where Sz_i denotes the multiplication intermediate significand digit of weight 10^i, Sx_i denotes the first input digit of weight 10^i, and Sy_i denotes the second input digit of weight 10^i. The sum of digits in each column in addition to any carries from previous columns lead to one nonlinear equation.

The algorithm uses two methods to solve the non-linear equations, it chooses the proper method according to the constraints on the intermediate result. The first method is used, if the intermediate result constraints are on the least p

digits, the method solves the nonlinear equation from right to left as shown in Figure 3. The second method is used, if the intermediate result constraints are on the most p digits and some or all the least digits,the method solves the nonlinear equation from left to right as shown in Figure 3.

								$*\ Sx_7$	Sx_6	Sx_5	Sx_4	Sx_3	Sx_2	Sx_1	Sx_0
								Sy_7	Sy_6	Sy_5	Sy_4	Sy_3	Sy_2	Sy_1	Sy_0
								Sx_7Sy_0	Sx_6Sy_0	Sx_5Sy_0	Sx_4Sy_0	Sx_3Sy_0	Sx_2Sy_0	Sx_1Sy_0	Sx_0Sy_0
							Sx_7Sy_1	Sx_6Sy_1	Sx_5Sy_1	Sx_4Sy_1	Sx_3Sy_1	Sx_2Sy_1	Sx_1Sy_1	Sx_0Sy_1	
						Sx_7Sy_2	Sx_6Sy_2	Sx_5Sy_2	Sx_4Sy_2	Sx_3Sy_2	Sx_2Sy_2	Sx_1Sy_2	Sx_0Sy_2		
					Sx_7Sy_3	Sx_6Sy_3	Sx_5Sy_3	Sx_4Sy_3	Sx_3Sy_3	Sx_2Sy_3	Sx_1Sy_3	Sx_0Sy_3			
				Sx_7Sy_4	Sx_6Sy_4	Sx_5Sy_4	Sx_4Sy_4	Sx_3Sy_4	Sx_2Sy_4	Sx_1Sy_4	Sx_0Sy_4				
			Sx_7Sy_5	Sx_6Sy_5	Sx_5Sy_5	Sx_4Sy_5	Sx_3Sy_5	Sx_2Sy_5	Sx_1Sy_5	Sx_0Sy_5					
		Sx_7Sy_6	Sx_6Sy_6	Sx_5Sy_6	Sx_4Sy_6	Sx_3Sy_6	Sx_2Sy_6	Sx_1Sy_6	Sx_0Sy_6						
	Sx_7Sy_7	Sx_6Sy_7	Sx_5Sy_7	Sx_4Sy_7	Sx_3Sy_7	Sx_2Sy_7	Sx_1Sy_7	Sx_0Sy_7							
Sz_{15}	Sz_{14}	Sz_{13}	Sz_{12}	Sz_{11}	Sz_{10}	Sz_9	Sz_8	Sz_7	Sz_6	Sz_5	Sz_4	Sz_3	Sz_2	Sz_1	Sz_0

Figure 3. The Products of the Multiplication Operation assuming Precision 8.

In the two methods, the algorithm achieves the constraint of each digit Sx_i, Sy_i, or Sz_i, by choosing each digit from its interval $[Nx_i, Mx_i]$, $[Ny_i, My_i]$, and $[Nz_i, Mz_i]$.

A) The First Method

In the first method, as shown in Figure 3, the algorithm attempts to solve the first two nonlinear equations from right which are $cr_0 = Sx_0 Sy_0 - Sz_0$ and $cr_1 = Sx_0 Sy_1 + Sx_1 Sy_0 + cr_0/10 - Sz_1$. The algorithm chooses randomly the digit Sz_0 from its interval, therefore Sz_0 is known, then it searches randomly on the combination of the digits $Sx_0, Sy_0, Sx_1, Sy_1, Sz_1$ that achieves the two conditions $(cr_0) Mod_{10} = 0$ and $(cr_1) Mod_{10} = 0$. The steps taken so far constitute the first outer iteration that gets the final values of Sz_1, Sx_0, Sy_0, and estimates the values of Sx_1, Sy_1 that may be refined in the following iteration.

In the second iteration, the algorithm attempt to solve the second and the third nonlinear equations which are $cr_1 = Sx_0 Sy_1 + Sx_1 Sy_0 + cr_0/10 - Sz_1$, and $cr_2 = Sx_0 Sy_2 + Sx_2 Sy_0 + Sx_1 Sy_1 + cr_1/10 - Sz_2$. It searches randomly on the combination of the digits $Sx_1, Sy_1, Sx_2, Sy_2, Sz_2$ that achieves the two conditions $(cr_1) Mod_{10} = 0$ and $(cr_2) Mod_{10} = 0$, where the digits $cr_{0,} Sx_0, Sy_0, Sz_0, Sz_1$, are known from the previous iteration. The algorithm does this procedure in the next iterations, until it find all digits of Sx and Sy. then, it multiply Sx with Sy to get the all digits of Sz.

In general, the algorithm determines the maximum number of digits of the first input significand $min(p_z - no\ of\ digits\ of\ My, p) \leq p_x \leq no\ of\ digits\ of\ Mx$, and the maximum number of digits of the second input significand $p_y = p_z - p_x$, where p_z is number of digits of the intermediate result, which solve the problem of the leading zero digits in the intermediate result significand.

It chooses randomly the digit Sz_0 from its interval, and does outer iterations of index i, where $0 \leq i \leq p-1$. In each iteration, it gets the digits Sz_{i+1}, Sx_i, Sy_i, and estimates the digits Sx_{i+1}, Sy_{i+1}, such that the combination of the previous digits achieves the conditions $(cr_i) mod_{10} = 0$ and $(cr_{i+1}) mod_{10} = 0$.

The general form of the two nonlinear equations that each iteration attempt to solve are:

$$cr_i = \sum_{j=0}^{j=i} Sx_{i-j} Sy_j - Sz_i \tag{3.1}$$

$$cr_{i+1} = \sum_{j=0}^{j=i+1} Sx_{i-j+1} Sy_j + cr_i/10 - Sz_{i+1} \tag{3.2}$$

In the last of each outer iteration, Sz_{i+1} is replaced by $Sz_{i+1} - cr_i/10$, such that the nonlinear equations are in the previous general form.

Finally, after getting all digits of Sx and Sy, it calculates the intermediate result significand $Sz = Sx * Sy$, to get all digits of Sz. The engine chooses different p_x and p_y and repeats all the iterations, if one of the conditions in any iteration is not achieved.

B) The Second method

In the second method, the algorithm iterates to solve the nonlinear equations from left to right. As shown in Figure 3, for $p=8$, the first nonlinear equation from left is $Sz_{14} - Sx_7 Sy_7 = br_{14}$ where br_{14} is the value of carries that transfer from previous weights to the weight of 10^{14}, or the borrow generated from this weight to lower weights. The second and the third non linear equations are $Sz_{13} + 10 * br_{14} - Sx_7 Sy_6 - Sx_6 Sy_7 = br_{13}$, and $Sz_{12} + 10 * br_{13} - Sx_7 Sy_5 - Sx_6 Sy_6 - Sx_6 Sy_7 = br_{12}$.

In general the nonlinear equation for the column of index n, where $n \leq p-1$, is :

$$br_n = Sz_n + 10*br_{n-1} - \sum_{j=n-p+1}^{j=p-1} Sy_j Sx_{n-j}, \tag{3.3}$$

To start the solution, the algorithm attempts to solve the first three nonlinear equations (representing columns 7 to 5) together based on the range of carries that may transfer from the next lower significant columns. The algorithm chooses randomly the digits $Sz_{15}, Sz_{14},$ and $Sz_{13},$ from their intervals, and replaces the digit Sz_{14} with the value $Sz_{14} + 10*Sz_{15}.$ Then since the ranges of borrow digit $br_{12},$ the digit $Sx_5,$ and the digit Sy_5 are known as $Ncr_{13} \le br_{13} \le Mcr_{13},$ $Nx_5 \le Sx_5 \le Mx_5,$ and $Ny_5 \le Sy_5 \le My_5,$ where Ncr_{12} and Mcr_{12} are equal to

$$Ncr_{12} = \frac{\sum_{j=6}^{j=7} Sy_j Nx_{11-j} + \sum_{j=4}^{j=5} Ny_j Sx_{11-j}}{10} + \frac{\sum_{j=6}^{j=7} Sy_j Nx_{10-j} + \sum_{j=3}^{j=4} Ny_j Sx_{10-j} + Ny_5 Nx_5}{100}$$

$$Mcr_{12} = \frac{\sum_{j=6}^{j=7} Sy_j Mx_{11-j} + \sum_{j=4}^{j=5} My_j Sx_{11-j}}{10} + \frac{\sum_{j=6}^{j=7} Sy_j Mx_{10-j} + \sum_{j=3}^{j=4} My_j Sx_{10-j} + My_5 Mx_5}{100}$$

The algorithm transforms the third nonlinear equation to the inequality condition:

$$Ncr_{12} + Nx_5 Sy_7 + Sx_7 Ny_5 \le Sz_{12} + 10*br_{13} - Sx_6 Sy_6 \le Mcr_{12} + Mx_5 Sy_7 + Sx_7 My_5. \tag{3.4}$$

Finally, it searches randomly on the combination of the values of $Sx_7, Sy_7, Sx_6, Sy_6, Sz_{13}$ that satisfy the first nonlinear equation, the second nonlinear equation and the Inequality 3.4. The steps taken so far constitute the first outer iteration that gets the final values of $Sx_7,$ $Sy_7,$ $Sz_{13},$ and estimates the values of $Sx_6,$ Sy_6 which may be refined in the next iteration.

In the second iteration, the algorithm transforms the fourth nonlinear equation $Sz_{11} + 10*br_{12} - Sx_7*y_4 - Sx_6 Sy_5 - Sx_5 Sy_6 - Sx_4 Sy_7 = br_{11}$ to the inequality condition:

$$Ncr_{11} + Nx_4*Sy_7 + Sx_7*Ny_4 \le Sz_{11} + 10*br_{12} - Sx_6 Sy_5 - Sx_5 Sy_6 \le Mcr_{11} + Mx_4 Sy_7 + Sx_7 My_4.$$

It searches randomly on the combination of values of $Sx_6, Sy_6, Sx_5, Sy_5, Sz_{12}$ that achieves the second nonlinear equation, the third nonlinear equation and the inequality condition, where the digits $Sx_7,$ $Sy_7,$ $br_{14}, Sz_{14}, Sz_{13}$ are known from the previous iteration. The algorithm does this procedure in all the iterations and gets all digits of Sx and Sy.

In general, the algorithm gets the digits Sz_{2p-1}, Sz_{2p-2}, and Sz_{2p-3} from their intervals, and replaces Sz_{2p-2} with $Sz_{2p-2}+10*Sz_{2p-1}$. It does number of iterations of index i, from $i=p-1$ to $i=0$. It gets in each iteration the digits Sx_i, Sy_i, Sz_{i+p-3}, and estimates the digits Sx_{i-1}, Sy_{i-1}, such that this combination of digits achieves two nonlinear equations and the inequality condition. The general form of the two nonlinear equations and the inequality condition are:

$$br_{i+p-1}=Sz_{i+p-1}-\sum_{j=i}^{j=p-1} Sx_j Sy_{i-j+p-1} \tag{3.5}$$

$$br_{i+p-2}=Sz_{i+p-2}+10*br_{i+p-1}-\sum_{j=i-1}^{j=p-1} Sx_j Sy_{i-j+p-2} \tag{3.6}$$

$$\begin{aligned} Ncr_{i+p-3}+Sx_{p-1}Ny_{i-2}+Nx_{i-2}Sy_{p-1}\leq Sz_{i+p-3}+10*br_{i+p-2}-\sum_{j=i-1}^{j=p-2} Sx_j*Sy_{i-j+p-3}\leq \\ Sx_{p-1}My_{i-2}+Mx_{i-2}Sy_{p-1}+Mcr_{i+p-3}. \end{aligned} \tag{3.7}$$

Note that, Ncr_{i+p-3} and Mcr_{i+p-3} are the minimum and the maximum carries that generated from the columns that follow the column of index $i+p-3$. Since the column that has the maximum product sum, is the column of index $p-1$, where the maximum product sum at $p=34$ is equal to $33*9*9=2673$. This number means that a carry from any column, at $p\leq 34$, may affect the previous three columns directly by a value more than one and affects the higher columns indirectly by a value less than or equal to one. Based on that, the algorithm determines the range of carries that transfer to the column $i+p-3$ from the next three columns $i+p-4$, $i+p-5$, $i+p-6$. The general form of the carries equations are:

$$\begin{aligned} Ncr_{i+p-3}=&(\sum_{j=p-2}^{j=p-1} Sy_j Nx_{i+p-4-j}+\sum_{j=i-3}^{j=i-2} Ny_j Sx_{i+p-4-j}+\sum_{j=i-1}^{j=p-3} Sy_j Sx_{i+p-4-j})/10+ \\ &(\sum_{j=p-3}^{j=p-1} Sy_j Nx_{i+p-5-j}+\sum_{j=i-4}^{j=i-2} Ny_j Sx_{i+p-5-j}+\sum_{j=i-1}^{j=p-4} Sy_j Sx_{i+p-5-j})/100+ \\ &(\sum_{j=p-4}^{j=p-1} Sy_j Nx_{i+p-6-j}+\sum_{j=i-5}^{j=i-2} Ny_j Sx_{i+p-6-j}+\sum_{j=i-1}^{j=p-5} Sy_j Sx_{i+p-6-j})/1000 \end{aligned} \tag{3.6}$$

$$
\begin{aligned}
Mcr_{i+p-3} = & \left(\sum_{j=p-2}^{j=p-1} Sy_j Mx_{i+p-4-j} + \sum_{j=i-3}^{j=i-2} My_j Sx_{i+p-4-j} + \sum_{j=i-1}^{j=p-3} Sy_j Sx_{i+p-4-j}\right)/10 + \\
& \left(\sum_{j=p-3}^{j=p-1} Sy_j Mx_{i+p-5-j} + \sum_{j=i-4}^{j=i-2} My_j Sx_{i+p-5-j} + \sum_{j=i-1}^{j=p-4} Sy_j Sx_{i+p-5-j}\right)/100 + \\
& \left(\sum_{j=p-4}^{j=p-1} Sy_j Mx_{i+p-6-j} + \sum_{j=i-5}^{j=i-2} My_j Sx_{i+p-6-j} + \sum_{j=i-1}^{j=p-5} Sy_j Sx_{i+p-6-j}\right)/1000
\end{aligned} \tag{3.7}
$$

The values of Sx and Sy calculated so far achieve only the most significand digits of Sz. The algorithm must alter correlates the values of Sx and Sy, such that they achieve all the constraints on the digits of Sz.

The algorithm calculates the intermediate result using $\bar{Sz} = Sx * Sy$, and gets Sz by assign to it $\bar{Sz}$ with replacing the digits that do not achieve the constraints with one that achieve. It checks that either condition 1 $(|Sz - \bar{Sz}|) \bmod x \leq maxerror$ is achieved, or condition 2 $(|Sz - \bar{Sz}|) \bmod y \leq maxerror$ is achieved. If condition 1 is achieved, it replaces Sx with $Sx + \frac{(Sz - \bar{Sz}) - (Sz - \bar{Sz}) \bmod Sy}{Sy}$.

Otherwise, if condition 2 is achieved, it replaces Sy with $Sy + \frac{(Sz - \bar{Sz}) - (Sz - \bar{Sz}) \bmod Sx}{Sx}$. If the two conditions are not achieved the algorithm repeats all the iterations to get new values of Sx and Sy, until one of the conditions is achieved. The algorithm does not guarantee that the conditions is achieved. In this case, it refines the constraints which leads to refine the maximum error, which the case that the engine may not solve.

Finally the algorithm gets the final value of the intermediate result using $Sz = Sx * Sy$.

3.2 The Main Ideas of the Multiplication Models

The models are defined using a Cartesian product between two or more lists of constraints with ignoring the impossible combinations, and allowing the other constraints to be chosen randomly.

All the proposal ideas of the models are in [22], however we describe the ideas in the form of the engine constraints.

A) Inputs Types Model

The model verifies the possible combinations of input types, we separate the model into four sub-models as follows:

1. It verifies the design when one of the inputs is Zero using, (1) the significand of one of the inputs is equal to zero, (2) a list of zero significand input that consists of the exponent interval $[qmin, qmax]$, (3) a list from all input types of the other input.

2. It verifies the design when one of the inputs is Infinity, sNaN, or qNaN using, (1) a list of one of the inputs from the Infinities, sNaN, and qNaN, (2) all input types list of the other input.

3. It verifies the design in solving other types of input using, (1) a list of one of the inputs from the minimum Subnormal, the maximum Subnormal, the minimum Normal, and the maximum Normal, (2) a list of the other input from the exponent interval $[qmin, qmax]$.

4. It verifies the design when one of the inputs is equal to One using, (1) one of the inputs is equal to One, (2) a list of the other input from the exponent interval $[qmin, qmax]$.

B) Result Types Model

The model verifies the generation of different types of the final result. We separate the model into four sub-models as follows:

1.It verifies all the result exponents using, (1) a list of the intermediate result exponent from the interval $[qmin, qmax]$.

2. It verifies the generation of the first hundred Subnormal numbers, the last hundred Subnormal numbers, and the first hundred Normal numbers using, (1) the intermediate result exponent is equal to $qmin$, (2) a list of the intermediate result significand from the intervals $\{[2,100],[10^{p-1}-100,10^{p-1}+100]\}$.

3. It verifies the generation of numbers from one to 100 using, (1) the intermediate result exponent is equal zero, (2) a list of the intermediate result significand from the interval $[1,100]$.

4. It verifies the last hundred Normal numbers using, (1) the exponent intermediate result is equal to $qmax$, (2) a list of the intermediate result significand from the interval $[10^P-1, 10^P-100]$.

C)Rounding model

The model verifies the rounding process. We separate the model into four sub-models as follows:

1. It verifies the rounding process using, (1) a list from the five rounding modes, (2) a list of the intermediate result significand from the cross product of the guard digit interval $[0,9]$, the least significand digit interval $[0,9]$, the sticky bit interval $[0,1]$, and number of digits of the intermediate result interval $[1,2p]$.

2. It verifies all possible carry propagations in the intermediate result significand due to the rounding process using, (1) a list from the five rounding modes, (3) a list of the intermediate result exponent that consists of $\{qmax, emin, zero, random\,number\}$, (2) a list of intermediate result significand from the cross product of the guard digit interval $[0,9]$, the sticky bit interval $[0,1]$, number of digits of the intermediate result interval $[p,2p]$, and the patterns $\{\overbrace{99\cdots9}^{p}X\cdots X, \overbrace{\{0-8\}9\cdots9}^{p}X\cdots X, \overbrace{X\{0-8\}9\cdots9}^{p}X\cdots X, \cdots, \overbrace{XX\cdots X\{0-8\}}^{p}X\cdots X\}$.

3. It verifies the sticky bit calculations using, (1) a list of the intermediate result significand from the cross product of number of digits interval $[p,2p]$, and the patterns $\{\overbrace{XX\cdots X}^{p+1}0\{1-9\}XX\cdots X, \overbrace{XX\cdots X}^{p+1}00\{1-9\}XX\cdots X, \cdots, \overbrace{XX\cdots X}^{p+1}00\cdots0\{1-9\}\}$.

D)Trailing and Leading Zeros Model

The model verifies the trailing and leading zeros in the input significands and the intermediate result significand. We separate the model into two sub -models as follows:

1.It verifies the patterns of zeros in the input significands using, (1) a list of the first input significand, (2) a list of the second input same like previous list, the list consists of

$$\{\overbrace{\{1-9\}00\cdots00}^{P},\overbrace{0\{1-9\}00\cdots00}^{P},\cdots,\overbrace{00\cdots0\{1-9\}}^{P}\}$$
$$\{\overbrace{\{1-9\}\{1-9\}0\cdots00}^{P},\overbrace{0\{1-9\}\{1-9\}0\cdots00}^{P},\cdots,\overbrace{00\cdots0\{1-9\}\{1-9\}}^{P}\}$$
$$\{\overbrace{\{1-9\}X\{1-9\}0\cdots00}^{P},\overbrace{0\{1-9\}X\{1-9\}0\cdots00}^{P},\cdots,\overbrace{00\cdots0\{1-9\}X\{1-9\}}^{P}\}$$
$$\vdots$$
$$\{\overbrace{\{1-9\}XX\cdots X\{1-9\}}^{P}\}$$

2. It verifies the trailing and leading zeros in the intermediate result significand using, (1) a list of the intermediate result sigificand from the patterns

$$\{\overbrace{\{1-9\}00\cdots00}^{P+1},\overbrace{0\{1-9\}00\cdots00}^{P+1},\cdots,\overbrace{00\cdots0\{1-9\}}^{P+1}\}$$
$$\{\overbrace{\{1-9\}\{1-9\}0\cdots00}^{P+1},\overbrace{0\{1-9\}\{1-9\}0\cdots00}^{P+1},\cdots,\overbrace{00\cdots0\{1-9\}\{1-9\}}^{P+1}\}$$
$$\{\overbrace{\{1-9\}X\{1-9\}0\cdots00}^{P+1},\overbrace{0\{1-9\}X\{1-9\}0\cdots00}^{P+1},\cdots,\overbrace{00\cdots0\{1-9\}X\{1-9\}}^{P+1}\}$$
$$\vdots$$
$$\{\overbrace{\{1-9\}XX\cdots X\{1-9\}}^{P+1}\}$$

E) Overflow Model

The model verifies the overflow cases. We separate the model into five sub-models as follows:

1. It verifies the overflow cases when the result exponent is larger than $qmax$, using, a list of the intermediate result exponent from the interval $[qmax-p-1,2qmax]$.

2. It verifies the overflow and the near-overflow cases due to the rounding process using, (1) the intermediate result significand is equal to $10^{16}-1$, and has the guard digit interval $[5,9]$, (2) the two rounding modes Round ties to even and Round ties to away, (3) a list of the intermediate result exponent from the interval $[qmax-p-1,qmax-1]$.

3. The intermediate result significand is equal to $10^{16}-1$, with a list of guard digit in the interval $[1,9]$ at the two rounding modes, Round to positive and Round to negative, with a list of the intermediate result exponent in the interval $[qmax-p-1,qmax-1]$, to verify the overflow and the near-overflow cases due to the rounding process.

4. It verify the overflow cases due to number of digits of the intermediate result significand using, (1) a list of the intermediate result exponent from the interval

$[qmax-p-1, qmax]$, (2) a list of number of digits of the intermediate result significand from the interval $[p, 2p]$.

5. It verifies the near-overflow cases which need to shift the intermediate result significand to left using, (1) a list of the intermediate result exponent from the interval $[qmax, qmax+p-1]$, (2) a list of number of digits of the intermediate result significand from the interval $[1, p]$.

F)Underflow Model

The model verifies the underflow cases. We separate the model into four sub-models as follows:

1. It verifies the underflow cases when the intermediate result exponent is less than $qmin$ using, (1) a list of the intermediate result exponent from the interval $[2qmin, qmin]$.

2. It verifies the underflow and the near-underflow cases when the intermediate result significand is shifted to right and the result is inexact using, (1) a list of the intermediate result exponent from the interval $[qmin-2p, qmin]$, (2) a list of number of digits of the intermediate result from the interval $[1,2p]$.

3. It verifies the underflow and the near-underflow cases when the intermediate result significand is shifted to right and the result is exact using, (1) a list of the intermediate result exponent from the interval $[qmin-2p, qmin]$, (2) a list of the intermediate result significand that consists of the patterns $\{\{1-9\}00\cdots0, X\{1-9\}00\cdots0, \cdots, XX\cdots X\{1-9\}\}$,

4. It verifies the near-underflow cases and the subnormals numbers using, (1) a list of the intermediate result exponent from the interval $[qmin, qmin+p-1]$, (2) a list of number of digits of the intermediate result from the interval $[1,2p]$.

3.3 Previous work

The Fpgen multiplication algorithm by IBM [1] is given the constraints on the intermediate result $\bar{S_z}$ which has up to $2p$ digits, and on the difference $0 \le d \le p$ between the actual and the preferred exponents.

The algorithm represents the problem into two cases:

Case 1: The sticky bit is zero and $d-1$ trailing zeros after the guard digit exist, the algorithm factorizes $\bar{S}_z = S_z.10^{d-1}$ to its prime factors, then selects random factors for S_x and S_y such that the value of each is smaller than 10^p, then selects random exponent e_x, e_y such that $e_x + e_y = e_z - d.$

Case 2: The sticky bit is one and $d > 2$, the algorithm uses the following procedure: (1) it computes the range of possible values of $\bar{S}_z$ using $S_z.10^{d-1} \le \bar{S}_z \le (S_z+1).10^{d-1}$, (2) it selects randomly the number of digits $|S_y| \le p$ and the value of S_y using $S_y \le (\frac{\bar{S}_z}{10^p - 1})$, (3) it chooses S_x using $(\frac{S_z.10^{d-1}}{S_y} \le S_x \le \frac{(S_z+1).10^{d-1}}{S_y})$, if a decimal value is founded in that range, this mean that the solution exists, otherwise the algorithm returns to step two. On the average the algorithm can find a solution for S_x within 10 trials.

3.4 Comparison

The Fpgen multiplication algorithm cannot solve simultaneous constraints on the inputs significand and the intermediate result significand, and cannot solve all the constraints on the digits that follow the guard digit of the intermediate result significand, while our engine solves these constraints numerically. Both of them cannot find the solution from the first trail, but they find the solution in practical time.

3.5 Summary

This chapter represents the main steps that the multiplication engine uses to solve all the constraints numerically. It also describes the main ideas of the coverage models that have been solved by the engine to generate test vectors can verify all the corner cases in the hardware or software implementations of the decimal floating-point multiplication operation.

The engine solved the coverage models one time and generated about 96000 test vectors in Decimal64, the test vectors have proved efficiency by discovering bugs in Silminds design. The bugs are appeared in the input types model.

Chapter 4

Engine and Models of Decimal Fused Multiply Add (FMA) Operation

The fused-multiply-add(FMA) engine generates FMA test vectors to cover all corner cases, to verify a tested implementation of decimal fused-multiply-add (FMA) operation to achieve the compliance with the IEEE standard (754-2008) for Floating Point Arithmetic.

The FMA engine solved the coverage models one time and generated about 425000 test vectors in Decimal64, the test vectors have proved their efficiency by discovering bugs in Silminds design[15] and FMA DecNumber implementation [23].

The generated test vector is a decimal vector that has six sets. The first set is the operation type (FMA), number of the precision (64 or 128), and the rounding mode. The second set is sign, significand, and exponent of the first input. The third set is sign, significand, and exponent of the second input. The fourth set is sign, significand, and exponent of the second input. The fifth set is sign, significand, and exponent of the result. Finally the sixth set is one or two of four flags(invalid, inexact, underflow and overflow). The designer enters the input sets to his implementation and verifies the implementation output against last two sets.

The FMA operation $x*y\pm b=c$ multiplies the first two inputs, and adds without rounding the result of the multiplication operation to the third input.

The task given to the fused-multiply-add(FMA) engine is the set of constraints on eleven elements, (1) the significand of the first input Sx, (2) the significand of the second input Sy, (3) the significand of the third input Sb, (4) the multiplication intermediate result Sz, (5) the addition intermediate result Sc, (6) the exponent of the first input, (7) the multiplication intermediate result exponent which is the sum of the first two inputs exponents, (8) identifier number sid to determine the smaller exponent input of the addition operation (i.e the exponent of third input or the exponent of the multiplication intermediate result), such that the engine determines which significand will be shifted to right, (9) right shift value of the smaller exponent addition input. (10) the addition intermediate result exponent at which the addition_subtraction operation occurs, (11) the rounding mode.

The constraint on Sx is a mask starting from the minimum number Nx to the maximum number Mx. The constraint on Sy is a mask starting from the minimum number Ny to the maximum number My. The constraint on Sb is a mask starting from the minimum number Nb to the maximum number Mb. Each number in the previous masks has p digits. Similarly, the mask on Sz consists of two numbers Nz and Mz, each number has $2p$ digits, Also, the mask on Sc consists of two numbers Nc and Mc, each number has $2p+1$ digits, $p+1$ digits before the fractional point and p digits after it. The first input exponent, the multiplication intermediate result exponent, the addition intermediate result exponent and the rounding direction are either given explicitly in the task or left to the engine to choose randomly.

The ability of the engine to choose randomly within the range of the mask or to choose the input exponents and the rounding direction empowers the engine to generate test vectors discovering more bugs.

An example to explain the format of the decimal FMA task at $p=16$ is as follows:

$$\begin{gathered}64{*}{+}T: \quad -1 \quad -9999999999999999 \quad -1 \quad -9999999999999999 \quad -100000 \quad -999999 \\ +1000000000000000 \quad +99999999999999999999999999999999 \\ -9999999999999999.9000000000000000 \quad -9999999999999999.9999999999999999 \\ R \quad R \quad 0 \quad 6 \quad R \quad R\end{gathered}$$

This task means that $Nx=-1$, $Mx=-9999999999999999$, $Ny=-1$, $My=-9999999999999999$, $Nb=-100000$, $Mb=-999999$, $Nz=+1000000000000000$, $Mz=+99999999999999999999999999999999$, $Nc=-9999999999999999.9000000000000000$, $Mc=-9999999999999999.9999999999999999$.

Also, it means that the engine chooses randomly the exponent of the first input, the exponent of the multiplication intermediate result, the exponent of the addition intermediate result, and the rounding mode. The engine determines from the task that the third input is the smaller addition exponent, and the significand of the smaller addition exponent (the third input exponent) will be shifted to right six digits.

One of the solutions of this task is the test vector d64*+ =0 $-9046436700100791E-59$ $-11054076131311E127$ $-81183E76$ -> $+9999999999999999E81$ X.

The $d64$ means decimal64, the *+ means FMA operation(i.e multiplication operation followed by addition operation), the following =0 means that the rounding mode is Round ties to even, the first input is $x=-9046436700100791*10^{-59}$, the second input is $y=-11054076131311*10^{127}$, the third input is $b=-81183*10^{76}$, the rounded result is $c=+9999999999999999*10^{81}$, and the following X indicates that the inexact flag is high, because the exact result is $9999999999999999.2782750967001*10^{81}$.

4.1 The FMA engine

The engine determines the number of digits of the multiplication intermediate result p_z from the interval $[no\, of\, digits\, of\, Nz, no\, of\, digits\, of\, Mz]$, as $0\le p_z\le 2p$, and number of digits of the third input p_b from the interval $[no\, of\, digits\, of\, Nb, no\, of\, digits\, of\, Mb]$. The engine shifts to right both Nz and Mz with the value srm_z, to be in the format of maximum p digits before the fractional point.

According to the value of sid, the engine determines the smaller exponent input of the addition operation. Therefore, the engine chooses between two procedures, (1) procedure 1, the multiplication intermediate result exponent is the smaller exponent input of the addition operation, therefore the multiplication intermediate result significand is shifted to right and the third input significand is shifted to left, (2) procedure 2, the third input exponent is the smaller exponent input of the addition operation, therefore the third input significand is shifted to right and the multiplication intermediate result significand is shifted to left.

In procedure 1, the engine chooses randomly the right-shift value sr_z, either from the intervals $[1,p]$ or $[p+1, qmax-2*qmin]$. If sr_z is equal to zero, it will choose randomly the left-shift value sl_b, from the interval $[0, p-p_b]$. Otherwise, if sr_z is larger than zero, sl_b is equal to $p-p_b$. The engine shifts to left both Nb and Mb with the value of sl_b, and shifts to right both Nz and Mz with the value of sr_z. Then, the engine uses the Addition Algorithm (in 2.1.1) to get the third input significand Sb, the multiplication intermediate result significand Sz, and the addition intermediate result of Sc. After that, the engine shifts to left the significand Sz with the value of sr_z+srm_z, and factorizes Sz to the two inputs significands Sx and Sy using the Multiplication Algorithm (in 3.1.1).

The engine recalculates the new value of Sc by replacing Sc with $Sz+Sb$, as the Multiplication Algorithm changes some digits in Sz. It shifts to right the third input significand Sb, with the value of sl_b, and calculates the input exponents that achieve the values of sl_b and sr_z.

To calculate the exponents, the engine chooses the addition intermediate result exponent Ec from the interval $[max(sr_z+srm_z+2*qmin, qmin), qmax-sl_b]$, then it calculates the exponent of the multiplication intermediate result $Ez=Ec-sr_z-srm_z$, and the third input exponent $Eb=Ec+sl_b$. It chooses the first input exponent Ex using $max(qmin, Ez-qmax) \le E_x \le min(qmax, Ez-qmin)$, or Ex is given explicitly, and it calculates the second input exponent using $Ey=Ez-Ex$.

However, if Ez is given explicitly to the engine, the engine gets the first input exponent Ex using $max(qmin, Ez-qmax) \le E_x \le min(qmax, Ez-qmin)$, or Ex is given explicitly, and it gets the second input exponent using $Ey=Ez-Ex$. The exponent of the addition intermediate result Ec is equal to $Ez+sr_z+srm_z$, and the third input exponent Eb is equal to $Ec+sl_b$.

In procedure 2, the engine chooses randomly the right-shift value sr_b, either from the intervals $[1,2p]$, $[p+1, qmax-qmin]$, or $[qmax+1-qmin, 2*qmax-qmin]$. If sr_b is equal to zero, it will choose randomly the left-shift value sl_z, from the interval $[0, p-p_z]$. Otherwise, if sr_b is larger than zero, sl_z is equal to $p-p_z$. The engine shifts to left both Nz and Mz with the value of sl_z, and shifts to right both Nb and Mb with the value of sr_b. Then, the engine uses the Addition Algorithm (in 2.1.1) to get the third input significand Sb, the multiplication intermediate result significand Sz, and the addition intermediate result of Sc. After that, the engine shifts to right Sz with the value srm_z and shifts to left Sz with value sl_z. It factorizes Sz to the two inputs significands Sx and Sy using the Multiplication Algorithm(in 3.1.1).

The engine recalculates the new value of Sc by replacing Sc with $Sz+Sb$, as the Multiplication Algorithm changes some digits in Sz. It shifts to left the third input significand Sb, with the value of sr_b, and calculates the input exponents that achieve the values sr_b and sl_z.

The engine chooses the addition intermediate result Ec from the interval $[qmin+sr_b, qmax]$, it calculates the multiplication intermediate result exponent using $Ez=Ec+sl_z-srm_z$, and the third input exponent using $Eb=Ec-sr_b$. The engine gets the first input exponent Ex, either from the interval $[max(qmin, Ez-qmax), min(qmax, Ez-qmin)]$, or Ex is given explicitly, and it calculates the second input exponents using $Ey=Ez-Ex$.

However, if Ez is given to the engine, the engine gets the first input exponent Ex, either from the interval $[max(qmin, Ez-qmax), min(qmax, Ez-qmin)]$, or Ex is given explicitly, and it gets the second input exponent using $Ey=Ez-Ex$. The exponent of the addition intermediate result Ez is equal to $Ez-sl_z+srm_z$, and the third input exponent Eb is equal to $Ec-sr_b$.

The addition intermediate result may have cancellation digits, in that case the engine shifts Sc to left and decreases Ec with a value $scn=min(Ec-Ez,\ p-no\,of\,digits\,before\,point)$.

The addition intermediate result may have a carry digit, in that case the engine shifts Sc one digit to the right and increases Ec by one.

At clamping, where $Ec > qmax \wedge Ec + p_c \leq qmax + p$, the engine shifts to left Sc with the value $Ec - qmax$ and replaces Ec with $qmax$.

At special case of underflow, where $Ec < qmin$ and $Ec + p_c \geq qmin$, it shifts to right Sc with the value $qmin - Ec$ and replaces Ec with $qmin$.

The engine rounds the addition intermediate result according to the standard. The rounding process may generate a carry to force the engine to shift Sc one digit to right and increase Ec by one.

Finally, if Ec is larger than $qmax$, it is an overflow case, and if Ec is smaller than $qmin$, it is an underflow case. The result of these cases are according to the rounding mode.

4.2 The Main Ideas of the FMA Models

The models are defined using a Cartesian product between two or more lists of constraints while ignoring the impossible combinations and allowing the other constraints to be chosen randomly.

Some of the model proposal ideas are also in [22]. We write down during the explanation of these ideas that they are in [22]. However we describe these ideas in the form of our engine constraints. The other ideas are new ideas to verify new corner cases in the different FMA implementations. In total we present 42 sub-models of which 23 sub-model ideas are in [22] and 19 sub-model ideas are new.

A) Inputs Types Model

The model aims to verify the ability of the design to solve all possible combinations of the input types. The proposal ideas of the model are in [22]. We separate the model into three sub-models as follows:

1. It verifies the handling of Normal and Subnormal types of the first two inputs, using the following lists of constraints, (1) a first input list consists of

the minimum Subnormal, the maximum Subnormal, and the maximum Normal, (2) a second input exponent list consists of all the exponent values in the interval $[qmin, qmax]$.

2. It verifies the remaining of Normal and Subnormal types of the third input, using the following lists of constraints, (1) a third input list consists of the minimum Subnormal, the maximum Subnormal, and the maximum Normal, (2) a list of the multiplication intermediate result exponent consists of the exponent values in the interval $[2*qmin, 2*qmax]$.

3. It verifies the input types Zero, Infinities, sNaN, or qNaN; using the four combinations of lists in Table 1.

TABLE 1.COMBINATIONS OF INPUTS TYPES LISTS

Id	**The Contents of The lists**		
	First Input	***Second Input***	***Third input***
1	Zero with all possible exponents	All input types list	All input types list
2	All input types list	All input types list	Zero with all possible exponents
3	Infinities, sNaN, and qNaN	All input types list	All input types list
4	All input types list	All input types list	Infinities, sNaN, and qNaN

B) Result Types Model

The model aims to verify the ability of the design to generate all the result types that has not been generated in the previous model. The proposal ideas of the model are also in [22]. We separate the model into four sub-models as follows:

1. It verifies all the result exponents using, (1) a list of the addition intermediate result exponents consists of the interval $[qmin, qmax]$.

2. It verifies the generation of the first hundred Subnormal numbers, the last hundred Subnormal numbers, the first hundred Normal numbers, and numbers

from One to 100 using, (1) a list of the addition intermediate result significand consists of the intervals $\{[1,100],[10^{P-1}-100,10^{P-1}+100]\}$, (2) a list of the addition intermediate result exponent consists of zero and $qmin$.

3. It verifies the last hundred Normal numbers using, (1) a list of the addition intermediate result significand consists of the interval $[10^{P}-1,10^{P}-100]$, (2) the addition intermediate result exponent is equal to $qmax$.

C) Rounding Model

The model aims to verify the rounding process in the design. Some of the proposal ideas of the model are in [22], while the other ideas are new. We separate the model into eight sub-models as follows:

1. It verifies the rounding process at all combinations of the guard digit, the least significand digit, and the sticky bit using, (1) a list from the five rounding modes, (2) a list of the addition intermediate significand consists of, the guard digit interval $[0,9]$, the least digit interval $[0,9]$, and the sticky bit interval $[0,1]$, (3) a list from the two values of sid that determines the smaller exponent input of the addition operation. The proposal idea of this sub-model is also in [22].

2. It verifies the possible carry propagation due to rounding process using, (1) a list from the five rounding modes, (2)a list from two values of sid, (3) a list of the addition intermediate result significand consists of, the guard digit interval $[0,9]$, the sticky bit interval $[0,1]$, and the patterns $\{99\cdots 99,\{0-8\}99\cdots 99,X\{0-8\}9\cdots 99,\cdots,XX\cdots X\{0-8\}\}$. The proposal idea of this sub-model is in [22].

3. It verifies the sticky bit calculations using, (1) a list of right shift to the third input consists of the interval $[2,qmax-qmin]$, (2) sid indicates that the third input exponent is the smaller exponent input of the addition operation, (3) the number of digits of the third input significand is equal to one. The proposal idea of this sub-model is in [22].

4.It verifies the sticky bit calculations using, (1) a list of right shifts to the multiplication intermediate result from the interval $[2, qmax-2*qmin]$, (2) sid indicates that the multiplication intermediate result exponent is the smaller exponent input of the addition operation, (3) the number of digits of the multiplication intermediate result significand is equal to one. The proposal idea of this sub-model is in [22].

5. It verifies the rounding process when the right shift is less than p using; (1) a list from the five rounding modes, (2) a list of number of digits of the third input significand consists of the interval $[1, p]$, (3) sid indicates that the third input exponent is the smaller exponent input of the addition operation, (4) a list of the right shift consists of the interval $[1, p]$.

6. It verifies the rounding process when the right shift is less than p using, (1) a list from the five rounding modes, (2) a list of number of digits of the multiplication intermediate result significand consists of the interval $[1,2p]$, (3) sid indicates that the multiplication intermediate result exponent is the smaller exponent input of the addition operation, (4) a list of the right shift consists of the interval $[1, p]$.

7. It verifies the sticky bit when the right shift value is less than p using, (1) the right shift value is less than p, (2) sid indicates that the multiplication intermediate result exponent is the smaller exponent input of the addition operation, (3) a list of the multiplication intermediate result significand consists of the pattern

$$
\begin{gathered}
\overbrace{\{1-9\}00\cdots0}^{p+1}\overbrace{\{1-9\}X\cdots X}^{p-1}, \overbrace{X\{1-9\}00\cdots0}^{p+1}\overbrace{\{1-9\}X\cdots X}^{p-1}, \cdots, \overbrace{X\cdots X\{1-9\}}^{p+1}\overbrace{\{1-9\}X\cdots X}^{p-1}, \\
\overbrace{\{1-9\}00\cdots0}^{p+1}\overbrace{0\{1-9\}X\cdots X}^{p-1}, \overbrace{X\{1-9\}00\cdots0}^{p+1}\overbrace{0\{1-9\}X\cdots X}^{p-1}, \cdots, \overbrace{X\cdots X\{1-9\}}^{p+1}\overbrace{0\{1-9\}X\cdots X}^{p-1}, \\
\overbrace{\{1-9\}00\cdots0}^{p+1}\overbrace{00\{1-9\}X\cdots X}^{p-1}, \overbrace{X\{1-9\}00\cdots0}^{p+1}\overbrace{00\{1-9\}X\cdots X}^{p-1}, \cdots, \overbrace{X\cdots X\{1-9\}}^{p+1}\overbrace{00\{1-9\}X\cdots X}^{p-1}, \\
\vdots \\
\overbrace{\{1-9\}00\cdots0}^{p+1}\overbrace{00\cdots0\{1-9\}}^{p-1}, \overbrace{X\{1-9\}00\cdots0}^{p+1}\overbrace{00\cdots0\{1-9\}}^{p-1}, \cdots, \overbrace{X\cdots X\{1-9\}}^{p+1}\overbrace{00\cdots0\{1-9\}}^{p-1}
\end{gathered}
$$

8. It verifies the sticky bit when the right shift value is less than p using, (1) the right shift value is less than p, (2) sid indicates that the third input sigificand is the smaller exponent input of the addition operation, (3) the multiplication intermediate result significand has zero digits after the most p

digits, (4) the third input significand has the pattern $\overbrace{\{1-9\}00\cdots 0X}^{p}, \overbrace{X\{1-9\}00\cdots 0X}^{p}, \cdots, \overbrace{X\cdots X\{1-9\}0X}^{p}$.

D)Shift Model

The model aims to verify all the possible shifting of the input significands. The proposal ideas of the model are also in [22]. We separate the model into two sub-models as follows:

1. It verifies all the possible shifting to the third input significand using, (1) a list of right shift to the third input consists of the interval $[1, qmax-qmin]$, (2) sid indicates that the third input exponent is the smaller exponent input of the addition operation.

2. It verifies all the possible shifting to the multiplication intermediate result significand using, (1) a list of right shift to the multiplication intermediate result consists of the interval $[1, qmax-2*qmin]$, (2) sid indicates that the multiplication intermediate result exponent is the smaller exponent input of the addition operation.

E) Trailing and Leading Zeros Model

The model aims to verify all the possible trailing and leading zeros in the input significands and the addition intermediate result significand. The proposal ideas of the model are also in [22]. We separate the model into three sub-models as follows:

1. It verifies the different patterns of digits of the input significands using, (1) a list is from two values of sid, (2) a list of the third input significand, (3) a similar list of the multiplication intermediate result significand that has $2p$ digits. The second and the third lists have the same pattern

$$\overbrace{\{1-9\}00\cdots 00}^{p},\overbrace{0\{1-9\}00\cdots 00}^{p},\cdots,\overbrace{00\cdots 0\{1-9\}}^{p},$$
$$\overbrace{\{1-9\}\{1-9\}0\cdots 00}^{p},\overbrace{0\{1-9\}\{1-9\}0\cdots 00}^{p},\cdots,\overbrace{00\cdots 0\{1-9\}\{1-9\}}^{p},$$
$$\overbrace{\{1-9\}X\{1-9\}0\cdots 00}^{p},\overbrace{0\{1-9\}X\{1-9\}0\cdots 00}^{p},\cdots,\overbrace{00\cdots 0\{1-9\}X\{1-9\}}^{p},$$
$$\vdots$$
$$\overbrace{\{1-9\}XX\cdots X\{1-9\}}^{p}$$

2. It verifies different patterns of digits of the addition intermediate result significands using (1) a list of the addition intermediate result significand of p digits before fractional point, consists of similar pattern of the previous sub-model.

3. It verifies the final carry with different pattern of zeros in the addition intermediate result significand using, (1) a list of the addition intermediate result sigificand consists of the following patterns

$$\overbrace{1\{1-9\}00\cdots 00}^{p+1},\overbrace{10\{1-9\}00\cdots 00}^{p+1},\cdots,\overbrace{100\cdots 0\{1-9\}}^{p+1},\overbrace{100\cdots 00}^{p+1},$$
$$\overbrace{1\{1-9\}\{1-9\}0\cdots 00}^{p+1},\overbrace{10\{1-9\}\{1-9\}0\cdots 00}^{p+1},\cdots,\overbrace{100\cdots 0\{1-9\}\{1-9\}}^{p+1},$$
$$\overbrace{1\{1-9\}X\{1-9\}0\cdots 00}^{p+1},\overbrace{10\{1-9\}X\{1-9\}0\cdots 00}^{p+1},\cdots,\overbrace{100\cdots 0\{1-9\}X\{1-9\}}^{p+1},$$
$$\vdots$$
$$\overbrace{1XX\cdots X\{1-9\}}^{p+1}$$

F) Carry and Borrow model

The model aims to verify all the possible propagation of carries and borrows in the addition operation. The Ideas of the model are all new. We separate the model into four sub-models as follows:

1. It verifies all patterns of the borrow propagation when the addition operation is effective subtraction using, (1) a list of right shift values to the third input consists of the interval $[1,2p]$, (2) sid indicates that the third input exponent is the smaller exponent input of the addition operation, (3) a list of the multiplication intermediate result significand consists of the following pattern

$$\begin{gathered}
\overbrace{\{1-9\}00\cdots0X}^{2p},\overbrace{\{1-9\}00\cdots0XX}^{2p},\cdots,\overbrace{\{1-9\}X\cdots XX}^{2p},\\
\overbrace{X\{1-9\}0\cdots0X}^{2p},\overbrace{X\{1-9\}0\cdots0XX}^{2p},\cdots,\overbrace{X\{1-9\}X\cdots XX}^{2p},\\
\overbrace{XX\{1-9\}0\cdots0X}^{2p},\overbrace{XX\{1-9\}0\cdots0XX}^{2p},\cdots,\overbrace{XX\{1-9\}X\cdots XX}^{2p},\\
\vdots\\
\overbrace{XXX\cdots X\{1-9\}}^{2p}
\end{gathered}$$

2. It verifies all patterns of the borrow propagation when the addition operation is effective subtraction using, (1) a list of right shift to the multiplication intermediate result consists of the interval $[1,p]$, (2) sid indicates that the multiplication intermediate result exponent is the smaller exponent input of the addition operation, (3) a list of the third input significand consists of similar pattern to the pattern in sub-model 1, but with p digits.

3. It verifies all patterns of the carry propagation when the addition operation is effective addition using, (1) a list of right shift values to the third input in the interval $[1,2p]$, (2) sid indicates that the third input exponent is the smaller exponent input of the addition operation, (3) a list of the multiplication intermediate result significand consists of the following pattern.

$$\begin{gathered}
\overbrace{\{1-9\}99\cdots99}^{2p},\overbrace{\{1-9\}99\cdots99X}^{2p},\overbrace{\{1-9\}99\cdots9XX}^{2p},\cdots,\overbrace{\{1-9\}X\cdots XX}^{2p},\\
\overbrace{X\{1-9\}99\cdots99}^{2p},\overbrace{X\{1-9\}99\cdots99X}^{2p},\overbrace{X\{1-9\}99\cdots9XX}^{2p},\cdots,\overbrace{X\{1-9\}X\cdots XX}^{2p},\\
\overbrace{XX\{1-9\}99\cdots99}^{2p},\overbrace{\{1-9\}99\cdots99X}^{2p},\overbrace{XX\{1-9\}99\cdots9XX}^{2p},\cdots,\overbrace{XX\{1-9\}X\cdots XX}^{2p},\\
\vdots\\
\overbrace{XXX\cdots X\{1-9\}}^{2p}
\end{gathered}$$

4. It verifies all patterns of the carry propagation when the operation is effective addition using, (1) a list of right shift values to the multiplication result consists of the interval $[1,p]$. (2) sid indicates that the multiplication intermediate result exponent is the smaller exponent input of the addition operation, (3) a list of the third input significand of similar pattern to the pattern in sub-model 3, but with p digits.

G) Overflow Model

The model aims to verify all the overflow and the near overflow cases. We separate the model into four sub-models as follows:

1.It verifies the overflow cases due to the rounding process using, (1) the addition intermediate result significand is equal to 10^p-1, with a guard digit consists of the interval $[5,9]$, (2) the addition intermediate result exponent is equal to $qmax$, (3) a list is from two rounding modes Round ties to even and Round ties to away,(4) a list of the multiplication intermediate result exponent consists of the interval $[qmax-p, qmax]$.

2.It verifies the overflow cases due to the rounding process using, (1) the addition intermediate result significand is equal to 10^p-1, with a guard digit consists of the interval $[1,9]$, (2) the addition intermediate result exponent is equal to $qmax$, (3) two rounding modes are Round to positive and Round to negative, (4) a list of the multiplication intermediate result exponent consists of the interval $[qmax-p, qmax]$.

3. It verifies the overflow cases due to the final carry at the effective addition operation using, (1) the number of digits before fractional point of the addition intermediate result significand is equal to $p+1$, (2) the addition intermediate result exponent is equal to $qmax$, (3) a list of the multiplication intermediate result exponent consists of the interval $[qmax-p, qmax]$, (4) a list of number of digits of the third input significand consists of the interval $[1, p]$.

4. It verifies the overflow cases due to the result of the multiplication operation using, (1) a list of the multiplication intermediate result exponent consists of the interval $[qmax-p, 2*qmax]$. The proposal idea of this sub-model is in [22].

H) Clamping Model

The clamping occurs when the intermediate result exponent is larger than $qmax$, and the number of digits of the intermediate result significand is less than p, such that the sum of the intermediate result exponent to the number of digits of the intermediate result significand is less than or equal to $qmax+p$. At that case, the engine shifts to left the intermediate result significand and reduces the number of leading zeros.

The model aims to verify all clamping cases. We separate the model into two sub-models as follows:

1. It verifies the clamping case using, (1) a list of the multiplication intermediate result exponent consists of the interval $[qmax+1, qmax+p-1]$, (2) a list of number of digits of the multiplication intermediate result significand consists of the interval $[1, p]$, (3) the multiplication intermediate result exponent to the number of digits of the multiplication intermediate result signicand is less than or equal to $qmax+p$, (4) a list of third input significand consists of $\{zero, random\, number\}$, (5) the third input exponent is equal to $qmax$.

2. It verifies the cases of left shift to the addition intermediate result significand due to the preferred exponent condition using, (1) a list of the multiplication intermediate result exponent consists of the interval $[qmin+1, qmax+p]$, (2) a random value of number of digits of the multiplication intermediate result significand from the interval $[1, p]$, (3) the third input significand is equal to zero, (4) the third input exponent is less than the multiplication intermediate result exponent.

I) Underflow Model

The model aims to verify all the underflow and the near underflow cases. We separate the model into three sub-models as follows:

1. It verifies the underflow due to the result of the multiplication operation using, (1) a list of the multiplication intermediate result exponents consists of the interval $[2*qmin, qmin]$, (2) a list of third input significand consists of $\{zero, random\, number\}$. The proposal idea of this sub-model is in [22].

2. It verifies the underflow flag when the result is inexact and the result exponent is equal to $qmin$ using, (1) a list of the multiplication intermediate result exponent consists of the interval $[qmin-2\text{p}, qmin]$, (2) a list of number of digits of the multiplication intermediate result consists of the interval $[1,2p]$, (3) the third input significand is equal to zero.

3. It verifies the underflow flag when the result is exact and the result exponent is equal to $qmin$ using, (1) a list of the multiplication intermediate result exponent consists of the interval $[qmin-2\mathrm{p}, qmin]$, (2) a list of the multiplication intermediate result significand consists of the pattern $\{\{1-9\}00\cdots0, X\{1-9\}00\cdots0,\cdots, XX\cdots X\{1-9\}\}$, (3) the third input significand is equal to zero.

J) Cancellation Model

The model aims to verify all the cancellation cases, which has cancellation digits in the most digits of the addition intermediate result due to the effective subtraction operation. We separate the model into ten sub-models as follows:

1. It verifies the cases of all possible number of the cancellation digits using, (1) a list of the addition intermediate result significand consists of an interval of number of digits before the fractional point $[1, p-1]$, and an interval of number of digits after the fractional point $[1, p-1]$ at zero value before the fractional point, (2) a list of right shift consists of the interval $[0,1]$, (3) a list of number of digits of the multiplication intermediate result significand consists of the interval $[1,2p]$, (4) sid identifies the third input exponent as the smaller addition exponent. The proposal idea of this sub-model is in [22].

2. It verifies the cases of all possible number of the cancellation digits, (1) a list of the addition intermediate result significand similar to the list in sub-model 1, (2) a list of right shifts consists of the interval $[0,1]$, (3) a list of number of digits of the third input significand consists of the interval $[1, p]$, (4) sid identifies the multiplication intermediate result exponent as the smaller exponent. The proposal idea of this sub-model is in [22].

3. It verifies the zero result due to cancellation using, (1) the addition intermediate result significand is equal to zero value, (2) the right shift is zero, (3) a list of number of digits of the multiplication intermediate result significand consists of the interval $[1,2p]$. The proposal idea of this sub-model is in [22].

4. It verifies the cases when the result is exact due to cancellation using, (1) the addition intermediate result significand has zero value after the fractional point, (2)a list of the multiplication intermediate result significand consists of the pattern

$$\overbrace{\{1-9\}XX\cdots X}^{p}\overbrace{000\cdots 0}^{p-1}\{1-9\},\overbrace{\{1-9\}XX\cdots X}^{p}\overbrace{00\cdots 0}^{p-2}\{1-9\}X,$$
$$\overbrace{\{1-9\}XX\cdots X}^{p}\overbrace{00\cdots 0}^{p-3}\{1-9\}XX,\cdots,\overbrace{\{1-9\}XX\cdots X}^{p}\{1-9\}XX\cdots X$$

(3) a list of right shift to the third input significand consists the interval $[p, 2\mathrm{p}-1]$, (4) sid identifies the third input exponent as the smaller addition exponent.

5. It verifies the cases when the result is exact due to the cancellation using, (1) the addition intermediate result significand has zero value after point, (2) a list of the multiplication intermediate result significand consists of the pattern

$$\overbrace{\{1-9\}XX\cdots X}^{p}\{1-9\}00\cdots 0,\overbrace{\{1-9\}XX\cdots X}^{p}X\{1-9\}00\cdots 0,$$
$$\overbrace{\{1-9\}XX\cdots X}^{p}XX\{1-9\}00\cdots 0,\cdots,\overbrace{\{1-9\}XX\cdots X}^{p}XX\cdots X\{1-9\}$$

(3) a list of right shift to the third input significand from the interval $[1, p]$, (4) sid identifies the third input exponent as the smaller addition exponent.

6. It verifies the underflow cases due to cancellation using, (1) a list of the addition intermediate result significand consists of the interval of number of digits before fractional point $[1, p-1]$, and the interval of number of digits after point $[1, p]$, (2) a list of values of the addition intermediate result exponent in the interval $[qmin, qmin+p-1]$, (3) a list of right shift consists of the interval $[0,1]$. The proposal idea of this sub-model is in [22].

7. It verifies the underflow due to cancellation using, (1) One cancellation digit in the addition intermediate result significand (2) the addition intermediate result exponent is equal to $qmin$, (3) a list of the multiplication intermediate result exponent consists of the interval $[2*qmin, qmin+1]$.

8. It verifies the near overflow cases with cancellation using, (1) a list of the addition intermediate result significand consists of the interval of number of

digits before point $[1, p-1]$, (2) a right shift is equal to one, (3) the addition intermediate result exponent is equal to $qmax+1$.

9. It verifies the cancellation cases with one digit using, (1) one cancellation digit in the addition intermediate result significand, (2) a list of right shift from the interval $[2, qmax-2*qmin]$, (3) sid identifies the multiplication result exponent as the smaller exponent. The proposal idea of this sub-model is in [22].

10. It verifies the cancellation cases with one digit using, (1) one cancellation digit in the addition intermediate result significand, (2) a list of right shift from the interval $[2, qmax-qmin]$, (3) sid identifies the third input exponent as the smaller exponent. The proposal idea of this sub-model is in [22].

4.3 Summary

This chapter represents the main steps of the first FMA engine to solve all the constraints numerically. It also describes the main ideas of the coverage models that have been solved by the engine to generate test vectors can verify all the corner cases in the hardware or software implementations of the decimal floating-point FMA operation.

The engine cannot find the solution from the first trial, and may not solve all the constraints on the least digits of the multiplication intermediate result that have weight less than 10^{p-1}.

The engine solved the coverage models one time and generated about 425000 test vectors in Decimal64, the test vectors have proved their efficiency by discovering bugs in Silminds design and FMA DecNumber implementation. The DecNumber bugs are discovered using the carry and borrow model, while most of Silminds bugs are discovered using the overflow and the underflow models.

Chapter 5

Engine and Models of Decimal Square Root Operation

The square root engine is a software tool written in C++ to generate square root test vectors can cover all corner cases, to verify a tested implementation of decimal square root operation to achieve the compliance with the IEEE standard (754-2008) for Floating Point Arithmetic, it takes coverage models as inputs and generates test vectors as outputs.

The engine generates the test vectors in two formats of the IEEE standard: Decimal64 and Decimal128. The engine time to generate one test vector depends on the constraints that have been solved to generate it and the factor of randomization that the engine needed. The engine generates as many test vectors as the user wants. Every time the engine runs, it generates new test vectors. The verification engine value is neither in the time needed to generate the test vector, if this time is practical, nor in the number of the generated test vectors, but rather in the functionality of the cases that the test vector covers.

The engine solved the coverage models one time and generated about 50000 test vectors in Decimal64 and about 199000 test vectors in Decimal128, the test vectors have proved an efficiency by discovering bugs in DecNumber library[23] and Silminds design [7]. Table 2 shows the maximum and the minimum times that the engine needed to solve a task of the existing constraints and generate one test vector, on Intel(R) Pentium(R) 4 CPU 3.20GHZ with g++ (Ubuntu 4.4.3) compiler.

TABLE 2. THE TIME PERFORMANCE OF THE SQUARE ROOT ENGINE

Test vector Format	Minimum Time	Maximum Time
Decimal 64	0.006 seconds	37 seconds
Decimal 128	0.017 seconds	2.35 minutes

Although the engine solves constraints on the input and the intermediate result only, it managed to discover some faults inside the operation in two designs by forcing the engine to solve constraints on patterns of zeros and nines in the intermediate result significand.

The generated test vector is a decimal vector that has three sets, The first set is type of the operation square root, number of the precision (64 or 128), and the rounding mode. The second set is sign, significand, and exponent of the input. The third set is sign, significand, and exponent of the result. Finally the fourth set is one of two flags(invalid, inexact). The designer enters the input set to his implementation and verifies his output against the last two sets.

The task given to the square root engine is the set of constraints on four elements, the significand of the input Sx, the intermediate result significand Sz, the exponent of the input, and the rounding mode. The constraint on Sx is a mask starting from the minimum number Nx to the maximum number Mx. Similarly, the mask on Sz consists of two numbers Nz and Mz. The input exponent and the rounding direction are either given explicitly in the task or left to the engine to choose randomly.

An example to explain the format of the decimal square root task at $p=16$ is as follows:

$$\begin{array}{c} 64\mathrm{V}\,T: \quad +1000000000 \quad +9999999999 \\ +0000000000009999.600000000000000000000000 \\ +9999999999999999.699999999999999999999999 \quad R \quad R \end{array}$$

This task means that $Nx=+1000000000$, $Mx=+9999999999$, $Nz=+0000000000009999.600000000000000000000000$, $Mz=+9999999999999999.699999999999999999999999$, the engine chooses randomly the exponent of the input, and it chooses randomly the rounding mode.

One of the solutions of this task is the test vector $d64V$ 0 +3425834081E146 -> +5853062515469999E62 X. The $d64$ means decimal64, the V means the square root operation, the following *0* means that the rounding mode is Round toward Zero, the input is $x=+3425834081*10^{146}$, the rounded result is $z=+5853062515469999*10^{62}$, and the following X indicates that the inexact flag is high, because the exact result is $+585306251546999965921080720938974330 1409\cdots$.

We represent the intermediate result with length $2.5p$ digits not including the leading zeros to guarantee that the engine can generate all the possible hardest-to-round cases, where the hardest-to round case needs only $2\mathrm{p}-1$ digits not including leading zeros to do the rounding process according to the standard.

5.1 The Square Root Engine

The inverse operation of the square root is the multiplication of the intermediate result with itself which gives the input of the square root operation. The engine is based on solving the non linear equations that result from multiplying the intermediate result with itself. We can estimate these non linear equations from Figure 4, where each column represents one nonlinear equation. The figure shows the squarer of the intermediate result at $p=16$, where Sz_i denotes the intermediate result digit of weight 10^i, and Sx_i denotes the input digit of weight 10^i.

The engine uses $2.5p$ digits only for the intermediate result significand Sz. Hence, if the infinity precise square root of the input significand Sx has more digits, then Sz is truncated, i.e. it is slightly less than the infinitely precise square root. The square of Sz will thus be $Sx-\Delta$ with $0<\Delta\leq 10^{-L}$ where L depends on the number of digits of Sz. This explains the series of nines that follows Sx_0-1 as seen in Figure 4. Also if the input exponent is odd, the engine shifts the input significand one digit to the left which explains that Sx_p may exist. For example if the input is $x=8116261898426249*10^{351}$ which has *16* digits in the input significand, the engine solves it as $x=81162618984262490*10^{350}$ which has *17* digits.

The square of the most significant digit of Sz such as in Figure 4 should be on a column with an even index for Sx. If $Sz_8<4$, that squaring does not generate a carry into a higher position. Otherwise, if $Sz_8\geq 4$, its square generates a carry into position Sx_{17}. Note that even if $Sz_8=3$ (the square is 9) a carry into the position of $Sz_8 Sz_8$ will lead to a carry out into the position of

Sx_{17}. So, in general, if the position formula w_x of the most significant nonzero digit Sx_{w_x} of the input is odd then, Sx_{w_x} is a carry from the first nonlinear equation.

The engine steps begin by choosing the input exponent formula E_x according to its constraints. If the input exponent is odd, the engine shifts Nx and Mx by one digit to the left, and subtracts one from E_x.

Then, the engine gets the intermediate result significand Sz and the input significand Sx that achieve the constraints. It achieves the constraint on each digit Sx_n or Sz_n by choosing the digit from its interval formula $[Nx_n, Mx_n]$ or formula $[Nz_n, Mz_n]$. It solves the significands constraints using one of two algorithms, the first algorithm is the Square-Root-Most-Digits-Constraints-Algorithm to solve the constraints on the most significant p digits of the intermediate result significand and the $p+1$ digits of the input significand,

								$*\ Sz_8$	Sz_7	Sz_6	Sz_5	Sz_4	Sz_3	Sz_2	Sz_1	Sz_0	$Sz_{-1}\cdots$
								Sz_8	Sz_7	Sz_6	Sz_5	Sz_4	Sz_3	Sz_2	Sz_1	Sz_0	$Sz_{-1}\cdots$
Sz_8Sz_8	Sz_8Sz_7	Sz_8Sz_6	Sz_8Sz_5	Sz_8Sz_4	Sz_8Sz_3	Sz_8Sz_2	Sz_8Sz_1	Sz_8Sz_0	Sz_8Sz_{-1}	Sz_8Sz_{-2}	Sz_8Sz_{-3}	Sz_8Sz_{-4}	Sz_8Sz_{-5}	Sz_8Sz_{-6}	Sz_8Sz_{-7}	Sz_8Sz_{-8}	$Sz_8Sz_{-9}\cdots$
	Sz_7Sz_8	Sz_7Sz_7	Sz_7Sz_6	Sz_7Sz_5	Sz_7Sz_4	Sz_7Sz_3	Sz_7Sz_2	Sz_7Sz_1	Sz_7Sz_0	Sz_7Sz_{-1}	Sz_7Sz_{-2}	Sz_7Sz_{-3}	Sz_7Sz_{-4}	Sz_7Sz_{-5}	Sz_7Sz_{-6}	Sz_7Sz_{-7}	$Sz_7Sz_{-8}\cdots$
		Sz_6Sz_8	Sz_6Sz_7	Sz_6Sz_6	Sz_6Sz_5	Sz_6Sz_4	Sz_6Sz_3	Sz_6Sz_2	Sz_6Sz_1	Sz_6Sz_0	Sz_6Sz_{-1}	Sz_6z_{-2}	Sz_6Sz_{-3}	Sz_6Sz_{-4}	Sz_6Sz_{-5}	Sz_6Sz_{-6}	$Sz_6Sz_{-7}\cdots$
			Sz_5Sz_8	Sz_5Sz_7	Sz_5Sz_6	Sz_5z_5	Sz_5Sz_4	Sz_5Sz_3	Sz_5Sz_2	Sz_5Sz_1	Sz_5Sz_0	Sz_5Sz_{-1}	Sz_5Sz_{-2}	Sz_5Sz_{-3}	Sz_5Sz_{-4}	Sz_5Sz_{-5}	$Sz_5Sz_{-6}\cdots$
				Sz_4Sz_8	Sz_4Sz_7	Sz_4Sz_6	Sz_4Sz_5	Sz_4Sz_4	Sz_4Sz_3	Sz_4Sz_2	Sz_4Sz_1	Sz_4Sz_0	Sz_4Sz_{-1}	Sz_4Sz_{-2}	Sz_4Sz_{-3}	Sz_4Sz_{-4}	$Sz_4Sz_{-5}\cdots$
					Sz_3Sz_8	Sz_3Sz_7	Sz_3Sz_6	Sz_3Sz_5	Sz_3Sz_4	Sz_3Sz_3	Sz_3Sz_2	Sz_3Sz_1	Sz_3Sz_0	Sz_3Sz_{-1}	Sz_3Sz_{-2}	Sz_3Sz_{-3}	$Sz_3Sz_{-4}\cdots$
						Sz_2Sz_8	Sz_2Sz_7	Sz_2Sz_6	Sz_2Sz_5	Sz_2Sz_4	Sz_2Sz_3	Sz_2Sz_2	Sz_2Sz_1	Sz_2Sz_0	Sz_2Sz_{-1}	Sz_2Sz_{-2}	$Sz_2Sz_{-3}\cdots$
							Sz_1Sz_8	Sz_1Sz_7	Sz_1Sz_6	Sz_1Sz_5	Sz_1Sz_4	Sz_1Sz_3	Sz_1Sz_2	Sz_1Sz_1	Sz_1Sz_0	Sz_1Sz_{-1}	$Sz_1Sz_{-2}\cdots$
								Sz_0Sz_8	Sz_0Sz_7	Sz_0Sz_6	Sz_0Sz_5	Sz_0Sz_4	Sz_0Sz_3	Sz_0Sz_2	Sz_0Sz_1	Sz_0Sz_0	$Sz_0Sz_{-1}\cdots$
									$Sz_{-1}Sz_8$	$Sz_{-1}Sz_7$	$Sz_{-1}Sz_6$	$Sz_{-1}Sz_5$	$Sz_{-1}Sz_4$	$Sz_{-1}Sz_3$	$Sz_{-1}Sz_2$	$Sz_{-1}Sz_1$	$Sz_{-1}Sz_0\cdots$
										$Sz_{-2}Sz_8$	$Sz_{-2}Sz_7$	$Sz_{-2}Sz_6$	$Sz_{-2}Sz_5$	$Sz_{-2}Sz_4$	$Sz_{-2}Sz_3$	$Sz_{-2}Sz_2$	$Sz_{-2}Sz_1\cdots$
											$Sz_{-3}Sz_8$	$Sz_{-3}Sz_7$	$Sz_{-3}Sz_6$	$Sz_{-3}Sz_5$	$Sz_{-3}Sz_4$	$Sz_{-3}Sz_3$	$Sz_{-3}Sz_2\cdots$
												$Sz_{-4}Sz_8$	$Sz_{-4}Sz_7$	$Sz_{-4}Sz_6$	$Sz_{-4}Sz_5$	$Sz_{-4}Sz_4$	$Sz_{-4}Sz_3\cdots$
													$Sz_{-5}Sz_8$	$Sz_{-5}Sz_7$	$Sz_{-5}Sz_6$	$Sz_{-5}Sz_5$	$Sz_{-5}Sz_4\cdots$
														$Sz_{-6}Sz_8$	$Sz_{-6}Sz_7$	$Sz_{-6}Sz_6$	$Sz_{-6}Sz_5\cdots$
															$Sz_{-7}Sz_8$	$Sz_{-7}Sz_7$	$Sz_{-7}Sz_6\cdots$
																$Sz_{-8}Sz_8$	$Sz_{-8}Sz_7\cdots$
																	$Sz_{-9}Sz_8\cdots$
Sx_{16}	Sx_{15}	Sx_{14}	Sx_{13}	Sx_{12}	Sx_{11}	Sx_{10}	Sx_9	Sx_8	Sx_7	Sx_6	Sx_5	Sx_4	Sx_3	Sx_2	Sx_1	Sx_0-1	$9\ \cdots$

Figure 4. The squarer of the Intermediate Result assuming Precision 16

the second algorithm is the Square-Root-Least-Digits Constraints-Algorithm to solve the constraints on the least significant digits that follow the highest p digits of the intermediate result significand.

After the engine gets the significand value of Sx and Sz it shifts to left the significand formula Sz by $p-w_x/2$ and calculates the result exponent formula $E_z=E_x/2-p+w_x/2$, if the result is inexact. If the input exponent is odd, the engine shifts Sx by one digit to right and increases E_x by one.

5.1.1The Square Root Most Digits Constraints Algorithm

The algorithm iterates to solve the nonlinear equations from left to right. As shown in Figure 4, for $p=16,$ the first non linear equation from left is

$$Sx_{16}-Sz_8*Sz_8=br_{16} \qquad (5.1)$$

where br_{16} is the value of carries that transfer from previous weights to the weight of $10^{16},$ or the borrow generated from this weight to lower weights. The second and the third non linear equations are:

$$Sx_{15}+10*br_{16}-2*Sz_7*Sz_8=br_{15} \qquad (5.2)$$

$$Sx_{14}+10*br_{15}-2*Sz_6*Sz_8-Sz_7*Sz_7=br_{14}. \qquad (5.3)$$

In general the nonlinear equation for the column of index n is :

$$br_n=Sx_n+10*br_{n+1}-\sum_{j=n-w_x/2}^{w_x/2} Sz_j*Sz_{n-j}, \qquad (5.4)$$

To start the solution, the algorithm attempts to solve equations 5.1 to 5.3 (representing columns 16 to 14) together based on the range of carries that may transfer from the next lower significant columns. The algorithm chooses the digit Sx_{16} and the digit Sx_{15} randomly from their intervals. Then since the ranges of borrow digit br_{14} and the digit Sz_6 are known as $Ncr_{14}\leq br_{14}\leq Mcr_{14}$ and $Nz_6\leq Sz_6\leq Mz_6,$ the algorithm transforms Equation 5.3 to the inequality condition:

$$Ncr_{14}+2*Nz_6*Sz_8\leq Sx_{14}+10*br_{15}-Sz_7*Sz_7\leq Mcr_{14}+2*Mz_6*Sz_8. \qquad (5.5)$$

Finally, it searches randomly on the values of $Sz_8,$ $Sz_7,$ Sx_{14} that satisfy Equation 5.1, Equation 5.2 and the Inequality 5.5. The steps taken so far constitute the first outer iteration that gets the final values of $Sz_8,$ $Sx_{16},$ $Sx_{15},$ Sx_{14} and estimates the value of Sz_7 that may be refined in the following iteration.

In the second iteration, the algorithm transforms the fourth nonlinear equation $Sx_{13}+10*br_{14}-2*Sz_5*Sz_8-2*Sz_6*Sz_7=br_{13}$ to

$$Ncr_{13}+2*Nz_5*Sz_8\leq Sx_{13}+10*br_{14}-2*Sz_6*Sz_7\leq Mcr_{13}+2*Mz_5*Sz_8,$$

and searches randomly on the values of Sz_7, Sz_6, Sx_{13} that achieve the second nonlinear equation, the third nonlinear equation and the inequality condition, where the digits Sz_8, Sb_{16}, Sx_{16}, Sx_{15}, Sx_{14}, are known from the previous iteration. The algorithm does this procedure in all the iterations and gets all digits of Sx and Sz.

In general, for any precision, the algorithm gets randomly the first two digits of Sx, which are Sx_{w_x} and Sx_{w_x-1} from their intervals. If w_x is odd, it gets randomly the digit Sx_{w_x-2}, replaces Sx_{w_x-1} with $Sx_{w_x-1}+10*Sx_{w_x}$, and replaces w_x with w_x-1.

Then, it loops through a number of outer iterations equal to the number of nonlinear equations(i.e number of columns). The index of the outer iterations goes from formula $1 \le i \le 2.5\mathrm{p}$. The algorithm gets in iteration i the values of $Sz_{w_x/2-i+1}$ and Sx_{w_x-i-1} and estimates the value of $Sz_{w_x/2-i}$. Then, in the next iteration it gets the values of $Sz_{w_x/2-i}$ and Sx_{w_x-i-2} and estimates $Sz_{w_x/2-i-1}$, and so on.

The general form of Equation 5.1, at iteration i, is

$$br_{w_x-i+1}=Sx_{w_x-i+1}-\sum_{j=w_x/2-i+1}^{w_x/2} Sz_j*Sz_{w_x-i+1-j}. \tag{5.6}$$

Equation 5.6 calculates the borrow from the column of index w_x-i+1. The equation has one unknown br_{w_x-i+1} (i.e the borrow of the column), while the other elements of the equation are known from the previous iterations and the value $Sz_{w_x/2-i+1}$.

The general form of Equation 5.2, at iteration formula i, is

$$br_{w_x-i}=Sx_{w_x-i}+10*br_{w_x-i+1}-\sum_{j=w_x/2-i}^{w_x/2} Sz_j*Sz_{w_x-i-j}, \tag{5.7}$$

which calculates the borrow from the column of index w_x-i. The equation has one unknown br_{w_x-i} (i.e the borrow of the column), while the other elements of the equation are known from the previous iterations, the values of $Sz_{w_x/2-i+1}, Sz_{w_x/2-i}$, and the value of br_{w_x-i+1} from Equation 5.6.

Similarly, the general form of Equation 5.3, at iteration formula i, is

$$br_{w_x-i-1}=Sx_{w_x-i-1}+10*br_{w_x-i}-\sum_{j=w_x/2-i-1}^{w_x/2} Sz_j*Sz_{w_x-i-1-j}. \quad (5.8)$$

As the ranges of br_{w_x-i-1} and $Sz_{w_x/2-i-1}$ are known, the algorithm transforms Equation 5.8 to inequality 5.9, which is the general form of inequality 5.5.

$$\begin{aligned} &Ncr_{w_x-i-2}+Ncr_{w_x-i-3}+Ncr_{w_x-i-4}+2*Sz_{w_x/2}*Nz_{w_x/2-i-1}\leq \\ &Sx_{w_x-i-1}+10*br_{w_x-i}-\sum_{j=w_x/2-i}^{w_x/2-1} Sz_j*Sz_{w_x-i-1-j} \\ &\leq 2*Sz_{w_x/2}*Mz_{w_x/2-i-1}+Mcr_{w_x-i-2}+Mcr_{w_x-i-3}+Mcr_{w_x-i-4}+1 \end{aligned} \quad (5.9)$$

Within each outer iteration, the engine does a second level of iterations to get the values of Sx_{w_x-i-1}, $Sz_{w_x/2-i+1}$, $Sz_{w_x/2-i}$ that achieve at each outer iteration inequality 5.9. At this second level of iterations, the engine just chooses random numbers from the intervals of Sx_{w_x-i-1}, $Sz_{w_x/2-i+1}$, $Sz_{w_x/2-i}$. If these numbers do not satisfy inequality 5.9, it chooses another combination of numbers, and so on until it finds a set of numbers that satisfy this inequality.

The range of br_{w_x-i-1} is the range of the carries that transfer from the columns following the column w_x-i-1. Since the algorithm solves only $2.5p$ columns, the maximum product sum of any column at $p=34$ is equal to $2.5*34*9*9=6685$. This number means that a carry from any column,at $p\leq 34$, may affect the previous three columns directly by a value more than one and affects the higher columns indirectly by a value less than or equal to one. Based on that, the algorithm determines the range of carries that transfer to the column formula w_x-i-1 from the next three columns formula w_x-i-2, w_x-i-3, w_x-i-4.

Equation 5.10 and Equation 5.11 get the maximum and the minimum carries formula $Mcr_{w_x-i-2}, Ncr_{w_x-i-2}$ from the column of index formula w_x-i-2 to the column of index formula w_x-i-1.

$$Mcr_{w_x-i-2}=\frac{\sum_{j=w_x/2-1}^{w_x/2} 2*Sz_j*Mz_{w_x-i-2-j}+\sum_{j=w_x/2-i}^{w_x/2-2} Sz_j*Sz_{w_x-i-2-j}}{10}, \quad (5.10)$$

$$Ncr_{w_x-i-2}=\frac{\sum_{j=w_x/2-1}^{w_x/2} 2*Sz_j*Nz_{w_x-i-2-j}+\sum_{j=w_x/2-i}^{w_x/2-2} Sz_j*z_{w_x-i-2-j}}{10}, \tag{5.11}$$

Equation 5.12 and Equation 5.13 get the maximum and the minimum carries formula $Mcr_{w_x-i-3}, Ncr_{w_x-i-3}$ from the column of index formula w_x-i-3 to the column of index formula w_x-i-1.

$$Mcr_{w_x-i-3}=\frac{\sum_{j=w_x/2-2}^{w_x/2} 2*Sz_j*Mz_{w_x-i-3-j}+\sum_{j=w_x/2-i}^{w_x/2-3} Sz_j*Sz_{w_x-i-3-j}}{100}, \tag{5.12}$$

$$Ncr_{w_x-i-3}=\frac{\sum_{j=w_x/2-2}^{w_x/2} 2*Sz_j*Nz_{w_x-i-3-j}+\sum_{j=w_x/2-i}^{w_x/2-3} Sz_j*Sz_{w_x-i-3-j}}{100}, \tag{5.13}$$

Equation 5.14 and Equation 5.15 get the maximum and the minimum carries formula $Mcr_{w_x-i-4}, Ncr_{w_x-i-4}$ from the column of index formula w_x-i-4 to the column of index formula w_x-i-1.

$$Mcr_{w_x-i-4}=\frac{\sum_{j=w_x/2-3}^{w_x/2} 2*Sz_j*Mz_{w_x-i-4-j}+\sum_{j=w_x/2-i}^{w_x/2-4} Sz_j*Sz_{w_x-i-4-j}}{1000}, \tag{5.14}$$

$$Ncr_{w_x-i-4}=\frac{\sum_{j=w_x/2-3}^{w_x/2} 2*Sz_j*Nz_{w_x-i-4-j}+\sum_{j=w_x/2-i}^{w_x/2-4} Sz_j*Sz_{w_x-i-4-j}}{1000}, \tag{5.15}$$

After getting the iteration values Sx_{w_x-i-1}, $Sz_{w_x/2-i+1}$, $Sz_{w_x/2-i}$, the algorithm propagates the borrows between the digits of Sx to be in the form of the general Equations 6.15 to 8.15 It replaces formula Sx_{w_x-i+1} with formula $Sx_{w_x-i+1}-br_{w_x-i+1}$, formula Sx_{w_x-i} with formula $Sx_{w_x-i}+10*br_{w_x-i+1}-br_{w_x-i}$, and formula Sx_{w_x-i-1} with formula $Sx_{w_x-i-1}+10*br_{w_x-i}$. Then, the algorithm begins the next outer iteration using the same procedure, and so on until it gets all digits of Sx and Sz.

5.1.2The Square Root least Digits Constraints Algorithm

The previous algorithm gets the digits of Sx that satisfy the constraints on the most significant digits of Sz and do not take the constraints of the least digits of Sz in its calculations. Hence, in case there are constraints on the least significant digits of the intermediate result significand Sz (that have weight less than $10^{w_s/2-p}$), the previous algorithm alone will not succeed to get a solution in some hard constraints. An example of the hard constraints is a series of zeros or nines in the least digits of Sz, which are needed to verify the rounding process in the different designs.

The Square Root least digits algorithm gives the value of the input significand Sx, which yields the needed hard constraints in the intermediate result significand Sz. This algorithm solves the series of zeros constraint and the series of nines constraint in similar ways starting from right (least significant) to left.

As shown in Figure 5, the intermediate result significand Sz has a series of zeros from the weight 10^{-9} to 10^{-19}, due to this series of zeros, the elements are decreased in the columns of indexes from -2 to -12. The algorithm solves the nonlinear equations of the columns of indexes from -12 to -1, to get the digits of Sz from Sz_{-8} to Sz_7.

The algorithm gets randomly the elements of the products in the column of index -12, which are Sz_{-8}, Sz_{-7}, Sz_{-6}, Sz_{-5}, and Sz_{-4} from their intervals. It calculates the carries cr_{-12}, cr_{-13}, and cr_{-14} of the columns of indexes $-12, -13$, and -14, then replaces cr_{-12} with $cr_{-12}+cr_{-13}/10+cr_{-14}/100$, such that formula $cr_{-12} mod_{10}=0$.

Then, the algorithm attempts to solve the non linear equations of the columns of indexes $-11, -10, -9$. It searches randomly on the combination of values of Sz_{-3}, Sz_{-2}, Sz_{-1} that achieves the conditions $cr_{-11} mod_{10}=0$, $cr_{-10} mod_{10}=0$, and $cr_{-9} mod_{10}=0$. Up to now, the algorithm does the first iteration, gets the digit Sz_{-3}, and estimates the digits Sz_{-2}, Sz_{-1}. In the second iteration, it searches randomly on the values of Sz_{-2}, Sz_{-1}, Sz_0 that achieve the nonlinear equations of the columns of indexes $-10, -9, -8$, to get the digit value of z_{-2}, and estimates the digits Sz_{-1}, Sz_0. The algorithm does this procedure in all iterations to get the remaining digits of Sz, from Sz_{-1} to Sz_7.

The general form of the nonlinear equations is:

$$cr_n=\sum_{j=n-w_x/2}^{w_x/2} Sz_j * Sz_{n-j}+cr_{n-1}/10-Sx_n, \tag{5.16}$$

In general, the algorithm determines the series of zeros after the most p digits in the mask of the intermediate result significand formula Mz, Nz. The weight of the first zero from the left is denoted by formula 10^{Fw} and the weight of the last zero in the series is denoted by formula 10^{Lw}. It gets the digits from Sz_{Fw+1}, to $Sz_{w_x/2-1-Fw+Lw}$, which are the elements of the products of the column of index formula $W_x/2+Lw$.

Equation 5.17 gets the value of the carry generated from the column of index formula $W_x/2+Lw$.

	$10^{w_x/2}=10^7$ ↑										$10^{Fw}=10^{-9}$ ↑					$10^{Lw}=10^{-19}$ ↑
	$Sz_7\cdots Sz_1$	Sz_0	Sz_{-1}	Sz_{-2}	Sz_{-3}	Sz_{-4}	Sz_{-5}	Sz_{-6}	Sz_{-7}	Sz_{-8}	0	0	0	0	0	$0\ \cdots 0\ Sz_{-20}\cdots$
*	$Sz_7\cdots Sz_1$	Sz_0	Sz_{-1}	Sz_{-2}	Sz_{-3}	Sz_{-4}	Sz_{-5}	Sz_{-6}	Sz_{-7}	Sz_{-8}	0	0	0	0	0	$0\ \cdots 0\ Sz_{-20}\cdots$
		$\cdots Sz_7Sz_{-7}$	Sz_7Sz_{-8}	0	0	0	0	0	0	0	0	0	0	0	$2*Sz_7Sz_{-20}$	$2*Sz_7Sz_{-21}\cdots$
		$\cdots Sz_6Sz_{-6}$	Sz_6Sz_{-7}	Sz_6Sz_{-8}	0	0	0	0	0	0	0	0	0	0	0	$2*Sz_6Sz_{-20}\cdots$
		$\cdots Sz_5Sz_{-5}$	Sz_5Sz_{-6}	Sz_5Sz_{-7}	Sz_5Sz_{-8}	0	0	0	0	0	0	0	0	0	0	$0\cdots$
		$\cdots Sz_4Sz_{-4}$	Sz_4Sz_{-5}	Sz_4Sz_{-6}	Sz_4Sz_{-7}	Sz_4z_{-8}	0	0	0	0	0	0	0	0	0	$0\cdots$
		$\cdots Sz_3Sz_{-3}$	Sz_3Sz_{-4}	Sz_3Sz_{-5}	Sz_3Sz_{-6}	Sz_3Sz_{-7}	Sz_3Sz_{-8}	0	0	0	0	0	0	0	0	$0\cdots$
		$\cdots Sz_2Sz_{-2}$	Sz_2Sz_{-3}	Sz_2Sz_{-4}	Sz_2Sz_{-5}	Sz_2Sz_{-6}	Sz_2Sz_{-7}	Sz_2Sz_{-8}	0	0	0	0	0	0	0	$0\cdots$
		$\cdots Sz_1Sz_{-1}$	Sz_1Sz_{-2}	Sz_1Sz_{-3}	Sz_1Sz_{-4}	Sz_1Sz_{-5}	Sz_1Sz_{-6}	Sz_1Sz_{-7}	Sz_1Sz_{-8}	0	0	0	0	0	0	$0\cdots$
		$\cdots Sz_0Sz_0$	Sz_0Sz_{-1}	Sz_0Sz_{-2}	Sz_0Sz_{-3}	Sz_0Sz_{-4}	Sz_0Sz_{-5}	Sz_0Sz_{-6}	Sz_0Sz_{-7}	Sz_0Sz_{-8}	0	0	0	0	0	$0\cdots$
		$\cdots Sz_{-1}Sz_1$	$Sz_{-1}Sz_0$	$Sz_{-1}Sz_{-1}$	$Sz_{-1}Sz_{-2}$	$Sz_{-1}Sz_{-3}$	$Sz_{-1}Sz_{-4}$	$Sz_{-1}Sz_{-5}$	$Sz_{-1}Sz_{-6}$	$Sz_{-1}Sz_{-7}$	$Sz_{-1}Sz_{-8}$	0	0	0	0	$0\cdots$
		$\cdots Sz_{-2}Sz_2$	$Sz_{-2}Sz_1$	$Sz_{-2}Sz_0$	$Sz_{-2}Sz_{-1}$	$Sz_{-2}Sz_{-2}$	$Sz_{-2}Sz_{-3}$	$Sz_{-2}Sz_{-4}$	$Sz_{-2}Sz_{-5}$	$Sz_{-2}Sz_{-6}$	$Sz_{-2}Sz_{-7}$	$Sz_{-2}Sz_{-8}$	0	0	0	$0\cdots$
		$\cdots Sz_{-3}Sz_3$	$Sz_{-3}Sz_2$	$Sz_{-3}Sz_1$	$Sz_{-3}Sz_0$	$Sz_{-3}Sz_{-1}$	$Sz_{-3}Sz_{-2}$	$Sz_{-3}Sz_{-3}$	$Sz_{-3}Sz_{-4}$	$Sz_{-3}Sz_{-5}$	$Sz_{-3}Sz_{-6}$	$Sz_{-3}Sz_{-7}$	$Sz_{-3}Sz_{-8}$	0	0	$0\cdots$
		$\cdots Sz_{-4}Sz_4$	$Sz_{-4}Sz_3$	$Sz_{-4}Sz_2$	$Sz_{-4}Sz_1$	$Sz_{-4}Sz_0$	$Sz_{-4}Sz_{-1}$	$Sz_{-4}Sz_{-2}$	$Sz_{-4}Sz_{-3}$	$Sz_{-4}Sz_{-4}$	$Sz_{-4}Sz_{-5}$	$Sz_{-4}Sz_{-6}$	$Sz_{-4}Sz_{-7}$	$Sz_{-4}Sz_{-8}$	0	$0\cdots$
		$\cdots Sz_{-5}Sz_5$	$Sz_{-5}Sz_4$	$Sz_{-5}Sz_3$	$Sz_{-5}Sz_2$	$Sz_{-5}Sz_1$	$Sz_{-5}Sz_0$	$Sz_{-5}Sz_{-1}$	$Sz_{-5}Sz_{-2}$	$Sz_{-5}Sz_{-3}$	$Sz_{-5}Sz_{-4}$	$Sz_{-5}Sz_{-5}$	$Sz_{-5}Sz_{-6}$	$Sz_{-5}Sz_{-7}$	$Sz_{-5}Sz_{-8}$	$0\cdots$
		$\cdots Sz_{-6}Sz_6$	$Sz_{-6}Sz_5$	$Sz_{-6}Sz_4$	$Sz_{-6}Sz_3$	$Sz_{-6}Sz_2$	$Sz_{-6}Sz_1$	$Sz_{-6}Sz_0$	$Sz_{-6}Sz_{-1}$	$Sz_{-6}Sz_{-2}$	$Sz_{-6}Sz_{-3}$	$Sz_{-6}Sz_{-4}$	$Sz_{-6}Sz_{-5}$	$Sz_{-6}Sz_{-6}$	$Sz_{-6}Sz_{-7}$	$Sz_{-6}Sz_{-8}\cdots$
		$\cdots Sz_{-7}Sz_7$	$Sz_{-7}Sz_6$	$Sz_{-7}Sz_5$	$Sz_{-7}Sz_4$	$Sz_{-7}Sz_3$	$Sz_{-7}Sz_2$	$Sz_{-7}Sz_1$	$Sz_{-7}Sz_0$	$Sz_{-7}Sz_{-1}$	$Sz_{-7}Sz_{-2}$	$Sz_{-7}Sz_{-3}$	$Sz_{-7}Sz_{-4}$	$Sz_{-7}Sz_{-5}$	$Sz_{-7}Sz_{-6}$	$Sz_{-7}Sz_{-7}\cdots$
			$Sz_{-8}Sz_7$	$Sz_{-8}Sz_6$	$Sz_{-8}Sz_5$	$Sz_{-8}Sz_4$	$Sz_{-8}Sz_3$	$Sz_{-8}Sz_2$	$Sz_{-8}Sz_1$	$Sz_{-8}Sz_0$	$Sz_{-8}Sz_{-1}$	$Sz_{-8}Sz_{-2}$	$Sz_{-8}Sz_{-3}$	$Sz_{-8}Sz_{-4}$	$Sz_{-8}Sz_{-5}$	$Sz_{-8}Sz_{-6}\cdots$
		$\cdots x_0-1$	9	9	9	9	9	9	9	9	9	9	9	9	9	$9\ \cdots$
														↓ $w_x/2+Lw=-12$		↓ $w_x/2+Lw-2=-14$

Figure 5. The Squarer of the Intermediate Result with Constraint of Series of Zeros on the Least Digits

the carry from the column of index formula $W_x/2+Lw-1$, and the carry from the column of index formula $W_x/2+Lw-2$. The carry from the column of index formula $W_x/2+Lw-1$ to the column of index formula $w_x/2+Lw$, is the products sum of the column formula $W_x/2+Lw-1$ divided by 10. The carry from the column of index formula $W_x/2+Lw-2$ to the column of index formula $W_x/2+Lw$, is the products sum of the column formula $W_x/2+Lw-2$ divided by 100.

$$cr_{w_x/2+Lw} = \sum_{j=Fw+1}^{w_x/2-1-Fw+Lw} Sz_j * Sz_{w_x/2+Lw-j} - 9 + \frac{\sum_{j=Fw+1}^{w_x/2-2-Fw+Lw} Sz_j * Sz_{w_x/2-1+Lw-j}}{10} + \frac{\sum_{j=Fw+1}^{w_x/2-3-Fw+Lw} Sz_j * Sz_{w_x/2-2+Lw-j}}{100}, \tag{5.17}$$

Note that the column of index $W_x/2+Lw-1$ has two unknown products $2*Sz_{w_x/2}*Sz_{Lw-1}$, and the column of index $W_x/2+Lw-2$ has four unknown products $2*Sz_{w_x/2}*Sz_{Lw-2}$, $2*Sz_{w_x/2-1}*Sz_{Lw-1}$. The engine assumes the sum value of these unknown products $(2*z_{w_x/2}*z_{Lw-1})/10+(2*z_{w_x/2}*z_{Lw-2}+2*z_{w_x/2-1}*z_{Lw-1})/100$, to be equal to $(10-(cr_{w_x/2+Lw}) mod_{10})$, and replaces $cr_{w_x/2+Lw}$ with $cr_{w_x/2+Lw}+(10-(cr_{w_x/2+Lw}) mod_{10})$, in case of a series of zeros, such that $(cr_{w_x/2+Lw}) mod_{10}=0.$

In case of a series of nines, the algorithm solves it in the same way like the series of zeros by adding one to the weight of the last nine in the series of nines of the intermediate result significand mask, and replaces formula $cr_{w_x/2+Lw}$ with formula $cr_{w_x/2+Lw}-(cr_{w_x/2+Lw}) mod_{10}$, such that formula $(cr_{w_x/2+Lw}) mod_{10}=0.$

Then, the algorithm iterates on the iteration indexes formula $Lw+1 \le i \le Fw+1$ to get in each iteration the value of a new digit formula $Sz_{w_x/2-1-Fw+i}$, and estimates the digits formula $Sz_{w_x/2-Fw+i}$, $Sz_{w_x/2-Fw+i+1}$ which may be refined in next iterations. Then, it does another number of iterations from formula $Fw+2 \le i \le -1-w_x/2$ to check that the previous chosen digits value of Sz will make formula $Sx_{w_x/2+i}=9$ for all $Fw+2 \le i \le -1-w_x/2$.

Each iteration on formula $Lw+1 \le i \le Fw+1$, it searches randomly on the values of formula $Sz_{w_x/2-Fw+i-1}$, $Sz_{w_x/2-Fw+i}$, and $Sz_{w_x/2-Fw+i+1}$. It calculates the carry generated from the columns of index formula $w_x/2+i$, $w_x/2+i+1$, $w_x/2+i+2$, using Equation 5.18, Equation 5.19 and Equation 5.20, and checks that the carries satisfy the conditions $(cr_{w_x/2+i}) mod_{10}=0$, $(cr_{w_x/2+i+1}) mod_{10}=0$, and $(cr_{w_x/2+i+2}) mod_{10}=0$.

$$cr_{w_x/2+i} = cr_{w_x/2+i-1}/10 + \sum_{j=Fw+1}^{w_x/2-1-Fw+i} Sz_j * Sz_{w_x/2+i-j} - 9, \tag{5.18}$$

$$cr_{w_x/2+i+1} = cr_{w_x/2+i}/10 + \sum_{j=Fw+1}^{w_x/2-Fw+i} Sz_j * Sz_{w_x/2+i+1-j} - 9, \quad (5.19)$$

$$cr_{w_x/2+i+2} = cr_{w_x/2+i+1}/10 + \sum_{j=Fw+1}^{w_x/2+1-Fw+i} Sz_j * Sz_{w_x/2+i+2-j} - 9, \quad (5.20)$$

The algorithm repeats all the iterations, if the check in any iteration is not achieved. As in the first, the algorithm chooses randomly the digits in the column of index formula $W_x/2+Lw$, and the nonlinear equations in the next iterations depend on this values. This combination of these digits may fail to satisfy the conditions in the next iteration.

In the iterations of $Lw+1 \le i \le Fw+1$, the algorithm gets digits of Sz from $Sz_{w_x/2+Lw-Fw}$ to $Sz_{w_x/2}$. The algorithm does other iterations on $Fw+2 \le i \le -1-w_x/2$ to calculate in each iteration the carry generated from the column of index $w_x/2+i$, using Equation 5.21, and checks that $(cr_{w_x/2+i}) mod_{10} = 0$. This check may make the algorithm fail to get any solution as the number of these iterations increase. As the algorithm has chosen all digits of Sz in the previous iterations without taking in its considerations the nonlinear equations in the iterations of $Fw+2 \le i \le -1-w_x/2$. In this case the engine refines the constraints to get the best solution.

$$cr_{w_x/2+i} = cr_{w_x/2+i-1}/10 + \sum_{j=i}^{w_x/2} Sz_j * Sz_{w_x/2+i-j} - 9, \quad (5.21)$$

After getting the needed digits of Sz, the least digits algorithm squares Sz to get Sx. Then it uses the most digits algorithm to get all digits of Sz using the digits of Sx.

5.2 Decimal Square Root Rounding Boundaries

We use the engine also to get the hardest-to-round cases and determine the number of digits needed to do the correct rounding according to the standard. The problem termed as “table-maker's-dilemma”[11] appears when the result is inexact and the intermediate result has a series of zeros after p digits, or after $p+1$ digits. At this case we do not know the value of the sticky bit, therefore we cannot do correct rounding.

We use the engine to find the largest number of zeros that follow p digits. We find that the largest number of zeros at $p>6$ is $p-2$. The engine generates cases at $p=16$ with 14 zeros, and at $p=34$ with 32 zeros. Two examples from these cases are : (1) at $p=16$, when the input exponent is even and $Sx=6693849239557175$, the result is $Sz=8181594734253937000000000000001894$, (2) at $p=34$, when the input exponent is even and $Sx=3011112066528974958465370408325306$, the intermediate result is $Sz=548736008161390385575435195676408900000000000000000000000000000000007198$.

$$
\begin{array}{lcccccc}
* \; Sz_7 \cdots Sz_{-8} 0 \cdots 0 & 0 & 0 & 0 & 0 & 0 \cdots 0 \; Sz_{-24} \cdots & \\
\;\; Sz_7 \cdots Sz_{-8} 0 \cdots 0 & 0 & 0 & 0 & 0 & 0 \cdots 0 \; Sz_{-24} \cdots & \\
\hline
\cdots 0 & 0 & 0 & 0 & 0 & 0 \cdots & \\
\cdots 0 & 0 & 0 & 0 & 0 & 0 \cdots & \\
\cdots 0 & 0 & 0 & 0 & 0 & 0 \cdots & \\
\cdots 0 & 0 & 0 & 0 & 0 & 0 \cdots & \\
\cdots 0 & 0 & 0 & 0 & 0 & 0 \cdots & \\
\cdots 0 & 0 & 0 & 0 & 0 & 0 \cdots & \\
\cdots 0 & 0 & 0 & 0 & 0 & 0 \cdots & \\
\cdots 0 & 0 & 0 & 0 & 0 & 0 \cdots & \\
\cdots 0 & 0 & 0 & 0 & 0 & 0 \cdots & \\
\cdots 0 & 0 & 0 & 0 & 0 & 0 \cdots & \\
\cdots Sz_{-3} Sz_{-8} & 0 & 0 & 0 & 0 & 0 \cdots & \\
\cdots Sz_{-4} Sz_{-7} & Sz_{-4} Sz_{-8} & 0 & 0 & 0 & 0 \cdots & \\
\cdots Sz_{-5} Sz_{-6} & Sz_{-5} Sz_{-7} & Sz_{-5} Sz_{-8} & 0 & 0 & 0 \cdots & \\
\cdots Sz_{-6} Sz_{-5} & Sz_{-6} Sz_{-6} & Sz_{-6} Sz_{-7} & Sz_{-6} Sz_{-8} & 0 & 0 \cdots & \\
\cdots Sz_{-7} Sz_{-4} & Sz_{-7} Sz_{-5} & Sz_{-7} Sz_{-6} & Sz_{-7} Sz_{-7} & Sz_{-7} Sz_{-8} & 0 \cdots & \\
\cdots Sz_{-8} Sz_{-3} & Sz_{-8} Sz_{-4} & Sz_{-8} Sz_{-5} & Sz_{-8} Sz_{-6} & Sz_{-8} Sz_{-7} & Sz_{-8} Sz_{-8} \cdots & \\
\hline
\cdots \; 9 & 9 & 9 & 9 & 9 & 9 \; \cdots & \\
 & & & & & \uparrow & \\
 & & & & & w_x - 2p &
\end{array}
$$

Figure 6. The squarer of the intermediate result with a series of zeros equals $p-1$.

Lemma 1: In the decimal square root operation, number of trailing zeros after p digits in the intermediate result significand Sz that might be followed by a non-zero digit cannot be more than or equal to $p-1$, for all $p>6$.

Proof: Let us assume that $p-1$ zeros or more exist that followed by a non zero digit, and $p>6$, as shown in Figure 6. The figure shows that the sum of the elements must equal to the formula $d_7 d_6 999999$, where $0 \le d_i < 9$. The sum of the elements can be represented using Equation 5.22.

$$
\begin{aligned}
ElementsSum = cr + (Sz_{w_x/2-p} * Sz_{w_x/2-p}) * 10^0 + (2 * Sz_{w_x/2-p} * Sz_{w_x/2-p+1}) * 10^1 + \\
(2 * Sz_{w_x/2-p} * Sz_{w_x/2-p+2} + Sz_{w_x/2-p+1} * Sz_{w_x/2-p+1}) * 10^2 + \\
(2 * Sz_{w_x/2-p} * Sz_{w_x/2-p+3} + 2 * Sz_{w_x/2-p+1} * Sz_{w_x/2-p+2}) * 10^3 + \\
(2 * Sz_{w_x/2-p} * Sz_{w_x/2-p+4} + 2 * Sz_{w_x/2-p+1} * Sz_{w_x/2-p+3} + Sz_{w_x/2-p+2} * Sz_{w_x/2-p+2}) * 10^4 + \\
(2 * Sz_{w_x/2-p} * Sz_{w_x/2-p+5} + 2 * Sz_{w_x/2-p+1} * Sz_{w_x/2-p+4} + 2 * Sz_{w_x/2-p+2} * Sz_{w_x/2-p+3}) * 10^5 .
\end{aligned}
\tag{5.22}
$$

Where $0 \le cr \le 2*9*9/10 + 4*9*9/100 + 1$ is the carry that propagates from the columns of next lower weights to the digit of weight 10^{w_x-2p}, and each of the six digits $z_{w_x/2-p}, z_{w_x/2-p+1}, z_{w_x/2-p+2}, z_{w_x/2-p+3}, z_{w_x/2-p+4}, z_{w_x/2-p+5}$ has an interval $[0,9]$.

Note that, for $p \leq 6$, Equation 5.22 is not exit, which means that number of trailing zeros may be more than $p-2$, however number of trailing zeros will not be more than p zeros.

The condition that the sum of the elements is equal to formula $d_7 d_6 999999$, can be represented as the formula $(ElementsSum - 999999) mod_{1000000} = 0$.

An exhaustive search for all the values of $cr, z_{w_x/2-p}, z_{w_x/2-p+1}, z_{w_x/2-p+2}, z_{w_x/2-p+3}, z_{w_x/2-p+4}, z_{w_x/2-p+5}$, indicates that the condition $(ElementsSum - 999999) mod_{1000000} = 0$ cannot be achieved. Hence the assumption of $p-1$ zeros or more is invalid and the lemma is proven.

Theorem 1: Only $2\text{p}-1$ digits not including leading zeros are sufficient to do the correct rounding to Decimal Floating-Point Square Root operation, at $p > 6$.

Proof: Based on the previous lemma, no more than $p-1$ digits are needed after the rounding position to ensure the correct calculation of the sticky bit. Hence the total number of digits is $p + p - 1 = 2\text{p} - 1$.

5.3 The Main Ideas of the Square Root Models

The models are defined using a Cartesian product between two or more lists of constraints with ignoring the impossible combinations, and allowing the other constraints to be chosen randomly.

All the model proposal ideas are in [22], except the ideas of the nines and zeros model. However we describe all the ideas in the form of our engine constraints.

A) Inputs Types Model

The model aims to verify the ability to solve all possible combinations of the input types. The proposal ideas of the model are in [22]. We separate the model into three sub-models as follows:

1. It verifies the Zero input using, (1) a list of the input exponent from the

interval $[qmin, qmax]$, (2) the input significand is equal to zero (3) a list from the two types of the input sign.

2. It verifies the design when the input is Infinity, sNaN, or qNaN using, (1) a list of input from the Infinities, sNaN, and qNaN, (2) a list from the two types of the input sign.

3. It verifies the design in solving the other input types using, (1) a list of the input from the minimum Subnormal, the maximum Subnormal, the minimum Normal , and the maximum Normal, (2) a list from the two types of the input sign.

B) Result Types Model

The model aims to verify the generation of the different types of the final result. The proposal ideas of the model are in [22]. We separate the model into four sub-models as follows:

1. It verifies all the result exponents using, (1) a list of the input exponents from the interval $[qmin, qmax]$.

2. It verifies the generation of the first hundred numbers and the last hundred subnormal numbers, and the first hundred normal numbers using, (1) the input exponent is equal to $qmin$, (2) a list of the intermediate result significand that consists of the intervals $\{[2{,}100],[10^{p-1}-100{,}10^{p-1}+100]\}$.

3. It verifies the generation of numbers from One to 100 using, (1) the input exponent is equal zero, (2) a list of the intermediate result significand from the interval $[1{,}100]$.

4. It verifies the last hundred Normal numbers using, (1) the input exponent is equal to $qmax$, (2) a list of the intermediate result significands from the interval $[10^{p}-100{,}10^{p}-1]$.

C) Rounding Model

The model aims to verify the rounding process in the design. The proposal ideas of the model are in [22]. We separate the model into three sub-models as follows:

1. It verifies the rounding process at the all combinations from the guard digit, the least significand digit, and the sticky bit using, (1) a list from the five rounding modes, (2) a list of the intermediate result significand consists of the cross products of the guard digit interval $[0,9]$, the least significand digit interval $[0,9]$.

2. It verifies the possible carry propagation due to rounding process using, (1) a list from the five rounding modes, (2) a list of the intermediate result significand consists of the guard digit interval $[0,9]$, and the patterns $\{\overbrace{99\cdots 9}^{p}, \overbrace{\{0-8\}9\cdots 9}^{p}, \overbrace{X\{0-8\}9\cdots 9}^{p}, \cdots, \overbrace{XX\cdots X\{0-8\}}^{p}\}$.

3. It verifies the sticky bit calculations using, (1) a list of the intermediate result significand that consists of the patterns

$$\begin{gathered}
\overbrace{\{1-9\}x\cdots x}^{P}0x\cdots x, \overbrace{\{1-9\}x\cdots x}^{P}00x\cdots x, \cdots, \overbrace{\{1-9\}x\cdots x}^{P}\overbrace{00\cdots 00}^{p_x-2}x\cdots x \\
0\overbrace{\{1-9\}x\cdots x}^{P}0x\cdots x, 0\overbrace{\{1-9\}x\cdots x}^{P}00x\cdots x, \cdots, 0\overbrace{\{1-9\}x\cdots x}^{P}\overbrace{00\cdots 00}^{p_x-2}x\cdots x \\
00\overbrace{\{1-9\}x\cdots x}^{P}0x\cdots x, 00\overbrace{\{1-9\}x\cdots x}^{P}00x\cdots x, \cdots, 00\overbrace{\{1-9\}x\cdots x}^{P}\overbrace{00\cdots 00}^{p_x-2}x\cdots x \\
\vdots \\
\overbrace{0\cdots 0}^{p/2}\overbrace{\{1-9\}x\cdots x}^{P}0x\cdots x, \overbrace{0\cdots 0}^{p/2}\overbrace{\{1-9\}x\cdots x}^{P}00x\cdots x, \cdots, \overbrace{0\cdots 0}^{p/2}\overbrace{\{1-9\}x\cdots x}^{P}\overbrace{00\cdots 00}^{p_x-2}x\cdots x
\end{gathered}$$

D)Trailing and Leading Zeros Model

The model aims to verify all the possible trailing and leading zeros in the input significand and the intermediate result significand. The proposal ideas of the model are also in [22]. We separate the model into two sub-models as follows:

1. It verifies the possible trailing and leading zeros the input significand using, (1) a list of the first input significand that consists of the patterns

$$\begin{gathered}
\overbrace{\{1-9\}00\cdots 00}^{P}, \overbrace{0\{1-9\}00\cdots 00}^{P}, \cdots, \overbrace{00\cdots 0\{1-9\}}^{P} \\
\overbrace{\{1-9\}\{1-9\}0\cdots 00}^{P}, \overbrace{0\{1-9\}\{1-9\}0\cdots 00}^{P}, \cdots, \overbrace{00\cdots 0\{1-9\}\{1-9\}}^{P} \\
\overbrace{\{1-9\}X\{1-9\}0\cdots 00}^{P}, \overbrace{0\{1-9\}X\{1-9\}0\cdots 00}^{P}, \cdots, \overbrace{00\cdots 0\{1-9\}X\{1-9\}}^{P} \\
\vdots \\
\overbrace{\{1-9\}XX\cdots X\{1-9\}}^{P}
\end{gathered}$$

2.A list of the intermediate result sigificand, to verify the generation of the trailing and leading zeros in the intermediate result significand, it consists of

$$\overbrace{\{1-9\}00\cdots00}^{p},\overbrace{0\{1-9\}00\cdots00}^{p},\cdots,\overbrace{00\cdots0\{1-9\}}^{p},$$
$$\overbrace{\{1-9\}\{1-9\}0\cdots00}^{p},\overbrace{0\{1-9\}\{1-9\}0\cdots00}^{p},\cdots,\overbrace{00\cdots0\{1-9\}\{1-9\}}^{p}$$
$$\overbrace{\{1-9\}X\{1-9\}0\cdots00}^{p},\overbrace{0\{1-9\}X\{1-9\}0\cdots00}^{p},\cdots,\overbrace{00\cdots0\{1-9\}X\{1-9\}}^{p}$$
$$\vdots$$
$$\overbrace{XX\cdots X\{1-9\}}^{p}$$

E) Zeros and Nines Model

The model aims to verify all the possible patterns of zeros and nines in the input significands and the intermediate result significand. The proposal ideas of the model are all new. We separate the model into four sub-models as follows:

1. It verifies the patterns of zeros in the intermediate result significand using, (1) a list of the intermediate result significand that consists of the patterns

$$\overbrace{\{1-9\}00\cdots0X}^{2p-1},\overbrace{\{1-9\}00\cdots0XX}^{2p-1},\cdots,\overbrace{\{1-9\}X\cdots XX}^{2p-1}$$
$$\overbrace{X\{1-9\}0\cdots0X}^{2p-1},\overbrace{X\{1-9\}0\cdots0XX}^{2p-1},\cdots,\overbrace{X\{1-9\}X\cdots XX}^{2p-1}$$
$$\overbrace{XX\{1-9\}0\cdots0X}^{2p-1},\overbrace{XX\{1-9\}0\cdots0XX}^{2p-1},\cdots,\overbrace{XX\{1-9\}X\cdots XX}^{2p-1}$$
$$\vdots$$
$$\overbrace{XXX\cdots X\{1-9\}}^{2p-1}$$

2. It verifies the patterns of nines in the intermediate result significand using , (1) a list of the intermediate result significand that consists of the patterns

$$\overbrace{\{1-9\}99\cdots99}^{2p-1},\overbrace{\{1-9\}99\cdots99X}^{2p-1},\overbrace{\{1-9\}99\cdots9XX}^{2p-1},\cdots,\overbrace{\{1-9\}X\cdots XX}^{2p-1}$$
$$\overbrace{X\{1-9\}99\cdots99}^{2p-1},\overbrace{X\{1-9\}99\cdots99X}^{2p-1},\overbrace{X\{1-9\}99\cdots9XX}^{2p-1},\cdots,\overbrace{X\{1-9\}X\cdots XX}^{2p-1}$$
$$\overbrace{XX\{1-9\}99\cdots99}^{2p-1},\overbrace{\{1-9\}99\cdots99X}^{2p-1},\overbrace{XX\{1-9\}99\cdots9XX}^{2p-1},\cdots,\overbrace{XX\{1-9\}X\cdots XX}^{2p-1}$$
$$\vdots$$
$$\overbrace{XXX\cdots X\{1-9\}}^{2p-1}$$

3. It verifies all patterns of zeros in the input significand using, (1) a list the first input significand that consists of the patterns

$$
\begin{gathered}
\overbrace{\{1-9\}00\cdots0X}^{p}, \overbrace{\{1-9\}00\cdots0\mathrm{XX}}^{p}, \cdots, \overbrace{\{1-9\}X\cdots XX}^{p} \\
\overbrace{X\{1-9\}0\cdots0X}^{p}, \overbrace{X\{1-9\}0\cdots0\mathrm{XX}}^{p}, \cdots, \overbrace{X\{1-9\}X\cdots XX}^{2p} \\
\overbrace{X\,X\{1-9\}0\cdots0X}^{2p}, \overbrace{X\,X\{1-9\}0\cdots0\mathrm{XX}}^{p}, \cdots, \overbrace{X\,X\{1-9\}X\cdots XX}^{p} \\
\vdots \\
\overbrace{XXX\cdots X\{1-9\}}^{p}
\end{gathered}
$$

4. It verifies all patterns of nines in the input significands using, (1) a list the first input significand that consists of the patterns

$$
\begin{gathered}
\overbrace{\{1-9\}99\cdots99}^{p}, \overbrace{\{1-9\}99\cdots99\mathrm{X}}^{p}, \overbrace{\{1-9\}99\cdots9\mathrm{XX}}^{p}, \cdots, \overbrace{\{1-9\}X\cdots XX}^{p} \\
\overbrace{X\{1-9\}99\cdots99}^{p}, \overbrace{X\{1-9\}99\cdots99\mathrm{X}}^{p}, \overbrace{X\{1-9\}99\cdots9\mathrm{XX}}^{p}, \cdots, \overbrace{X\{1-9\}X\cdots XX}^{p} \\
\overbrace{XX\{1-9\}99\cdots99}^{p}, \overbrace{\{1-9\}99\cdots99\mathrm{X}}^{p}, \overbrace{XX\{1-9\}99\cdots9\mathrm{XX}}^{p}, \cdots, \overbrace{XX\{1-9\}X\cdots XX}^{p} \\
\vdots \\
\overbrace{XXX\cdots X\{1-9\}}^{p}
\end{gathered}
$$

5.4 Summary

This chapter represents the main steps the first square root engine to solve all the constraints numerically. It also describes the main ideas of the coverage models that have been solved by the engine to generate test vectors can verify all the corner cases in the hardware or software implementations of the decimal floating-point square root operation.

The chapter also describes the rounding boundaries of the decimal Square root operation, which our engine and our models are based on. Therefore, it gives an advantage to the square root engine and the square root models.

The engine solved the coverage models one time and generated about 50000 test vectors in Decimal64 and about 199000 test vectors in Decimal128, the test vectors have proved an efficiency by discovering bugs in DecNumber library and Silminds design. Most of the bugs in the DecNumber library or Silminds design are discovered using the rounding model and the zeros and nines model.

Chapter 6

Engine and Models of Decimal Division Operation

The division engine generates test vectors, to cover corner cases, to verify a tested implementation of decimal division operation to achieve the compliance with the IEEE standard (754-2008) for Floating Point Arithmetic.

The engine is a software tool written in C++ to solve all the coverage models. Although the engine solves constraints on the inputs and the unbounded intermediate result only, it managed to discover some faults inside the operation by forcing the engine to solve constraints on patterns of zeros and nines in the intermediate result significand.

We design the engine to solve decimal division constraints on the unbounded intermediate result that consists of $2.5p$ digits and on simultaneous constraints of inputs and the unbounded intermediate result. Similar engines have been developed in [8], but they either solve constraints on the intermediate result which consist of $p+1$ digits and sticky bit, or solve simultaneous constraints of the inputs and the output. The engines in [8] do not solve simultaneous constraints on the inputs and the unbounded intermediate result. This means that our engine has the ability to generate test vectors to discover corner cases in the decimal division implementations that cannot be generated by the engines in [8].

We also design coverage models based on the chosen constraints of the division operation. The engine solves the coverage models to generate test vectors that verify the corner cases of the division in different implementations.

The engine generates the test vectors in two formats of the IEEE standard: Decimal64 and Decimal128. The engine time to generate one test vector depends on the constraints that have been solved to generate it and the factor of randomization that the engine needed. The engine generates as many test vectors as the user wants. Every time the engine runs, it generates new test vectors. The verification engine value is neither in the time needed to generate the test vector, if this time is practical, nor in the number of the generated test vectors, but rather in the functionality of the cases that the test vector covers.

The engine solved the coverage models one time and generated about 339000 test vectors in Decimal128 and about 146000 in Decimal64, the test vectors have proved their efficiency by discovering bugs in Silminds design [7]. Table 3 shows the maximum and the minimum times that the engine needed to solve a task of the existing constraints and generate one test vector, on Intel(R) Pentium(R) 4 CPU 3.20GHZ with g++ (Ubuntu 4.4.3) compiler.

TABLE 3. THE TIME PERFORMANCE OF THE DIVISION ENGINE

Test vector Format	**Minimum Time**	**Maximum Time**
Decimal 64	0.01 seconds	7 seconds
Decimal 128	0.03 seconds	2 minutes

The generated test vector is a decimal vector that has four sets, The first set is the operation type division, number of the precision (64 or 128), and the rounding mode. The second set is sign, significand, and exponent of the first input. The third set is sign, significand, and exponent of the second input. The fourth set is sign, significand, and exponent of the result. Finally the fifth set is one or two from five flags(invalid, inexact, underflow, overflow, division by zero). The designer enters the input sets to his implementation and verifies the implementation output against last two sets.

The task given to the division engine is the set of constraints on five elements, the significand of the first input (dividend) Sx, the significand of the second input (divisor) Sy, the intermediate result Sz, the exponent of the first input, and the rounding mode. The constraint on Sx is a mask starting from the minimum number Nx to the maximum number Mx. The constraint on Sy is a mask starting from the minimum number Ny to the maximum number My. Similarly, the mask on Sz consists of two numbers Nz and Mz. The first input exponent, the intermediate result exponent and the rounding direction are either given explicitly in the task or left to the engine to choose randomly.

An example to explain the format of the decimal division task at $p=16$ is as follows:

$$64/T:\ +1\ \ +9999999999999999\ \ \ +1\ \ +9999999999999999$$
$$+1000000000000002\text{p}4000000000000000000000000$$
$$+9999999999999992\text{p}4000000000000000000000000$$
$$R\ \ R\ \ 2$$

This task means that $Nx=+1,\ Mx=+9999999999999999,$ $Ny=+1,\ My=+9999999999999999,$ $Nz=+1000000000000002\text{p}4000000000000000000000000,$ $Mz=+9999999999999992\text{p}4000000000000000000000000,$ the engine chooses randomly the exponent of the first input, and the intermediate result exponent, while the rounding mode is Round to Zero.

One of the solutions of this task is the test vector d64/ 0 +961708551261171E70 +937500E-103 -> +1025822454678582E167 X. The *d64* means decimal64, the / means the division operation, the following *0* means that the rounding mode is Round toward Zero, the input is $x=+961708551261171*10^{70}$, $y=+937500*10^{-103}$, the rounded result is $z=+1025822454678582*10^{167}$, and the following X indicates that the inexact flag is high, because the exact result is $+1025822454678582.40000000000\cdots*10^{167}$.

We represent the intermediate result with length $2.5p$ digits not including the leading zeros to guarantee that the engine can generate all the possible hardest-to-round cases. The results show that this length is enough to put constraints on the rounding boundaries, where the hardest-to round case needs only $2\text{p}+1$ digits not including leading zeros to do the rounding process according to the standard.

6.1 The Division Engine

The inverse operation of the division $z=x/y$ is the multiplication of the intermediate result with the divisor which gives the dividend of the division operation. The engine is based on solving the non linear equations that result from multiplying the intermediate result with the divisor. We can estimate these non linear equations from Figure 7, where each column represents one nonlinear equation. The figure shows the multiplication of the intermediate result with the divisor at $p=16$, where Sz_i denotes the intermediate result digit of weight 10^i, Sx_i denotes the first input (dividend) digit of weight 10^i, and Sy_i denotes the second input (divisor) digit of weight 10^i.

The engine solves the signifiand in the normalized form, it solves the inputs significands in the form of $Sx_0.Sx_{-1}\cdots Sx_{-p+2}Sx_{-p+1}$ and $Sy_0.Sy_{-1}\cdots Sy_{-p+2}Sy_{-p+1}$, and generates the intermediate result significand in the form $Sz_0.Sz_{-1}\cdots Sz_{-p+2}Sz_{-p+1}\cdots$. Such that the inputs most significand digits $Sx_0\neq 0 \wedge Sy_0\neq 0$, however the intermediate result most significand digit Sz_0 may equal to zero or may not. The normalized form guarantees that the intermediate result significand has fixed form, and we can easily estimate the nonlinear equation shown in Figure 7 using the normalized form.

The engine uses $2.5p$ digits only for the intermediate result significand Sz. Hence, if the infinitely precise division Sx/Sy has more digits, then Sz is truncated, i.e. it is slightly less than the infinitely precise division. The multiplication of $Sz*Sy$ will thus be $Sx-\Delta$ with $0<\Delta\leq 10^{-L}$ where L depends on the number of digits of Sz. This explains the series of nines that follows $Sx_{-p+1}-1$ as seen in Figure 7.

The engine steps begin by normalizing the mask of the input significands, it shifts the mask $\{Nx, Mx\}$ to the right with the value srx and the mask $\{Ny, My\}$ to right with the value sry.

Then, the engine gets the intermediate result significand Sz and the inputs significand Sx and Sy that achieve the constraints. It achieves the constraint on each digit Sx_n, Sy_n, or Sz_n by choosing the digit from its interval $[Nx_n, Mx_n]$, interval $[Ny_n, My_n]$, or interval $[Nz_n, Mz_n]$. It solves the significands constraints using one of two algorithms, the first algorithm is the Division-Most-Digits-Constraints-Algorithm to solve the constraints on the most significant p digits of the intermediate result significand and the p digits of the inputs significand.

The second algorithm is the Division-Least-Digits Constraints-Algorithm to solve the constraints on the least significant digits that follow the highest p digits of the intermediate result significand and the p digits of the divisor significand.

The engine also chooses the first input exponent Ex either from the interval $[qmin, qmax]$, or it is given explicitly.

$$
\begin{array}{ccccccccccccccccc}
{*}\,Sy_0 & Sy_{-1} & Sy_{-2} & Sy_{-3} & Sy_{-4} & Sy_{-5} & Sy_{-6} & Sy_{-7} & Sy_{-8} & Sy_{-9} & Sy_{-10} & Sy_{-11} & Sy_{-12} & Sy_{-13} & Sy_{-14} & Sy_{-15} & \\
Sz_0 & Sz_{-1} & Sz_{-2} & Sz_{-3} & Sz_{-4} & Sz_{-5} & Sz_{-6} & Sz_{-7} & Sz_{-8} & Sz_{-9} & Sz_{-10} & Sz_{-11} & Sz_{-12} & Sz_{-13} & Sz_{-14} & Sz_{-15} & Sz_{-16}\cdots \\
\hline
Sz_0Sy_0 & Sz_0Sy_{-1} & Sz_0Sy_{-2} & Sz_0Sy_{-3} & Sz_0Sy_{-4} & Sz_0Sy_{-5} & Sz_0Sy_{-6} & Sz_0Sy_{-7} & Sz_0Sy_{-8} & Sz_0Sy_{-9} & Sz_0Sy_{-10} & Sz_0Sy_{-11} & Sz_0Sy_{-12} & Sz_0Sy_{-13} & Sz_0Sy_{-14} & Sz_0Sy_{-15} & \\
 & Sz_{-1}Sy_0 & Sz_{-1}Sy_{-1} & Sz_{-1}Sy_{-2} & Sz_{-1}Sy_{-3} & Sz_{-1}Sy_{-4} & Sz_{-1}Sy_{-5} & Sz_{-1}Sy_{-6} & Sz_{-1}Sy_{-7} & Sz_{-1}Sy_{-8} & Sz_{-1}Sy_{-9} & Sz_{-1}Sy_{-10} & Sz_{-1}Sy_{-11} & Sz_{-1}Sy_{-12} & Sz_{-1}Sy_{-13} & Sz_{-1}Sy_{-14} & Sz_{-1}Sy_{-15}\cdots \\
 & & Sz_{-2}Sy_0 & Sz_{-2}Sy_{-1} & Sz_{-2}Sy_{-2} & Sz_{-2}Sy_{-3} & Sz_{-2}Sy_{-4} & Sz_{-2}Sy_{-5} & Sz_{-2}Sy_{-6} & Sz_{-2}Sy_{-7} & Sz_{-2}Sy_{-8} & Sz_{-2}Sy_{-9} & Sz_{-2}Sy_{-10} & Sz_{-2}Sy_{-11} & Sz_{-2}Sy_{-12} & Sz_{-2}Sy_{-13} & Sz_{-2}Sy_{-14}\cdots \\
 & & & Sz_{-3}Sy_0 & Sz_{-3}Sy_{-1} & Sz_{-3}Sy_{-2} & Sz_{-3}Sy_{-3} & Sz_{-3}Sy_{-4} & Sz_{-3}Sy_{-5} & Sz_{-3}Sy_{-6} & Sz_{-3}Sy_{-7} & Sz_{-3}Sy_{-8} & Sz_{-3}Sy_{-9} & Sz_{-3}Sy_{-10} & Sz_{-3}Sy_{-11} & Sz_{-3}Sy_{-12} & Sz_{-3}Sy_{-13}\cdots \\
 & & & & Sz_{-4}Sy_0 & Sz_{-4}Sy_{-1} & Sz_{-4}Sy_{-2} & Sz_{-4}Sy_{-3} & Sz_{-4}Sy_{-4} & Sz_{-4}Sy_{-5} & Sz_{-4}Sy_{-6} & Sz_{-4}Sy_{-7} & Sz_{-4}Sy_{-8} & Sz_{-4}Sy_{-9} & Sz_{-4}Sy_{-10} & Sz_{-4}Sy_{-11} & Sz_{-4}Sy_{-12}\cdots \\
 & & & & & Sz_{-5}Sy_0 & Sz_{-5}Sy_{-1} & Sz_{-5}Sy_{-2} & Sz_{-5}Sy_{-3} & Sz_{-5}Sy_{-4} & Sz_{-5}Sy_{-5} & Sz_{-5}Sy_{-6} & Sz_{-5}Sy_{-7} & Sz_{-5}Sy_{-8} & Sz_{-5}Sy_{-9} & Sz_{-5}Sy_{-10} & Sz_{-5}Sy_{-11}\cdots \\
 & & & & & & Sz_{-6}Sy_0 & Sz_{-6}Sy_{-1} & Sz_{-6}Sy_{-2} & Sz_{-6}Sy_{-3} & Sz_{-6}Sy_{-4} & Sz_{-6}Sy_{-5} & Sz_{-6}Sy_{-6} & Sz_{-6}Sy_{-7} & Sz_{-6}Sy_{-8} & Sz_{-6}Sy_{-9} & Sz_{-6}Sy_{-10}\cdots \\
 & & & & & & & Sz_{-7}Sy_0 & Sz_{-7}Sy_{-1} & Sz_{-7}Sy_{-2} & Sz_{-7}Sy_{-3} & Sz_{-7}Sy_{-4} & Sz_{-7}Sy_{-5} & Sz_{-7}Sy_{-6} & Sz_{-7}Sy_{-7} & Sz_{-7}Sy_{-8} & Sz_{-7}Sy_{-9}\cdots \\
 & & & & & & & & Sz_{-8}Sy_0 & Sz_{-8}Sy_{-1} & Sz_{-8}Sy_{-2} & Sz_{-8}Sy_{-3} & Sz_{-8}Sy_{-4} & Sz_{-8}Sy_{-5} & Sz_{-8}Sy_{-6} & Sz_{-8}Sy_{-7} & Sz_{-8}Sy_{-8}\cdots \\
 & & & & & & & & & Sz_{-9}Sy_0 & Sz_{-9}Sy_{-1} & Sz_{-9}Sy_{-2} & Sz_{-9}Sy_{-3} & Sz_{-9}Sy_{-4} & Sz_{-9}Sy_{-5} & Sz_{-9}Sy_{-6} & Sz_{-9}Sy_{-7}\cdots \\
 & & & & & & & & & & Sz_{-10}Sy_0 & Sz_{-10}Sy_{-1} & Sz_{-10}Sy_{-2} & Sz_{-10}Sy_{-3} & Sz_{-10}Sy_{-4} & Sz_{-10}Sy_{-5} & Sz_{-10}Sy_{-6}\cdots \\
 & & & & & & & & & & & Sz_{-11}Sy_0 & Sz_{-11}Sy_{-1} & Sz_{-11}Sy_{-2} & Sz_{-11}Sy_{-3} & Sz_{-11}Sy_{-4} & Sz_{-11}Sy_{-5}\cdots \\
 & & & & & & & & & & & & Sz_{-12}Sy_0 & Sz_{-12}Sy_{-1} & Sz_{-12}Sy_{-2} & Sz_{-12}Sy_{-3} & Sz_{-12}Sy_{-4}\cdots \\
 & & & & & & & & & & & & & Sz_{-13}Sy_0 & Sz_{-13}Sy_{-1} & Sz_{-13}Sy_{-2} & Sz_{-13}Sy_{-3}\cdots \\
 & & & & & & & & & & & & & & Sz_{-14}Sy_0 & Sz_{-14}Sy_{-1} & Sz_{-14}Sy_{-2}\cdots \\
 & & & & & & & & & & & & & & & Sz_{-15}Sy_0 & Sz_{-15}Sy_{-1}\cdots \\
 & & & & & & & & & & & & & & & & Sz_{-16}Sy_0\cdots \\
\hline
Sx_0 & Sx_{-1} & Sx_{-2} & Sx_{-3} & Sx_{-4} & Sx_{-5} & Sx_{-6} & Sx_{-7} & Sx_{-8} & Sx_{-9} & Sx_{-10} & Sx_{-11} & Sx_{-12} & Sx_{-13} & Sx_{-14} & Sx_{-15}-1 & 9\ \cdots
\end{array}
$$

Figure 7. The Multiplication of the Intermediate Result with the Divisor assuming Precision 16

Then, given that $Ez = Ex - Ey$ and $Ex, Ez \in [qmin, qmax]$, the engine chooses the intermediate result exponent according to $max(qmin, Ex - qmax) \leq Ez \leq min(qmax, Ez - qmin)$. However, if Ez is given, it chooses the first input exponent using $max(qmin, Ez + qmin) \leq Ex \leq min(qmax, Ez + qmax)$. Finally, it calculates the second input exponent $Ey = Ex - Ez$.

After getting the significands and exponents of x, y, z, the engine shifts to left the significand Sx with the value srx and the significand Sy with the value sry. The engine replaces the intermediate result exponent Ez with $Ez + srx - sry$. Then, it shifts to left the intermediate result significand Sz with a value according to the standard and subtracts this value from Ez.

6.1.1 The Division Most Digits Constraints Algorithm

The algorithm iterates to solve the nonlinear equations from left to right. As shown in Figure 7, for $p=16,$ the first non linear equation from left is

$$Sx_0-Sz_0*Sy_0=br_0 \tag{6.1}$$

where br_0 is the value of carries that transfer from previous weights to the weight of $10^0,$ or the borrow generated from this weight to lower weights. The second and the third non linear equations are:

$$Sx_{-1}+10*br_0-Sz_0*Sy_{-1}-Sz_{-1}*Sy_0=br_{-1} \tag{6.2}$$

$$Sx_{-2}+10*br_{-1}-Sz_0*Sy_{-2}-Sz_{-1}*Sy_{-1}-Sz_{-2}*Sy_0=br_{-2}. \tag{6.3}$$

In general the nonlinear equation for the column of index n is :

$$br_n=Sx_n+10*br_{n+1}-\sum_{j=n}^{j=0}Sz_j*Sy_{n-j}, \tag{6.4}$$

To start the solution, the algorithm attempts to solve equations 6.1 to 6.3 (representing columns 0 to -2) together based on the range of carries that may transfer from the next lower significant columns. The algorithm chooses the digit Sx_0 and the digit Sx_{-1} randomly from their intervals. Then since the ranges of borrow digit $br_{-2},$ the digit $Sz_{-2},$ and the digit Sy_{-2} are known as $Ncr_{-2}\le br_{-2}\le Mcr_{-2},$ $Nz_{-2}\le Sz_{-2}\le Mz_{-2},$ and $Ny_{-2}\le Sy_{-2}\le My_{-2},$ the algorithm transforms Equation 3 to the inequality condition:

$$Ncr_{-2}+Nz_{-2}*Sy_0+Sz_0*Ny_{-2}\le Sx_{-2}+10*br_{-1}-Sz_{-1}*Sy_{-1}\le Mcr_{-2}+Mz_{-2}*Sy_0+Sz_0*My_{-2}. \tag{6.5}$$

Finally, it searches randomly on the values of $Sz_0,$ $Sz_{-1},$ $Sy_0,$ $Sy_{-1},$ Sx_{-2} that satisfy Equation 6.1, Equation 6.2 and the Inequality 6.5 . The steps taken so far constitute the first outer iteration that gets the final values of $Sz_0,$ $Sy_0,$ $Sx_0,$ $Sx_{-1},$ Sx_{-2} and estimates the values of $Sz_{-1},$ Sy_{-1} that may be refined in the following iteration.

In the second iteration, the algorithm transforms the fourth nonlinear equation $Sx_{-3}+10*br_{-2}-Sz_0*Sy_{-3}-Sz_{-3}*Sy_0-Sz_{-1}*Sy_{-2}-Sz_{-2}*Sy_{-1}=br_{-3}$ to the inequality condition:

$$Nbr_{-3}+Nz_{-3}*Sy_0+Sz_0*Ny_{-3}\le Sx_{-3}+10*br_{-2}-Sz_{-1}*Sy_{-2}-Sz_{-2}*Sy_{-1}\le Mbr_{-3}+Mz_{-3}*Sy_0+Sz_0*My_{-3},$$

it searches randomly on the values of Sz_{-1}, Sz_{-2}, Sy_{-1}, Sy_{-2}, Sx_{-3} that achieve the second nonlinear equation, the third nonlinear equation and the inequality condition, where the digits Sz_0, Sy_0, Sb_0, Sx_0, Sx_{-1}, Sx_{-2} are known from the previous iteration. The algorithm does this procedure in all the iterations and gets all digits of Sx, Sy, and Sz.

In general, for any precision, the algorithm gets randomly the first two digits of Sx, which are Sx_0 and Sx_{-1} from their intervals. If Sz_0 is chosen to be equal to zero, it gets randomly the digit Sx_{-2} and replaces Sx_{-1} with $Sx_{-1}+10*Sx_0$. In this case the engine begins to solve the nonlinear equations from the nonlinear equation of column index $w_z=-1$, where 10^{w_z} is the weight of the most significand digit in the intermediate result significand of Sz.

Then, it loops through a number of outer iterations equal to the number of nonlinear equations(i.e number of columns). The index of the outer iterations goes from $0 \leq i \leq 2.5p-1$. The algorithm gets in iteration i the values of Sz_{w_z-i}, Sy_{-i} and Sx_{w_z-i-2} and estimates the value of Sz_{w_z-i-1}, Sy_{-i-1}. Then, in the next iteration it gets the values of Sz_{w_z-i-1}, Sy_{-i-1} and Sx_{w_z-i-3} and estimates $Sz_{w_x/2-i-1}$, and so on.

The general form of Equation 6.1, at iteration i, is

$$br_{w_z-i}=Sx_{w_z-i}-\sum_{j=-i}^{0} Sz_{w_z+j}*Sy_{-i-j}. \tag{6.6}$$

Equation 6.6 calculates the borrow from the column of index w_z-i. The equation has one unknown br_{w_z-i} (i.e the borrow of the column), while the other elements of the equation are known from the previous iterations and the value Sz_{w_z-i}, Sy_{-i}.

The general form of Equation 6.2, at iteration i, is

$$br_{w_z-i-1}=Sx_{w_z-i-1}+10*br_{w_z-i-1}-\sum_{j=-i-1}^{0} Sz_{w_z+j}*Sy_{-i-j-1}, \tag{6.7}$$

which calculates the borrow from the column of index w_z-i-1. The equation has one unknown br_{w_z-i-1} (i.e the borrow of the column), while the other elements of the equation are known from the previous iterations, the values of Sz_{w_z-i}, Sz_{w_z-i-1}, Sy_{-i}, Sy_{-i-1}, and the value of br_{w_z-i} from Equation 6.6.

Similarly, the general form of Equation 6.3, at iteration i, is

$$br_{w_z-i-2}=Sx_{w_z-i-2}+10*br_{w_z-i-1}-\sum_{j=-i-2}^{0} Sz_{w_z+j}*Sy_{-i-j-2}. \tag{6.8}$$

As the ranges of br_{w_z-i-2}, Sz_{w_z-i-2}, and Sy_{-i-2}, are known, the algorithm transforms Equation 6.8 to inequality 6.9, which is the general form of inequality 6.5.

$$\begin{gathered} Ncr_{w_z-i-3}+Ncr_{w_z-i-4}+Nc_{w_z-i-5}+Sz_{w_z}*Ny_{-i-2}+Nz_{w_z-i-2}*Sy_0\leq \\ Sx_{w_z-i-2}+10*br_{w_z-i-1}-\sum_{j=-i-1}^{-1} Sz_{w_z+j}*Sy_{-i-j-2} \\ \leq Sz_{w_z}*My_{-i-2}+Mz_{w_z-i-2}*Sy_0+Mcr_{w_z-i-3}+Mcr_{w_z-i-4}+Mcr_{w_z-i-5}+1 \end{gathered} \tag{6.9}$$

Within each outer iteration, the engine does a second level of iterations to get the values of Sx_{w_z-i-2}, Sz_{w_z-i}, Sz_{w_z-i-1}, Sy_{-i}, Sy_{-i-1} that achieve at each outer iteration inequality 6.9. At this second level of iterations, the engine just chooses random numbers from the intervals of Sx_{w_z-i-2}, Sz_{w_z-i}, Sz_{w_z-i-1}, Sy_{-i}, Sy_{-i-1}. If these numbers do not satisfy inequality 6.9, it chooses another combination of numbers, and so on until it finds a set of numbers that satisfy this inequality.

The range of br_{w_z-i-2} is the range of the carries that transfer from the columns follow the column w_z-i-2. Since the algorithm solves only $2.5p$ columns, the maximum product sum of any column at $p=34$ is equal to $2.5*34*9*9=6685$. This number means that a carry from any column,at $p\leq 34$, may affect the previous three columns directly by a value more than one and affects the higher columns indirectly by a value less than or equal to one. Based on that, the algorithm determines the range of carries that transfer to the column w_z-i-2 from the next three columns w_z-i-3, w_z-i-4, w_z-i-5.

Equation 6.10 and Equation 6.11 get the maximum and the minimum carries $Mcr_{w_z-i-3}, Ncr_{w_z-i-3}$ from the column of index w_z-i-3 to the column of index w_z-i-2.

$$Mcr_{w_z-i-3}=\frac{\sum_{j=-i-3}^{-i-2} Mz_{w_z+j}*Sy_{-i-j-3}+\sum_{j=-i-1}^{-2} Sz_{w_z+j}*Sy_{-i-j-3}+\sum_{j=-1}^{0} Sz_{w_z+j}*My_{-i-j-3}}{10}, \quad (6.10)$$

$$Ncr_{w_z-i-3}=\frac{\sum_{j=-i-3}^{-i-2} Nz_{w_z+j}*Sy_{-i-j-3}+\sum_{j=-i-1}^{-2} Sz_{w_z+j}*Sy_{-i-j-3}+\sum_{j=-1}^{0} Sz_{w_z+j}*Ny_{-i-j-3}}{10}, \quad (6.11)$$

Equation 6.12 and Equation 6.13 get the maximum and the minimum carries $Mcr_{w_z-i-4}, Ncr_{w_z-i-4}$ from the column of index w_z-i-4 to the column of index w_z-i-2.

$$Mcr_{w_z-i-4}=\frac{\sum_{j=-i-4}^{-i-2} Mz_{w_z+j}*Sy_{-i-j-4}+\sum_{j=-i-1}^{-3} Sz_{w_z+j}*Sy_{-i-j-4}+\sum_{j=-2}^{0} Sz_{w_z+j}*My_{-i-j-4}}{100}, \quad (6.12)$$

$$Ncr_{w_z-i-4}=\frac{\sum_{j=-i-4}^{-i-2} Nz_{w_z+j}*Sy_{-i-j-4}+\sum_{j=-i-1}^{-3} Sz_{w_z+j}*Sy_{-i-j-4}+\sum_{j=-2}^{0} Sz_{w_z+j}*Ny_{-i-j-4}}{100}, \quad (6.13)$$

Equation 6.14 and Equation 6.15 get the maximum and the minimum carries $Mcr_{w_z-i-5}, Ncr_{w_z-i-5}$ from the column of index w_z-i-5 to the column of index w_z-i-2.

$$Mcr_{w_z-i-5}=\frac{\sum_{j=-i-5}^{-i-2} Mz_{w_z+j}*Sy_{-i-j-5}+\sum_{j=-i-1}^{-4} Sz_{w_z+j}*Sy_{-i-j-5}+\sum_{j=-3}^{0} Sz_{w_z+j}*My_{-i-j-5}}{1000}, \quad (6.14)$$

$$Ncr_{w_z-i-5}=\frac{\sum_{j=-i-5}^{-i-2} Nz_{w_z+j}*Sy_{-i-j-5}+\sum_{j=-i-1}^{-4} Sz_{w_z+j}*Sy_{-i-j-5}+\sum_{j=-3}^{0} Sz_{w_z+j}*Ny_{-i-j-5}}{1000}, \quad (6.15)$$

After getting the iteration values Sx_{w_z-i-2}, Sz_{w_z-i}, Sz_{w_z-i-1}, Sy_{-i}, Sy_{-i-1}, the algorithm propagates the borrows between the digits of Sx to be in the form of the general Equations 6 to 8. It replaces Sx_{w_z-i} with $Sx_{w_z-i}-br_{w_z-i}$, Sx_{w_z-i-1} with $Sx_{w_z-i-1}+10*br_{w_z-i}-br_{w_z-i-1}$, and Sx_{w_z-i-2} with the $Sx_{w_z-i-2}+10*br_{w_z-i-1}$. Then, the algorithm begins the next outer iteration using the same procedure, and so on until it gets all digits of Sx, Sy, and Sz.

6.1.2 The Division least Digits Constraints Algorithm

The previous algorithm gets the digits of Sx and Sy that satisfy the constraints on the most significant digits of Sz and do not take the constraints of the least digits of Sz in its calculations. Hence, if there are constraints on the least significant digits of the intermediate result significand Sz (that have weight less than 10^{w_z-p}), the previous algorithm alone will not succeed to get a solution in some hard constraints. An example of the hard constraints is a series of zeros or nines in the least digits of Sz, which are needed to verify the rounding process in the different designs.

The least digits algorithm gives the value of the inputs significands of Sx and Sy which yields the needed hard constraints in the intermediate result significand of Sz. This algorithm solves the series of zeros constraint and the series of nines constraint in similar ways starting from right (least significant) to left.

As shown in Figure 8, the intermediate result significand of Sz has a series of zeros from the weight 10^{-17} to 10^{-27}, due to this series of zeros, the elements are decreased in the columns of indexes from -17 to -27. The algorithm solves the nonlinear equations of the columns of indexes from -27 to -16, to get the digits of Sz from Sz_{-16} to Sz_0.

The algorithm gets randomly the elements of the products in the column of index -27, which are Sz_{-16}, Sz_{-15}, Sz_{-14}, Sz_{-13}, Sz_{-12}, Sy_{-15}, Sy_{-14}, Sy_{-13}, Sy_{-12}, Sy_{-11} from their intervals. It calculates the carries cr_{-27}, cr_{-28}, and cr_{-29} of the columns of indexes -27, -28, and -29, then replaces cr_{-27} with $cr_{-27}+cr_{-28}/10+cr_{-29}/100$, such that $cr_{-27} mod_{10}=0$.

Then, the algorithm attempts to solve the non linear equations of the columns of indexes -26, -25, -24. It searches randomly on the combination of values of Sz_{-11}, Sz_{-10}, Sz_{-9}, Sy_{-10}, Sy_{-9}, Sy_{-8} that achieves the conditions $cr_{-26} mod_{10}=0$, $cr_{-25} mod_{10}=0$, and $cr_{-24} mod_{10}=0$. Up to now, the algorithm does the first iteration, gets the digit Sz_{-11}, Sy_{-10}, and estimates the digits Sz_{-10}, Sz_{-9}, Sy_{-9}, Sy_{-8}. In the second iteration, it searches randomly on the values of Sz_{-10}, Sz_{-9}, Sz_{-8}, Sy_{-9}, Sy_{-8}, Sy_{-7} that achieve the nonlinear equations of the columns of indexes -25, -24, -23, to get the digit value of Sz_{-10}, Sy_{-9}, and estimates the digits Sz_{-9}, Sz_{-8}, Sy_{-8}, Sy_{-7}. The algorithm does this procedure in all iterations to get the remaining digits of Sz, from z_{-9} to Sz_{-1}, and the remaining digits of Sy, from Sy_{-8} to Sy_0. The algorithm chooses randomly the remaining digits of Sz which are Sz_0, and multiply the intermediate result significand Sz with the divisor significand Sy to get the dividend significand Sx.

The general form of the nonlinear equations is:

$$cr_n = \sum_{j=n}^{n+p-1} Sy_{n-j} * Sz_j + cr_{n-1}/10 - Sx_n, \qquad (6.16)$$

In general, the algorithm determines the series of zeros after the most p digits in the mask of the intermediate result significand Mz, Nz. The weight of the first zero from the left is denoted by 10^{Fw} and the weight of the last zero in the series is denoted by 10^{Lw}. It gets the digits of Sz from Sz_{Fw+1} to Sz_{Lw-1+p}, and the digits of Sy from Sy_{-p+1} to $Sy_{Lw-1-Fw}$, which are the elements of the products of the column of index Lw. Equation 6.17 gets the value of the carry generated from the column of index Lw.

$10^{w_z}=10^0$ ↑		$10^{Fw}=10^{-17}$ ↑										$10^{Lw}=10^{-27}$ ↑		
$Sz_0 \cdots Sz_{-15}$	Sz_{-16}	0	0	0	0	0	0	0	0	0	0	0	Sz_{-28}	$Sz_{-29}\cdots$
$* \; Sy_0 \cdots Sy_{-15}$														
$\cdots Sy_0 Sz_{-15}$	$Sy_0 Sz_{-16}$	0	0	0	0	0	0	0	0	0	0	0	$Sy_0 Sz_{-28}$	$Sy_0 Sz_{-29}\cdots$
$\cdots Sy_{-1} Sz_{-14}$	$Sy_{-1} Sz_{-15}$	$Sy_{-1} Sz_{-16}$	0	0	0	0	0	0	0	0	0	0	0	$Sy_{-1} Sz_{-28}\cdots$
$\cdots Sy_{-2} Sz_{-13}$	$Sy_{-2} Sz_{-14}$	$Sy_{-2} Sz_{-15}$	$Sy_{-2} Sz_{-16}$	0	0	0	0	0	0	0	0	0	0	$0\cdots$
$\cdots Sy_{-3} Sz_{-12}$	$Sy_{-3} Sz_{-13}$	$Sy_{-3} Sz_{-14}$	$Sy_{-3} Sz_{-15}$	$Sy_{-3} Sz_{-16}$	0	0	0	0	0	0	0	0	0	$0\cdots$
$\cdots Sy_{-4} Sz_{-11}$	$Sy_{-4} Sz_{-12}$	$Sy_{-4} Sz_{-13}$	$Sy_{-4} Sz_{-14}$	$Sy_{-4} Sz_{-15}$	$Sy_{-4} Sz_{-16}$	0	0	0	0	0	0	0	0	$0\cdots$
$\cdots Sy_{-5} Sz_{-10}$	$Sy_{-5} Sz_{-11}$	$Sy_{-5} Sz_{-12}$	$Sy_{-5} Sz_{-13}$	$Sy_{-5} Sz_{-14}$	$Sy_{-5} Sz_{-15}$	$Sy_{-5} Sz_{-16}$	0	0	0	0	0	0	0	$0\cdots$
$\cdots Sy_{-6} Sz_{-9}$	$Sy_{-6} Sz_{-10}$	$Sy_{-6} Sz_{-11}$	$Sy_{-6} Sz_{-12}$	$Sy_{-6} Sz_{-13}$	$Sy_{-6} Sz_{-14}$	$Sy_{-6} Sz_{-15}$	$Sy_{-6} Sz_{-16}$	0	0	0	0	0	0	$0\cdots$
$\cdots Sy_{-7} Sz_{-8}$	$Sy_{-7} Sz_{-9}$	$Sy_{-7} Sz_{-10}$	$Sy_{-7} Sz_{-11}$	$Sy_{-7} Sz_{-12}$	$Sy_{-7} Sz_{-13}$	$Sy_{-7} Sz_{-14}$	$Sy_{-7} Sz_{-15}$	$Sy_{-7} Sz_{-16}$	0	0	0	0	0	$0\cdots$
$\cdots Sy_{-8} Sz_{-7}$	$Sy_{-8} Sz_{-8}$	$Sy_{-8} Sz_{-9}$	$Sy_{-8} Sz_{-10}$	$Sy_{-8} Sz_{-11}$	$Sy_{-8} Sz_{-12}$	$Sy_{-8} Sz_{-13}$	$Sy_{-8} Sz_{-14}$	$Sy_{-8} Sz_{-15}$	$Sy_{-8} Sz_{-16}$	0	0	0	0	$0\cdots$
$\cdots Sy_{-9} Sz_{-6}$	$Sy_{-9} Sz_{-7}$	$Sy_{-9} Sz_{-8}$	$Sy_{-9} Sz_{-9}$	$Sy_{-9} Sz_{-10}$	$Sy_{-9} Sz_{-11}$	$Sy_{-9} Sz_{-12}$	$Sy_{-9} Sz_{-13}$	$Sy_{-9} Sz_{-14}$	$Sy_{-9} Sz_{-15}$	$Sy_{-9} Sz_{-16}$	0	0	0	$0\cdots$
$\cdots Sy_{-10} Sz_{-5}$	$Sy_{-10} Sz_{-6}$	$Sy_{-10} Sz_{-7}$	$Sy_{-10} Sz_{-8}$	$Sy_{-10} Sz_{-9}$	$Sy_{-10} Sz_{-10}$	$Sy_{-10} Sz_{-11}$	$Sy_{-10} Sz_{-12}$	$Sy_{-10} Sz_{-13}$	$Sy_{-10} Sz_{-14}$	$Sy_{-10} Sz_{-15}$	$Sy_{-10} Sz_{-16}$	0	0	$0\cdots$
$\cdots Sy_{-11} Sz_{-4}$	$Sy_{-11} Sz_{-5}$	$Sy_{-11} Sz_{-6}$	$Sy_{-11} Sz_{-7}$	$Sy_{-11} Sz_{-8}$	$Sy_{-11} Sz_{-9}$	$Sy_{-11} Sz_{-10}$	$Sy_{-11} Sz_{-11}$	$Sy_{-11} Sz_{-12}$	$Sy_{-11} Sz_{-13}$	$Sy_{-11} Sz_{-14}$	$Sy_{-11} Sz_{-15}$	$Sy_{-11} Sz_{-16}$	0	$0\cdots$
$\cdots Sy_{-12} Sz_{-3}$	$Sy_{-12} Sz_{-4}$	$Sy_{-12} Sz_{-5}$	$Sy_{-12} Sz_{-6}$	$Sy_{-12} Sz_{-7}$	$Sy_{-12} Sz_{-8}$	$Sy_{-12} Sz_{-9}$	$Sy_{-12} Sz_{-10}$	$Sy_{-12} Sz_{-11}$	$Sy_{-12} Sz_{-12}$	$Sy_{-12} Sz_{-13}$	$Sy_{-12} Sz_{-14}$	$Sy_{-12} Sz_{-15}$	$Sy_{-12} Sz_{-16}$	$0\cdots$
$\cdots Sy_{-13} Sz_{-2}$	$Sy_{-13} Sz_{-3}$	$Sy_{-13} Sz_{-4}$	$Sy_{-13} Sz_{-5}$	$Sy_{-13} Sz_{-6}$	$Sy_{-13} Sz_{-7}$	$Sy_{-13} Sz_{-8}$	$Sy_{-13} Sz_{-9}$	$Sy_{-13} Sz_{-10}$	$Sy_{-13} Sz_{-11}$	$Sy_{-13} Sz_{-12}$	$Sy_{-13} Sz_{-13}$	$Sy_{-13} Sz_{-14}$	$Sy_{-13} Sz_{-15}$	$Sy_{-13} Sz_{-16}\cdots$
$\cdots Sy_{-14} Sz_{-1}$	$Sy_{-14} Sz_{-2}$	$Sy_{-14} Sy_{-3}$	$Sy_{-14} Sz_{-4}$	$Sy_{-14} Sz_{-5}$	$Sy_{-14} Sz_{-6}$	$Sy_{-14} Sz_{-7}$	$Sy_{-14} Sz_{-8}$	$Sy_{-14} Sz_{-9}$	$Sy_{-14} Sz_{-10}$	$Sy_{-14} Sz_{-11}$	$Sy_{-14} Sz_{-12}$	$Sy_{-14} Sz_{-13}$	$Sy_{-14} Sz_{-14}$	$Sy_{-14} Sz_{-15}\cdots$
$\cdots Sy_{-15} Sz_0$	$Sy_{-15} Sz_{-1}$	$Sy_{-15} Sz_{-2}$	$Sy_{-15} Sz_{-3}$	$Sy_{-15} Sz_{-4}$	$Sy_{-15} Sz_{-5}$	$Sy_{-15} Sz_{-6}$	$Sy_{-15} Sz_{-7}$	$Sy_{-15} Sz_{-8}$	$Sy_{-15} Sz_{-9}$	$Sy_{-15} Sz_{-10}$	$Sy_{-15} Sz_{-11}$	$Sy_{-15} Sz_{-12}$	$Sy_{-15} Sz_{-13}$	$Sy_{-15} Sz_{-14}\cdots$
$\cdots Sx_{-15}-1$	9	9	9	9	9	9	9	9	9	9	9	9	9	$9\ \cdots$
												↓ $Lw=-27$		↓ $Lw-2=-29$

Figure 8. The Multiplication of the Intermediate Result with the Divisor at Constraints of Series of Zeros on the Least Digits

Note that, this carry depends on the subtraction value of the column products sum from the value of the digit $Sx_{Lw}=9$, the carry from the column of index $Lw-1$, and the carry from the column of index $Lw-2$. The carry from the column of index $Lw-1$ to the column of index Lw, is the products sum of the column $Lw-1$ divided by 10. The carry from the column of index $Lw-2$ to the column of index Lw, is the products sum of the column $Lw-2$ divided by 100.

$$cr_{Lw}=\sum_{j=Fw+1}^{Lw+p-1} Sy_{Lw-j}*Sz_j-9+\frac{\sum_{j=Fw+1}^{Lw+p} Sy_{Lw-j-1}*Sz_j}{10}+\frac{\sum_{j=Fw+1}^{Lw+p+1} Sy_{Lw-j-2}*Sz_j}{100}, \tag{6.17}$$

Note that the column of index $Lw-1$ has one unknown product Sy_0*Sz_{Lw-1}, and the column of index $Lw-2$ has two unknown products Sy_0*Sz_{Lw-2}, $Sy_{-1}*Sz_{Lw-1}$. The engine assumes the sum value of these unknown products $(Sy_0*Sz_{Lw-1})/10+(Sy_0*Sz_{Lw-2}+Sy_{-1}*Sz_{Lw-1})/100$, to be equal to $(10-(cr_{Lw})mod_{10})$, and replaces cr_{Lw} with $cr_{Lw}+(10-(cr_{Lw})mod_{10})$, in case of a series of zeros, such that $(cr_{Lw})mod_{10}=0$.

In case of a series of nines, the algorithm solves it in the same way like the series of zeros by adding one to the weight of the last nine in the series of nines of the intermediate result significand mask, and replaces cr_{Lw} with $cr_{Lw}-(cr_{Lw})mod_{10}$, such that $(cr_{Lw})mod_{10}=0$.

Then, the algorithm iterates on the iteration indexes $Lw+1\le i\le Fw+1$ to get in each iteration the values of new digits Sy_{i-1-Fw}, Sz_{i-1+p}, and estimates the digits Sy_{i-Fw}, Sy_{i+1-Fw}, Sz_{i+p}, Sz_{i+1+p} which may be refined in next iterations. Then, it does another number of iterations from $Fw+2\le i\le -p$ to check that the previous chosen digits value of Sz and Sy will make $Sx_i=9$ for all $Fw+2\le i\le -p+1$, and chooses the remaining digits of Sz.

Each iteration on $Lw+1\le i\le Fw+1$, it searches randomly on the values of Sy_{i-1-Fw}, Sy_{i-Fw}, Sy_{i+1-Fw}, Sz_{i-1+p}, Sz_{i+p}, Sz_{i+1+p}. It calculates the carries generated from the columns of index i, $i+1$, $i+2$, using Equation 6.18, Equation 6.19 and Equation 6.20, and checks that the carries satisfy the conditions $(cr_i)mod_{10}=0$, $(cr_{i+1})mod_{10}=0$, and $(cr_{i+2})mod_{10}=0$.

$$cr_i=cr_{i-1}/10+\sum_{j=Fw+1}^{i+p-1} Sy_{i-j}*Sz_j-9, \tag{6.18}$$

$$cr_{i+1}=cr_i/10+\sum_{j=Fw+1}^{i+p} Sy_{i+1-j}*Sz_j-9, \qquad (6.19)$$

$$cr_{i+2}=cr_{i+1}/10+\sum_{j=Fw+1}^{i+1+p} Sy_{i-j}*Sz_j-9, \qquad (6.20)$$

The algorithm repeats all the iterations, if the check in any iteration is not achieved. As in the beginning of the algorithm, it chooses randomly the digits in the column of index Lw, and the nonlinear equations in the next iterations depend on these digits. The combination of these digits may fail to satisfy the conditions in the next iteration.

In the iterations of $Lw+1 \le i \le Fw+1$, the algorithm gets digits of Sz from Sz_{Lw+p} to Sz_{Fw+p}, and the digits of Sy from Sy_{Lw-Fw} to Sy_0. The algorithm does other iterations on $Fw+2 \le i \le -p+1$ to get the remaining digits of Sz, and checks that the previous chosen digits of Sz and Sy will make $Sx_i=9$. It gets in each iteration the digit Sz_{i-1+p}, and calculates the carry generated from the column of index i, using Equation 6.21, such that $(cr_i) mod_{10}=0$. This check may make the algorithm fail to get any solution as the number of these iterations increase. As the algorithm has chosen all digits of Sy and the most digits of Sz in the previous iterations without taking in its considerations the nonlinear equations in the iterations of $Fw+2 \le i \le -p+1$. In this case the engine refines the constraints to get the best solution.

$$cr_i=cr_{i-1}/10+\sum_{j=i}^{i+p-1} Sy_{i-j}*Sz_j-9, \qquad (6.21)$$

After getting the needed digits of Sz, and all digits of Sy, the least digits algorithm multiply Sz with Sy, to get Sx. Then it uses the most digits algorithm to get all digits of Sz using the digits of Sx and the digits of Sy.

6.2 Decimal Division Rounding Boundaries

We use the engine to get the hardest-to-round cases and determine the number of digits needed to do the correct rounding according to the standard. The problem termed as “table-maker's-dilemma”[11] appears when the result is inexact and the intermediate result has a series of zeros after p digits, or after

$p+1$ digits. At this case we do not know the value of the sticky bit and therefore we cannot do the correct rounding.

We use the engine to find the largest number of zeros that follow p digits. The largest number of zeros that the engine gets is $p-1$. The engine generates cases at $p=16$ with 15 zeros, and at $p=34$ with 33 zeros . Two examples from these cases are : (1) at $p=16$, when the inputs are $Sx=4140631901663$ and $Sy=9186895982637069$, the result is $Sz=45071065455499420000000000000002177$, (2) at $p=34$, when the inputs are $Sx=1988488448466631984536725650933338$, and $Sy=7825666841614090843966690633705274$, then the intermediate result is $Sz=254098275420129472917015755290485400000000000000000000000000000000005111$.

Lemma2 : At the Decimal Division operation, number of trailing zeros after p digits in the intermediate result significand Sz that might be followed by a non-zero digit cannot be more than or equal to $p+1$.

Proof: Let us assume that $p+1$ zeros or more exist followed by a non zero digit, as shown in Figure 9. The figure shows that the sum of the elements from the column of index -2p to the least columns, must have a carry larger than or equal to 99.

$* \; Sz_0\cdots Sz_{-15}Sz_{-16}\cdots$	0	0	0	0	Sz_{-32}	$Sz_{-33}\cdots$
$Sy_0\cdots Sy_{-15}$						
	$\cdots 0$	0	0	0	$Sy_0 Sz_{-32}$	$Sy_0 Sz_{-33}\cdots$
	$\cdots 0$	0	0	0	0	$Sy_{-1} Sz_{-32}\cdots$
	$\cdots 0$	0	0	0	0	$0\cdots$
	$\cdots 0$	0	0	0	0	$0\cdots$
	$\cdots 0$	0	0	0	0	$0\cdots$
	$\cdots 0$	0	0	0	0	$0\cdots$
	$\cdots 0$	0	0	0	0	$0\cdots$
	$\cdots 0$	0	0	0	0	$0\cdots$
	$\cdots 0$	0	0	0	0	$0\cdots$
	$\cdots 0$	0	0	0	0	$0\cdots$
	$\cdots 0$	0	0	0	0	$0\cdots$
	$\cdots 0$	0	0	0	0	$0\cdots$
	$\cdots 0$	0	0	0	0	$0\cdots$
	$\cdots 0$	0	0	0	0	$0\cdots$
	$\cdots Sy_{-14}Sz_{-16}$	0	0	0	0	$0\cdots$
	$\cdots Sy_{-15}Sz_{-15}$	$Sy_{-15}Sz_{-16}$	0	0	0	$0\cdots$
	$\cdots\; 9$	9	9	9	9	$9\;\cdots$
					$\uparrow$ -2p	

Figure 9. The Multiplication of the Divisor with the Intermediate result that has a series of zeros equals $p+1$.

Let us assume that the each product in those columns has the maximum value which equal to $9*9=81$. At this case the sum of the products of those columns is equal to $1*81+2*81/10+3*81/100+4*81/1000+5*81/10000+\cdots+n*81/10^{n-1}$. This sum of products is less than or equal to 100, which means that the maximum carry of that sum is 10, while for $p+1$ zeros the carry must be larger than or equal to 99. Hence the assumption of $p+1$ zeros or more is invalid and the lemma is proven.

Theorem2: Only $2p+1$ digits not including leading zeros are enough to do the correct rounding to Decimal Floating-Point Division operation.

Proof: Based on the previous lemma, no more than $p+1$ digits are needed after the rounding position to make sure the correct calculation of the sticky bit. Hence the total number of digits is $p+p+1=2p+1$.

6.3 The Main Ideas of the Division Models

The models are defined using a Cartesian product between two or more lists of constraints with ignoring the impossible combinations, and allowing the other constraints to be chosen randomly.

All the model proposal ideas are in [22]and [8], except the ideas of the nines and zeros model. However we describe all the ideas in the form of our engine constraints.

A) Inputs Types Model

The model aims to verify the ability of the division designs to solve all possible combinations of the input types. The proposal ideas of the model are in [22]. We separate the model into five sub-models as follows:

1.It verifies the design when the second input is zero using, (1) a list of the second input exponent consists of the interval $[qmin,qmax]$, (2) the second input significand is equal to zero, (3) all types list of the first input.

2. It verifies the design when the first input is zero using, (1) a list of the first input exponent consists of the interval $[qmin,qmax]$, (2) the first input significand is equal to zero, (3) all types list of the second input.

3. It verifies the design when the first input is Infinity, sNaN, or qNaN using, (1) a list of the first input consists of the Infinities, sNaN, and qNaN, (2) all types list of the second input.

4. It verifies the design when the second input is Infinity, sNaN, or qNaN using, (1) a list of the second from the Infinities, sNaN, and qNaN inputs, (2) all types list of the first input.

5. It verifies the design in solving the other input types using, (1) a list of the first input from the minimum Subnormal input, the maximum Subnormal input, the minimum Normal input, and the maximum Normal input, (2) a same list of the second input.

B) Result Types Model

The model aims to verify the ability of the division design to generate the different types of the final result. The proposal ideas of the model are in [22]. We separate the model into four sub-models as follows:

1. It verifies all the result exponents using, (1) a list of the intermediate result exponent consists of the interval $[qmin, qmax]$.

2. It verifies the generation of the first hundred subnormal numbers, the last hundred normal numbers and the first hundred normal numbers using, (1) the intermediate result exponent is equal $qmin$, (2) a list of the intermediate result significand consists of the intervals $\{[2,100],[10^{p-1}-100,10^{p-1}+100]\}$.

3.It verifies the generation of numbers from one to 100, using, (1) the intermediate result exponent is equal zero, (2) a list of the intermediate result significand from the interval $[1,100]$.

4. It verifies the last hundred Normal numbers using, (1) the intermediate result exponent is equal to $qmax$, (2) a list of the intermediate result significand from the interval $[10^{p}-100,10^{p}-1]$.

C) Rounding Model

The model aims to verify the rounding process in the design. The proposal ideas of the model are in [22]. We separate the model into three sub-models as follows:

3. It verifies the rounding process at the all combinations from the guard digit, the least significand digit, and the sticky bit using, (1) a list from the five rounding modes, (2) a list of the intermediate result significand consists of the guard digit interval $[0,9]$, the least significand digit interval $[0,9]$, and the sticky bit interval $[0,1]$.

4. It verifies the possible carry propagation due to rounding process using, (1) a list from the five rounding modes, (2) a list of the intermediate result significand consists of the cross product of the guard digit interval $[0,9]$, and the patterns $\{\overbrace{99\cdots 9}^{p}, \overbrace{\{0-8\}9\cdots 9}^{p}, \overbrace{X\{0-8\}9\cdots 9}^{p}, \cdots, \overbrace{XX\cdots X\{0-8\}}^{p}\}$.

5. It verifies the sticky bit calculations using, (1) a list of number of digits of the first input significand from the interval $[1,p]$, (2) a list of number of digits of the second input significand from the interval $[1,p]$, (3) a list of the intermediate result significand consists of the patterns $\{\overbrace{\{1-9\}X\cdots X}^{P}0X\cdots X, \overbrace{\{1-9\}X\cdots X}^{P}00X\cdots X, \cdots, \overbrace{\{1-9\}X\cdots X}^{P}\overbrace{00\cdots 00}^{p}X\cdots X\}$.

D)Trailing and Leading Zeros Model

The model aims to verify all the possible trailing and leading zeros in the input significands and the intermediate result significand. The proposal ideas of the model are also in [22]. We separate the model into two sub-models as follows:

1. It verifies the design at all possible trailing and leading zeros in the input significands using, (1) a list of the first input significand, (2) the same list of the second input significand that consists of the patterns

$$
\begin{gathered}
\overbrace{\{1-9\}00\cdots 00}^{P}, \overbrace{0\{1-9\}00\cdots 00}^{P}, \cdots, \overbrace{00\cdots 0\{1-9\}}^{P} \\
\overbrace{\{1-9\}\{1-9\}0\cdots 00}^{P}, \overbrace{0\{1-9\}\{1-9\}0\cdots 00}^{P}, \cdots, \overbrace{00\cdots 0\{1-9\}\{1-9\}}^{P} \\
\overbrace{\{1-9\}X\{1-9\}0\cdots 00}^{P}, \overbrace{0\{1-9\}X\{1-9\}0\cdots 00}^{P}, \cdots, \overbrace{00\cdots 0\{1-9\}X\{1-9\}}^{P} \\
\vdots \\
\overbrace{\{1-9\}XX\cdots X\{1-9\}}^{P}
\end{gathered}
$$

2. It verifies the generation of the trailing and leading zeros in the intermediate result significand using, (1) a list of the intermediate result sigificand from the patterns

$$\overbrace{\{1-9\}00\cdots00}^{p+2},\overbrace{\{1-9\}\{1-9\}0\cdots00}^{p+2},\overbrace{\{1-9\}X\{1-9\}0\cdots00}^{p+2},\cdots,\overbrace{XX\cdots X\{1-9\}}^{p+2}, \qquad (2)$$

a list of number of digits of the first input significand from the interval $[1,p]$, (3) a list of number of digits of the second input significand from the interval $[1,p]$.

E) Zeros and Nines Model

The model aims to verify all the possible patterns of zeros and nines in the input significands and the intermediate result significand. The proposal ideas of the model are all new. We separate the model into four sub-models as follows:

1. It verifies the generation of all patterns of zeros in the intermediate result significand using, (1) a list of the intermediate result significand that consists of

$$\begin{gathered}
\overbrace{\{1-9\}00\cdots0X}^{2p},\overbrace{\{1-9\}00\cdots0XX}^{2p},\cdots,\overbrace{\{1-9\}X\cdots XX}^{2p}\\
\overbrace{X\{1-9\}0\cdots0X}^{2p},\overbrace{X\{1-9\}0\cdots0XX}^{2p},\cdots,\overbrace{X\{1-9\}X\cdots XX}^{2p}\\
\overbrace{XX\{1-9\}0\cdots0X}^{2p},\overbrace{XX\{1-9\}0\cdots0XX}^{2p},\cdots,\overbrace{XX\{1-9\}X\cdots XX}^{2p}\\
\vdots\\
\overbrace{XXX\cdots X\{1-9\}}^{2p}
\end{gathered}$$

2. It verifies the generation of all patterns of nines in the intermediate result significand using, (1)a list of the intermediate result significand that consists of

$$\begin{gathered}
\overbrace{\{1-9\}99\cdots99}^{2p},\overbrace{\{1-9\}99\cdots99X}^{2p},\overbrace{\{1-9\}99\cdots9XX}^{2p},\cdots,\overbrace{\{1-9\}X\cdots XX}^{2p}\\
\overbrace{X\{1-9\}99\cdots99}^{2p},\overbrace{X\{1-9\}99\cdots99X}^{2p},\overbrace{X\{1-9\}99\cdots9XX}^{2p},\cdots,\overbrace{X\{1-9\}X\cdots XX}^{2p}\\
\overbrace{XX\{1-9\}99\cdots99}^{2p},\overbrace{\{1-9\}99\cdots99X}^{2p},\overbrace{XX\{1-9\}99\cdots9XX}^{2p},\cdots,\overbrace{XX\{1-9\}X\cdots XX}^{2p}\\
\vdots\\
\overbrace{XXX\cdots X\{1-9\}}^{2p}
\end{gathered}$$

3. It verifies all patterns of zeros in the input significand using, (1) a list the first input significand, (2) the same list of the second input significand that consists of the patterns

$$\overbrace{\{1-9\}00\cdots0X}^{p},\overbrace{\{1-9\}00\cdots0XX}^{p},\cdots,\overbrace{\{1-9\}X\cdots XX}^{p}$$
$$\overbrace{X\{1-9\}0\cdots0X}^{p},\overbrace{X\{1-9\}0\cdots0XX}^{p},\cdots,\overbrace{X\{1-9\}X\cdots XX}^{2p}$$
$$\overbrace{XX\{1-9\}0\cdots0X}^{2p},\overbrace{XX\{1-9\}0\cdots0XX}^{p},\cdots,\overbrace{XX\{1-9\}X\cdots XX}^{p}$$
$$\vdots$$
$$\overbrace{XXX\cdots X\{1-9\}}^{p}$$

4. It verifies all patterns of nines in the input significands using, (1) a list the first input significand, (2) the same list of the second input significand that consists of the patterns

$$\overbrace{\{1-9\}99\cdots99}^{p},\overbrace{\{1-9\}99\cdots99X}^{p},\overbrace{\{1-9\}99\cdots9XX}^{p},\cdots,\overbrace{\{1-9\}X\cdots XX}^{p}$$
$$\overbrace{X\{1-9\}99\cdots99}^{p},\overbrace{X\{1-9\}99\cdots99X}^{p},\overbrace{X\{1-9\}99\cdots9XX}^{p},\cdots,\overbrace{X\{1-9\}X\cdots XX}^{p}$$
$$\overbrace{XX\{1-9\}99\cdots99}^{p},\overbrace{\{1-9\}99\cdots99X}^{p},\overbrace{XX\{1-9\}99\cdots9XX}^{p},\cdots,\overbrace{XX\{1-9\}X\cdots XX}^{p}$$
$$\vdots$$
$$\overbrace{XXX\cdots X\{1-9\}}^{p}$$

G) Overflow Model

The model aims to verify the overflow cases. The proposal ideas of the model are in [22]and [8]. We separate the model into two sub-models as follows:

1. It verifies the overflow cases when the result exponent is larger than $qmax$, using, (1) a list of the intermediate result exponent from the interval $[qmax-p+1, qmax-qmin]$, (2) a list of number of digits of the second input significand from the interval $[1,p]$.

2. It verifies the overflow cases and the near-overflow cases which need to shift the intermediate result significand to left, using, (1) a list of the intermediate result exponent from the interval $[qmax, qmax+2\mathrm{p}-1]$, (2) a list of number of digits of the first input significand from the interval $[1,p]$, (3)a list of number of digits of the second input significand from the interval $[1,p]$, (4) a list of the intermediate result significand that consists of the patterns $\{\overbrace{\{1-9\}00\cdots0}^{p}00\cdots0,\overbrace{X\{1-9\}00\cdots0}^{p}00\cdots0,\cdots,\overbrace{XX\cdots X\{1-9\}}^{p}00\cdots0\}$, and random digits pattern.

H)Underflow Model

The model aims to verify the underflow cases. The proposal ideas of the model are in [22] and [8]. We separate the model into three sub-models as follows:

1. It verifies the underflow cases when the intermediate result exponent is less than $qmin$ using, (1) a list of the intermediate result exponent from the interval $[qmin-qmax, qmin]$.

2. It verifies the underflow and the near-underflow cases when the result is exact or inexact, using (1) a list of the intermediate result exponent in the interval $[qmin-p, qmin]$, (2) a list of the second input significand (3) a list of the first input significand, such that the difference between number of digits of the first input significand to number of digits of the second input significand is from the interval $[1, p-1]$, (4) a list of the intermediate result significand that consists of $\{\overbrace{\{1-9\}00\cdots0}^{p}00\cdots0, \overbrace{X\{1-9\}00\cdots0}^{p}00\cdots0, \cdots, \overbrace{XX\cdots X\{1-9\}}^{p}00\cdots0\}$, and random digits pattern.

3. It verifies the near-underflow cases and the subnormals numbers using, (1) a list of the intermediate result exponent from the interval $[qmin, qmin+p-1]$, (2) a list of the first input significand, (3) a list of the second input significand, such that the difference between number of digits of the second input significand to number of digits of the first input significand from the interval $[1, p-1]$.

6.4 Previous Work

The Fpgen division algorithm by IBM [1] is given the significand of the quotient Sz and the difference d between the preferred exponent and the actual exponent.

The algorithm separates the problem into three cases:

Case1: The result is exact, $d=0$, and guard digit is equal to zero, it selects a random value for $1<Sy<\frac{10^p}{Sz}$, calculates $Sx=Sy*Sz$, and chooses the exponents such that $Ex-Ey=Ez$.

Case 2: The sticky bit is zero and either the exponent difference is not zero or the guard digit is not zero, the algorithm factorizes $Sz=Sz'.2^{j}.5^{k}$ where Sz' is prime to 10 and $Sz=\frac{Sx}{Sy}.10^{d+1}$, it initializes $Sx=Sz'.2^{max(0,j-d-1)}.5^{max(0,k-d-1)}$ and $Sy=2^{max(0,-j+d+1)}.5^{max(0,-k+d+1)}$, it multiplies Sx and Sy by random factor that keeping their size less than 10^{p}, it computes $Ex-Ey=Ez+d$.

Case3: The sticky bit is one, the algorithm calculates the range of number of digits $1+max(0,d-p)<|Sy|<p+min(0,d-p+1)$ and chooses $Sy\leq\frac{(10^{p}-1).10^{d+1}}{Sz+1}$ within the selected $|Sy|$, it chooses Sx from $Sz.Sy<Sx.(10^{d+1})<(Sz+1).Sy$ within $d+1$ trailing zeros, finally it computes $Ex-Ey=Ez+d.$

This algorithm requires several iteration, but in practical it produces the solution for most values of d. At the last case the algorithm may fail at large values of d, when there is no Sx with $d+1$ trailing zeros in its range. Test cases for large d values are often generated by relaxing the constraint on Sz when possible.

6.5 Comparison

The Fpgen division algorithm cannot solve simultaneous constraints on the inputs significand and the unbounded intermediate result significand, and cannot solve the constraints on the digits that follow the guard digits of the intermediate result significand, while our engine solves these constraints numerically. Both of them cannot find the solution from the first trail, but they find the solution in practical time.

An example to the test vector that generated using our engine, and cannot be generated using Fpgen division algorithms at [8], is at $p=16$, when the inputs are $Sx=4140631901663$ and $Sy=9186895982637069$, the intermediate result is $Sz=450710654554994200000000000000002177$.

6.6 Summary

This chapter represents the main steps that the division engine uses to solve all the constraints numerically. It also describes the main ideas of the coverage models that have been solved by the engine to generate test vectors can verify corner cases in the hardware or software implementations of the decimal floating-point division operation.

The chapter also describes the rounding boundaries of the decimal division operation, which our engine and our models are based on. Therefore, it gives an advantage to the division engine and the division models.

The engine solved the coverage models one time and generated about 339000 test vectors in Decimal128 and about 146000 in Decimal64, the test vectors have proved their efficiency by discovering bugs in Silminds design [7]. Most of bugs are discovered using the rounding models and the zeros and nines model.

Chapter 7

Conclusions

We have presented in this thesis our verification work of five decimal floating-point arithmetic operations which are addition-subtraction, multiplication, fused-multiply-add (FMA), square root, and division operations.

We have presented the algorithms used in each engine to solve the coverage models, and the ideas of these models, to generate test vectors can verify the different implementation of the five decimal floating-point arithmetic operations.

The main Idea of the algorithms in the engines of multiplication, FMA, square root, and division operations, is to solve the nonlinear equations generated from multiplying two significands.

We have succeeded to develop new engines to verify the implementations of FMA and square root operations, and our five engines have succeeded to solve the constraints to describe the corner cases of the operation, which include simultaneous constraints on inputs and intermediate result, and constraints on the unbounded intermediate result.

The generated test vectors of the five operations have proved efficiency, as they have succeeded to discover corner bugs in the five hardware designs of Silminds (addition-subtraction, multiplication, FMA, square root, and division) and in the software designs of DecNumber (FMA, and square root). One of the FMA test vectors that discovered bug in the FMA implementation of DecNumber library (version 3.68) is the test vector

$$d64 * -\ 0\ -1916972343725131\text{E}368\ +311281724013\text{E-}108\ -8846849875104544\text{E}253\ \text{->}\ -5967184560399999\text{E}271\ X$$

where the DecNumber result is $-5967184560400000\text{E}271$, and one of the square root test vectors that discovered bug in the square root implementation of DecNumber library (version 3.68) is the test vector $d64V\ <\ +3862493272490151\text{E}26\ \text{->}\ +6.214896034922990\text{E+}20\ X$, where the DecNumber result is

+6.214896034922991E20 .

There is a need to develop verification technique to verify the other elementary operations. Also our technique is not enough to verify the square root, division, and the elementary operations, where they may need formal verification methods or other verification technique as in [9]. These designs depend on iterative methods, where each iteration depends on the previous iterations, so that the verification technique need to verify the result of each iteration.

Appendix A

Test vectors Syntax

The test vectors are represented in IBM syntax as follows:

1- The type and precision: d64 for Decimal64, or d128 for Decimal128.

2- The operation: + for add, - for subtract, * for multiply, / for divide, *+ for fused-multiply-add, *- for fused-multiply-subtract, or V for square root.

3- The rounding mode: > for (positive infinity), < for (negative infinity), 0 for (zero), =0 for (nearest, ties to even), or h> (nearest, ties away from zero).

4- The data for input operands: <sign><significand>E<exp>. Where the sign is either + or -, the significand is a string of decimal digits, exp is the value of the unbiased exponent written as an integer number.

SNaN numbers are represented using the string S.

QNaN numbers are represented using the string Q.

Infinities are represented using the string <sign>inf.

5- A “->” sign, to separate inputs from results.

6- The data for output operand: <sign><significand>E<exp>. Where the sign is either + or -, the significand is a string of decimal digits, exp is the value of the unbiased exponent written as an integer number.

SNaN numbers are represented using the string S.

QNaN numbers are represented using the string Q.

Infinities are represented using the string <sign>inf.

7- Exceptions that occur following the operation: x (inexact), u (underflow), o (overflow), z (division by zero) and i (invalid).

References

[1] M. Aharoni, R. Maharik, A. Ziv, "Solving Constraints on the Intermediate Result of Decimal Floating-Point," in Proceeding of 18th IEEE Symposium on Computer Arithmetic, 2007.

[2] M. Aharoni, S. Asaf, R. Maharik, I. Nehama, I. Nikulshin, A. Ziv, "Solving Constraints on the Invisible Bits of the Intermediate Result for Floating-Point Verification," in Proceeding of 17th IEEE Symposium on Computer Arithmetic, 2006.

[3] M. Aharoni, S. A. L. Fournier, A. Koifman, and R. Nagel, "FPgen - A Test Generation Framework for Data path Floating-Point Verification," in Proceedings of IEEE International High Level Design Validation and Test Workshop, 2003.

[4] E. M. Clarke, S. M. Germanand, X. Zhao,"Verifying the SRT Division Algorithm Using Theorem Proving Techniques," Formal Methods in System Design, vol. 14, pp. 7-44, 1999.

[5] R. Drechsler, Advanced Formal Verification. Springer, 2004.

[6] A. Y. Duale, M. H. Decker, H. G. Zipperer, M. Aharoni, and T. J. Bohizic, "Decimal Floating-Point in z9: An Implementation and Testing Perspective," IBM Journal of Research and Development, 51, 2007.

[7] H. A. H. Fahmy, R. Raafat, A. M. Abdel-Majeed, R. Samy, T. ElDeeb, Y. Farouk, "Energy and Delay improvement via Decimal Floating Point Units," in Proceeding of 19th IEEE Symposium on Computer Arithmetic, 2009.

[8] E. Guralnik, M. Aharoni, A.J. Birnbaum, A. Koyfman, "Simulation Based Verification of Floating Point Division," in IEEE Transactions on Computers, Feb 2011.

[9] E. Guralnik, A. J. Birnbaum, A. Koyfman, A. Kaplan, “Implementation Specific Verification of Divide and Square Root Instructions,” in Proceeding of 19^{th} IEEE Symposium on Computer Arithmetic, 2009.

[10] J. Harison, “Floating-Point Verification,” in Journal of Universal Computer Science, 2007.

[11] V. Lefevre, J. M. Muller, and A. Tisserand, "Toward correctly Rounded Transcendentals" IEEE Transactions on Computers , vol 47, no 11, pp 1235-1243, Nov 1998.

[12] O. Leary, X. Zhao, R. Gerth, C. Johan, H. Seger, “Formally Verifying IEEE Compliance of Floating-Point Hardware,” Intel Technology Journal, 1999.

[13] R. Raafat, A. M. Abdel-Majeed, R. Samy, T. ElDeeb, Y. Farouk, M. Elkhouly, and H. A. H. Fahmy, “A decimal fully parallel and pipelined floating point multiplier,” in Forty-Second Asilomar Conference on Signals, Systems, and Computers, Asilomar, California, USA, Oct. 2008.

[14] D. Rusinoff, “A mechanically checked proof of IEEE compliance of a register-transfer-level specification of the AMD-K7 floating-point multiplication, division, and square root instructions,” LMS Journal of Computation and Mathematics, vol. 1, pp. 148–200, 1998.

[15] R. Samy, H. A. H. Fahmy, R. Raafat, A. Mohamed, T. ElDeeb and Y. Farouk, “A Decimal Floating-Point Fused-Multiply-Add Unit,” in the 53rd International Midwest Symposium on Circuits and Systems (MWSCAS), Aug 2010.

[16] A. Sayed-Ahmed, H. A. H. Fahmy, M. Y. Hassan, “Three Engines to Solve Verification Constraints of Decimal Floating-Point operations,” in Forty-Four Asilomar Conference on Signals, Systems, and Computers, Nov 2010.

[17] J. Sawada, D. Borrione, M. Kaufmann, and J. Moore, “Formal verification of divide and square root algorithms using series calculation,” in 3rd

International Workshop on the ACL2 Theorem Prover and its Applications, University of Grenoble, pp. 31–49, 2002.

[18] K. Yehia, H. A. H. Fahmy, M. Hassan, “A Redundant Decimal Floating-Point Adder,” in Forty-Four Asilomar Conference on Signals, Systems, and Computers, Asilomar, California, USA, 2010.

[19] A. Ziv, and L. Fournier, “Test Generation for the Binary Floating Point Add Operation With Mask-Mask-Mask Constraints,” Theoretical Computer Science, Vol. 291/2, pp. 183-201,2003.

[20] A. Ziv, M. Aharoni, and S. Asaf, “Solving Range Constraints for Binary Floating-Point Instructions,” in Proceeding of 16th IEEE Symposium on Computer Arithmetic, 2003.

[21]“IEEE standard for floating-point arithmetic,” New York, NY, Aug. 2008, (IEEE Std 754-2008).

[22]“Floating-Point test suite for IEEE 754R standard,” https://www.research.ibm.com/haifa/projects/verification/fpgen/ieeets.html, visited on May 2011.

[23]“The DecNumber library version 3.68,” http://speleotrove.com/decimal/decnumber.html, visited on May 2011.

[24]“Intel® Decimal Floating-Point Math Library,” http://software.intel.com/en-us/articles/intel-decimal-floating-point-math-library/, visited on May 2011.