Three Engines To Solve Verification Constraints of Decimal Floating-Point Operation

Amr A. R. Sayed-Ahmed, Hossam A. H. Fahmy
Electronics and Communications Department
Cairo University
Giza, Egypt
Email: hfahmy@stanfordalumni.org

Mahmoud Y. Hassan
Silminds, Maadi, 11431
Helwan, Egypt

***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 represents three engines, the first engine for the verification of decimal addition-subtraction operation, the second for the verification of decimal multiplication operation, and the third for the verification of decimal fused-multiply-add operation. Each engine solves constraints describing all corner cases of the operation, and generates test vectors to verify these corner cases in the tested design. The paper describes the constraints of each operation and the steps of each engine to solve these constraints.**

Keywords-component; Verification; Decimal Arithmetic Operations; Simulation based coverage models

I. Introduction

Decimal floating-point implementations as software or hardware based designs have many advantages over binary floating-point especially in the financial and commercial applications[1]. As decimal floating-point is newly defined in the IEEE Standard for Floating-Point Arithmetic (IEEE Std 754-2008)[2], new verification technologies are needed to verify the compliance of the decimal floating-point designs with the standard.

Our decimal floating-point verification method is 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 other hand the random simulation does not guarantee a good coverage due to the large space of the inputs where for *n* operation's inputs of 16 digits precision, the space is on the order of $10^{(n*16)}$ and for 34 digits it becomes $10^{(n*34)}$.

The coverage model consists of tasks, each task represents the constraints of a certain case. 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 with ignoring the impossible combinations.

The Addition-Subtraction, Multiplication, and Fused- Multiply-Add (FMA) engines are software tools to solve constraints on inputs, output, and specific features related to the operation. Each engine uses one algorithm to solve these constraints and another one to solve the special inputs constraints like Zero, sNaN, qNaN, and Infinity. These algorithms allow the engines to solve all constraints analytically including simultaneous constraints on inputs and output.

Formal verification methods[3,4] and simulation verification methods based on coverage models[5,6,7] have been developed to verify binary floating-point hardware designs. The verification of decimal floating-point using simulation method based on coverage models[8] is developed but still the used algorithms do not guarantee to find the solution of certain cases. They can not solve simultaneous constraints on inputs and output, and can not solve constraints on an unbounded intermediate result. Also there are no algorithms to solve constraints of FMA, square root, and power function.

Our engines use algorithms to solve simultaneous constraints on inputs and output and constraints on the unbounded intermediate result, and can find the solution of most cases if the solution exits.

The three engines are used for the verification of SilMinds decimal floating-point hardware implementations[1,9,10], and research decimal floating-point designs at cairo university[11]. The generated test vectors have proven the efficiency of the engines in discovering bugs.

II.Main Definitions

The standard defines two representations to decimal floating-point number: a normalized format and a unnormalized format. The engines represent decimal floating-point number in the unnormalized format. So that the number is defined as $(-1)^s(d_{P-1}d_{P-2}d_{P-3}...d_0)10^q$ where s is the sign, $d_{P-1}d_{P-2}d_{P-3}...d_0$ is the significand where $d_i=\{0,1,2,3,4,5,6,7,8,9\}$, and q is the exponent where $qmin \le q \le qmax$, $qmax = emax - p + 1$, $qmin = emin - p + 1$. The precision p is the maximum number of digits in the significand. $emax$ is the maximum exponent, and $emin = 1 - emax$ is the minimum exponent [2].

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 rounding or normalizing processes.

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-

The project "promoting Egypt as the First Decimal Arithmetic Intellectual Property Cores Provider for Financial Applications in the World" (grant number C2/S1/163) is 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.

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.

The mask of p digits significand consists of two numbers: the minimum absolute value of significand $d_{N_{P-1}}d_{N_{P-2}}...d_{N_1}d_{N_0}$ and the maximum absolute value $d_{M_{P-1}}d_{M_{P-2}}...d_{M_1}d_{M_0}$ where each digit d_i is chosen from the interval $[d_{N_i}, d_{M_i}]$, N for minimum, and M for maximum.

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.

III. Addition-Subtraction Engine

The addition-subtraction task is generated from the following constraints: (1) mask on significand of the first input that is set as the smaller exponent input. The mask is represented using two numbers with same signs denoted by mx_N, mx_M, (2) mask on significand of the second input that is set as the larger exponent input. The mask is represented using two numbers with same signs denoted by my_N, my_M, (3) mask on significand of the intermediate result is represented using two numbers with same or different signs denoted by mz_N, mz_M. Each number consists of 2p+1 digits, $p+1$ digits before the fractional point and p digits after it, (4) right shift value of the smaller exponent input str, (5) The intermediate result exponent $Iexp$ at which the addition_subtraction operation occurs, and (6) the rounding mode rm.

The engine chooses randomly the value of the previous constraints that are not specified by the task. We give it the ability to choose from all the possible values of a certain constraint, which give the power to the generated test vectors in discovering bugs

The engine determines the number of digits of the first input significand p_x from the interval $[no\ of\ digits\ of\ mx_N, no\ of\ digits\ of\ mx_M]$, and number of digits of the second input significand p_y from the interval $[no\ of\ digits\ of\ my_N, no\ of\ digits\ of\ my_M]$.

The engine shifts to left both my_N and my_M, which are the significand mask of the larger exponent input, with the value of sl_y, and shifts to right both mx_n and mx_M, which are the significand mask of the smaller exponent input, with the value of sr_x. It chooses randomly the right-shift value sr_x either from the interval $[1, p]$ or from the interval $[p+1, qmax-qmin]$ according to the value of str. If sr_x is equal zero, it will choose randomly left-shift value sl_y from the interval $[0, p-p_y]$, otherwise if sr_x is larger than zero, sl_y is equal $p-p_y$.

After the shifting process, the engine determines the range of each digit of the intermediate result significand. It determines the range of the total value of intermediate result significand as: $z_N = max(mx_N + my_N, mz_N)$, $z_M = min(mx_M + my_M, mz_M)$. Then, it limits the interval of each digit, from the p digits after the point as :

$$z_N[i] = max(mx_N[i] + my_N[i], mz_N[i]): \quad -1 \le i \le -p.$$

$$z_M[i] = min(mx_M[i] + my_M[i], mz_M[i]): \quad -1 \le i \le -p.$$

Depending on the significand mask of the intermediate result, the engine chooses from the interval $[z_N, z_M]$ the intermediate result significand Sz_{inter}. However, it needs first to propagate the borrows from the most digits to the least digits of z_N or z_M to make $z_M[i]$ larger than or equal $z_N[i]$ for all i using $z_M[i+1] = z_M[i+1] - 1$, $z_M[i] = z_M[i] + 10$, where the sign of Sz_{inter} is positive, otherwise, when the sign of Sz_{inter} is negative, it uses $z_N[i+1] = z_N[i+1] + 1$, $z_N[i] = z_N[i] - 10$.

At no constraint on the digit, the engine chooses the digit from its range $z_N[i] \le Sz_{intr}[i] \le z_M[i] \quad p \le i \le -p$. If $mz_N[i]$ is equal to $mz_M[i]$ (i.e. there is a constraint on the digit), $Sz_{inter}[i]$ is equal $mz_N[i]$, or $Sz_{inter}[i]$ is equal $mz_N[i] + 10$, such that the value of $Sz_{inter}[i]$ is between the range of the interval $[z_N[i], z_M[i]]$, then it calculates the total value using $Sz_{inter} = \sum_{i=-P}^{i=P} Sz_{inter}[i] * 10^i$.

The engine must get first the significand of the larger exponent input Sy from the interval $[y_N, y_m]$, where $y_N = max(my_N, Sz_{inter} - mx_M)$, $y_M = min(my_M, Sz_{inter} - mx_N)$. It uses the same method, used to find Sz_{inter}, again to find Sy depending on the mask of significand of the larger exponent input and it calculates total value using $Sy = \sum_{i=0}^{i=P-1} Sy[i] * 10^i$, and it finds easily the second input significand $Sx = Sz_{inter} - Sy$.

The intermediate result exponent Ez_{inter} either equals $Iexp$ or is calculated using: $q_{min} + sr_x \le Ez_{inter} \le q_{max} - sl_y$. It calculates the smaller exponent $Ex = Ez_{inter} - sr_x$ and the larger exponent $Ey = Ez_{inter} + sl_y$. The engine shifts Sx to the left with value of sr_x, and shifts to right Sy with a value of sl_y.

The intermediate result may have cancellation digits when sr_x is larger than zero, in that case the engine shifts Sz_{inter} to left and decreases Ez_{inter} with a value $scn = min(sr_x, p - no\ of\ digits\ before\ point)$. The intermediate result also may have a carry digit, in that case the engine shifts Sz_{inter} one digit to the right and increases Ez_{inter} 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_{inter} one digit to the right and increase Ez_{inter} by one. If Ez_{inter} is smaller than or equals $qmax$, Sz equals Sz_{inter} and Ez equals Ez_{inter}, otherwise it is an overflow case, its result is according to the rounding mode.

IV. Multiplication Engine

The multiplication task is generated from the following constraints: (1) mask on significand of the first input is represented using two numbers with same signs mx_N and mx_M, (2) mask on significand of the second input is represented using two numbers with same signs my_N and my_M, (3) mask on significand of the intermediate result is represented using two numbers with same signs denoted by mz_N and mz_M. Each number consists of 2p digits, (4) the exponent of the first input $Xexp$, (5) the intermediate result exponent $Iexp$ which is the sum of the two inputs exponents, and (6) the rounding mode rm.

The engine also chooses randomly the value of the previous constraints that are not specified by the task.

In the first, the engine determines value of the maximum number of digits of the first input p_x and maximum number of digits of the second input p_y using :

$$min(p_z - no\ of\ digits\ of\ my_M, p) \le p_x \le no\ of\ digits\ of\ mx_M. \quad (1)$$

$$p_y = p_z - p_x. \quad (2)$$

Where p_z is number of digits of the intermediate result. The previous two equations solve the problem of the leading zero digits in the intermediate result significand.

The engine solves the constraints and generates the significands of Sz_{inter}, Sx and Sy, by solving the nonlinear equations that generated from the multiplication of Sx and Sy. Fig.1 shows a simplified multiplication example with $p=8$, from it we see the following facts. The $2p-1$ digits of Sz_{inter} lead to $2p-1$ non linear equations in 2p unknowns (digits of Sx and digits of Sy).The sum of digits in each column in addition to any carries from the previous columns lead to one equation.

$$\begin{array}{r} x_7 x_6 x_5 x_4 x_3 x_2 x_1 x_0 \\ \underline{y_7 y_6 y_5 y_4 y_3 y_2 y_1 y_0} \end{array}$$

$$\begin{array}{r} x_7y_0\ x_6y_0\ x_5y_0\ x_4y_0\ x_3y_0\ x_2y_0\ x_1y_0\ x_0y_0 \\ x_7y_1\ x_6y_1\ x_5y_1\ x_4y_1\ x_3y_1\ x_2y_1\ x_1y_1\ x_0y_1 \\ x_7y_2\ x_6y_2\ x_5y_2\ x_4y_2\ x_3y_2\ x_2y_2\ x_1y_2\ x_0y_2 \\ x_7y_3\ x_6y_3\ x_5y_3\ x_4y_3\ x_3y_3\ x_2y_3\ x_1y_3\ x_0y_3 \\ x_7y_4\ x_6y_4\ x_5y_4\ x_4y_4\ x_3y_4\ x_2y_4\ x_1y_4\ x_0y_4 \\ x_7y_5\ x_6y_5\ x_5y_5\ x_4y_5\ x_3y_5\ x_2y_5\ x_1y_5\ x_0y_5 \\ x_7y_6\ x_6y_6\ x_5y_6\ x_4y_6\ x_3y_6\ x_2y_6\ x_1y_6\ x_0y_6 \\ x_7y_7\ x_6y_7\ x_5y_7\ x_4y_7\ x_3y_7\ x_2y_7\ x_1y_7\ x_0y_7 \\ \hline \bar{z}_{14}\ \bar{z}_{13}\ \bar{z}_{12}\ \bar{z}_{11}\ \bar{z}_{10}\ \bar{z}_9\ \bar{z}_8\ \bar{z}_7\ \bar{z}_6\ \bar{z}_5\ \bar{z}_4\ \bar{z}_3\ \bar{z}_2\ \bar{z}_1\ \bar{z}_0 \end{array}$$

Figure 1. The products of the multiplication operation at precision equal eight.

The engine uses two methods to solve the non-linear equations, It chooses the appropriate method according to the constraints of the intermediate result. If the intermediate constraints are on the least p digits, the engine solves the significands constraints using the first p equations of the operation:

$$\begin{array}{c} x_0 * y_0 - \bar{z}_0 = c_0 \\ x_1 * y_0 + x_0 * y_1 + c_0/10 - \bar{z}_1 = c_1 \\ \vdots \\ x_{P-1} * y_0 + x_{P-2} * y_1 + \cdots + x_1 * y_{P-2} + x_0 * y_{P-1} + c_p/10 - \bar{z}_{P-1} = c_{p-1} \end{array}$$

But if the intermediate constraints are on the most p digits and some or all the least digits, the engine solves the significands constraints using the last p equations :

$$\begin{array}{c} \bar{z}_{2p-2} - x_{p-1} * y_{p-1} = b_{2p-2} \\ \bar{z}_{2p-3} + 10 * b_{2p-2} - x_{p-1} * y_{p-2} - x_{p-2} * y_{p-1} = b_{2p-3} \\ \vdots \\ \bar{z}_{p-1} + 10 * b_p - x_{p-1} * y_0 - x_{p-2} * y_1 - \cdots - x_1 * y_{p-2} - x_0 * y_{p-1} = b_{p-1} \end{array}$$

In the two methods, the engine achieves the constraint of each digit $Sx[i]$, $Sy[i]$, or $Sz_{inter}[i]$ by choosing the digit from its interval $[mx_N[i], mx_M[i]]$, $[my_N[i], my_M[i]]$, or $[mz_N[i], mz_M[i]]$.

In the first method, the engine chooses $Sx[0]$, $Sy[0]$, and $Sz_{inter}[0]$, from their intervals, such that they achieve the equation:

$$\begin{array}{c} Sx[0] * Sy[0] - Sz[0]_{inter} = c[0], \\ (c[0])\ Mod_{10} = 0. \end{array} \quad (3)$$

It chooses the remain digits, digit by digit, for all i in the interval $[1, p-1]$, such that $Sx[i]$, $Sy[i]$, and $Sz_{inter}[i]$ achieve the equation:

$$\begin{array}{c} \sum_{j=0}^{j=i} Sx[i-j] * Sy[j] + c[i-1]/10 - Sz_{inter}[i] = c[i] \\ (c[i])\ mod_{10} = 0 \end{array} \quad (4)$$

Finally, after getting all digits of Sx and Sy, it calculates the intermediate result significand $Sz_{inter} = Sx * Sy$, to get all digits of Sz_{inter}. The engine repeats the method again, if Equation.4 is not achieved in one of the iterations.

In the second method, the engine chooses first, the five digits $Sx[p-1]$, $Sy[p-1]$, $Sz_{inter}[2p-1]$, $Sz_{inter}[2p-2]$, and $Sz_{inter}[2p-3]$ from their intervals to achieve the equations:

$$\begin{array}{l} b[2p-2] = \\ (Sz_{inter}[2p-2] + 10 * Sz_{inter}[2p-1] - Sy[p-1] * Sx[p-1]) \end{array} \quad (5)$$

$$\begin{array}{l} 0 \le 10 * b[2p-2] + Sz_{inter}[2p-3] \\ \qquad \le 9 * Sx[p-1] + 9 * Sy[p-1] + 28 \end{array} \quad (6)$$

Where the term $28 = 3*9*9/10 + 4*9*9/100 + 5*9*9/1000$ represents the maximum carry from the next equations.

Then, the engine chooses in each iteration the digits $Sx[i]$, $Sy[i]$, $Sz_{inter}[i+p-1]$, and $Sz_{inter}[i+p-2]$ from their intervals, for all i in the interval $[p-2, 0]$, to achieve the equations:

$$b[i+p-1]=Sz_{inter}[i+p-1]+10*b[i+p]-\sum_{j=i}^{j=p-1} Sx[j]*Sy[i-j+p-1] \tag{7}$$

$$0\le 10*b[i+p-1]+Sz_{inter}[i+p-2]-\sum_{\substack{j=i-1\\ j\neq -1}}^{j=p-2} Sx[j]*Sy[i-j+p-2] \le 9*Sx[p-1]+9*Sy[p-1]+28.$$

where $i\neq 0$ (8)

$$0\le 10*b[i+p-1]+Sz_{inter}[i+p-2]-\sum_{j=0}^{j=p-2} Sx[j]*Sy[i-j+p-2] \le 28$$

where $i=0$. (9)

After getting all digits of Sx and Sy, it calculates the intermediate result significand $\bar{Sz}_{inter}=Sx*Sy$. The engine repeats the method more than one time in two cases, first if Equation.7 or Equation.8 is not achieved, second if Equation.10 is not achieved. The engine checks the achievement of the constraints in the least p digits of $\bar{Sz}_{inter}$, and gets new intermediate result significand Sz_{inter}, by changing the digits that do not achieve their constraints to values that achieve the constraints. Such that Sz_{inter} achieves also equation.10.

$$(|Sz_{inter}-\bar{Sz}_{inter}|) mod\ Sx \le maxerror, \ \vee \ (|Sz_{inter}-\bar{Sz}_{inter}|) mod\ Sy \le maxerror \tag{10}$$

If Equation.10 is achieved, it calculates the new values of Sx or Sy using :

$$Sx=Sx+\frac{(Sz_{inter}-\bar{Sz}_{inter})-(Sz_{inter}-\bar{Sz}_{inter}) mod\ Sy}{Sy} \tag{11}$$

$$Sy=Sy+\frac{(Sz_{inter}-\bar{Sz}_{inter})-(Sz_{inter}-\bar{Sz}_{inter}) mod\ Sx}{Sx} \tag{12}$$

Finally it calculates the new value of the intermediate result using $Sz_{inter}=Sx*Sy$. Such that it has an error less than $maxerror$ in the least digits, but still the value of Sz_{inter} is between the range of the interval $[mz_N, mz_M]$.

The engine chooses the intermediate result exponent Ez_{inter} either from the interval $[qmin, qmax]$, or Ez_{inter} is equal $Iexp$. Then, it chooses the first input exponent Ex, either from the interval $[max(qmin, Ez_{inter}-qmax), min(qmax, Ez_{inter}-qmin)]$, or Ex is equal $Xexp$, and it calculates the second input exponent $Ey=Ez_{inter}-Ex$.

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

At clamping, where $Ez>qmax \wedge Ez+p_z\le qmax+p$, the engine shifts to left Sz_{inter} with a value is equal $Ez-qmax$ and equals Ez with $qmax$.

At special case of under flow, where $Ez<qmin$ and $Ez+p_z\ge qmin$, it shifts to right Sz_{inter} with a value is equal $qmin-Ez$ and equals Ez with $qmin$.

The engine rounds the intermediate result according to the standard. The rounding process may generate a carry to force the engine to shift Sz_{inter} one digit to right and increase Ez by one. Finally if Ez is smaller than or equal $qmax$, and larger than or equal $qmin$, Sz will equal Sz_{inter}, otherwise it is an overflow case or an underflow case, its result is according to the rounding mode.

V. Fused Multiply Add Engine

The fused-multiply-add(FMA) task is generated from the following constraints: (1) mask on significand of the first input, is represented using two numbers with the same signs mx_N and mx_M, (2) mask on significand of the second input, is represented using two numbers with the same signs my_N and my_M, (3) mask on significand of the third input is represented using two numbers with the same signs mb_N and mb_M, (4) mask on significand of the multiplication intermediate result is represented using two numbers with the same signs denoted by mz_N, mz_M. Each number consists of 2p digits, (5) mask on significand of the addition intermediate result is represented using two numbers with same or different signs are denoted by mc_N and mc_M. Each number consists of $2p+1$ digits, $p+1$ digits before the fractional point and p digits after it, (6) the exponent of the first multiplication 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), (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 engine also chooses randomly the value of the previous constraints that are not specified by the task.

The FMA Engine does similar steps like the addition -subtraction engine to solve the significand constraints on the third input, the multiplication intermediate result, and the addition intermediate result. It generates the third input significand Sb, an estimation to the multiplication intermediate result significand Sz_{inter}, and an estimation to the addition intermediate result Sc_{inter} (the final intermediate result), after determining the smaller exponent input according to the value of sid.

The engine shifts Sz_{inter} to left with a value msr to be in the format of $2p$ digits, then does similar steps like the multiplication engine to generate the first two significands inputs of the operation Sx, Sy. Therefore it does the FMA operation $Sc_{inter}=Sx*Sy+Sb$ to get the correct final intermediate result of the operation.

The engine solves the exponents constraints depending on the right shift value sr, the left shift value sl, the value

msr, and the value of sid. Where $sid=0$, if the third input is the smaller exponent input of the addition operation, and $sid=1$, if the multiplication intermediate result is the smaller exponent input of the addition operation. The engine chooses the result exponent Ec_{inter} from the interval $[max(|(sr+msr)*sid-2*qmin, sr*(1-sid)-qmin|,\ qmax-msr*sid]$. It calculates the third input exponent $Eb=Ec_{inter}-sr*(1-siz)+sl*siz$, and the multiplication intermediate result $Ez_{inter}=Ec_{inter}-msr-sr*siz+sl*(1-siz)$, then, it chooses Ex from the interval $[max(qmin, Ez_{inter}-qmax), min(qmax, Ez_{inter}-qmin)]$ and calculates $Ey=Ez_{inter}-Ex$.

VI. Conclusion

We have developed the method by Aharoni[8] to generate test cases to verify all corner cases of decimal floating-point operations. The engines solve the constraints to describe all corner cases of the operation, which include simultaneous constraints on inputs and output, and constraints on the unbounded intermediate result.

The three engines Addition-Subtraction, Multiplication, and FMA generated test vectors that proved the efficiency of the engines in solving the chosen constraints.

The three engines do not use the previous explained algorithms in solving special inputs like Zero, sNaN, qNaN, and Infinity. The engines solve these inputs and generate the output typically using the standard rules.

There is a need to develop new engines to solve the simultaneous constraints and the unbounded intermediate result constraints of Division and Square root operations which are specified by the standard but have not been generated yet.

References

[1] 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.

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

[3] 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.

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

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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), 2010.

[11] 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.

[12] 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.