Turkish Journal of Electrical Engineering & Computer Sciences

http://journals.tubitak.gov.tr/elektrik/

Research Article

Turk J Elec Eng & Comp Sci
(2019) 27: 471 – 483
© TÜBİTAK
doi:10.3906/elk-1806-138

A memory-efficient canonical data structure for decimal floating point arithmetic systems modeling and verification

Mohammad Saeed JAHANGIRY, Saeed SAFARI*
Department of Computer Architecture, School of Electrical & Computer Engineering, Faculty of Engineering, University of Tehran, Tehran, Iran

Received: 23.06.2018 • **Accepted/Published Online:** 24.10.2018 • **Final Version:** 22.01.2019

Abstract: Decimal floating point (DFP) number representation was proposed in IEEE-754-2008 in order to overcome binary floating point inaccuracy. Neglecting binary floating point verification has resulted in significant validity and economic losses. Formal verification can be a solution to similar DFP design problems. Verification techniques aiming at DFP are limited to functional methods whereas formal approaches have been neglected and traditional decision diagrams cannot model DFP representation complexity. In this paper, we propose an efficient canonical data structure that can model DFP properties. Our novel data structure models coefficient, exponent, sign, and bias of a DFP number. We will prove mathematically that our data structure is canonical. We will also show that the size of the proposed data structure will grow linearly for a DFP number for all format lengths, so our data structure can be built with a reasonable amount of memory and run time. Experimental results support our mathematical discussion for linear growth of the DFP data structure. Moreover, comparison of our data structure with integer decision diagrams reveals that our model is also efficient for modeling integer functions when utilized as an integer level diagram.

Key words: Formal verification, decision diagrams, decimal floating point

1. Introduction

Decimal floating point (DFP) number representation, standardized in IEEE-754-R, results in a higher level of computation accuracy compared to binary floating point (BFP) for arithmetic operations. Design of new architectures for DFP arithmetic cores and DFP digital signal processing (DSP) systems requires substantial verification efforts in order to prevent million-dollar financial loss recurrences similar to the Pentium floating point bug [1, 2]. In contrast to simulation-based techniques, which suffer from a lack of completeness, formal methods can provide a robust mathematical approach towards design verification of newly designed DFP architectures. Thus, in order to formally verify DFP DSP systems, a novel and efficient data structure that is capable of modeling both DFP numbers and basic arithmetic operations is required. The IEEE-754 standard [3] for BFP number representation is a widespread standard used for arithmetic calculations. It could represent a large range of numbers. A problem is raised, however, when some real numbers have no accurate representation in this system. In fact, many real numbers between any two consecutive BFP numbers exist that have no accurate representation according to IEEE-754. For example, the closest representation of 0.11 is 0.10999999940395355 in 64-bit format. IEEE-754-R was proposed to overcome this limitation using decimal representation [4]. DFP number representation is capable of representing any real number regarding its fixed accuracy. Its precision

*Correspondence: saeed@ut.ac.ir

depends on the format chosen: 16 digits for 64-bit format *(p = 16)* and 34 digits for 128-bit format *(p= 34)*. DFP arithmetic cores were also integrated in some CPUs. IBM designed a hardware engine for a DFP unit in z9 [5], z10, and z196 [6]. Intel provided some software libraries to support this concept [7]. Many other works like [8] provided arithmetic architectures for DFP operations. Unfortunately, verification of DFP units has received limited attention in the literature. To the best of our knowledge no significant research exists aiming at formal verification of DFP units. The rest of this paper is organized as follows. In Section 2 we provide some preliminary information on the DFP number representation standard. We then present related work on formal verification in Section 3. We focus on arithmetic circuit verification methods and seek floating point modeling approaches in that section. Our proposed data structure for DFP modeling is given in Section 4. Section 5 argues the canonicity of our modeling. Experimental results are provided in Section 6. Finally, Section 7 concludes the paper.

2. Decimal floating point standard

DFP notation was standardized in IEEE-754-R. The standard defines three data types: 32-bit, 64-bit, & 128-bit. The number representation base can be either 2 or 10, so that both binary and decimal representations are supported. In this paper we focus on decimal representation. Figure 1 shows bit formatting for 64-bit notation along with each field's bit length.

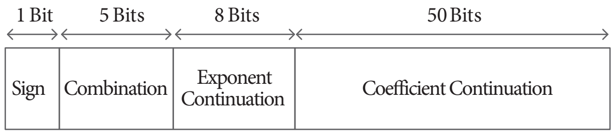

Figure 1. Decimal floating point 64-bit representation field.

In the 64-bit format there is one bit for the Sign (S) field, five bits for the Combination (C) field, eight bits for the Exponent Continuation (EC) field, and 50 bits for the Coefficient Continuation (CC) field. The DFP value is calculated as in Eq. (1). Note that the representation base is 10.

$$Value = -1^{Sign} \times Coefficient \times 10^{Exponent-Bias} \tag{1}$$

The sign is either 0 or 1, depending on the Sign field. Bias is 398 for the 64-bit format. The exponent and coefficient are derived from C, EC, and CC fields. The coefficient in a DFP number is determined in two steps: 1- The most significant digit (MSD) is determined by the Combination field. 2- Remaining digits are determined by the Coefficient Continuation field. The exponent of a DFP number is also constructed in two steps:

1. Two most significant bits (MSBs) of an exponent are decoded using the Combination field.

2. The remaining bits are decoded using the EC field.

According to IEEE-754-R, densely packed decimal (DPD) encoding is used in order to represent three decimal digits using 10 bits (instead of 12 bits). In the case of 64-bit format, 50 bits of the CC field are categorized in five groups of 10 bits, resulting in 15 digits. The MSD of the coefficient is decoded using the Combination field. Totally we have 16 decimal digits, which constitute the DFP coefficient of a 64-bit representation. A similar procedure could be easily applied to other formats.

3. Related work

Verification of arithmetic circuits and systems is performed by either simulation or mathematical methods [9]. In this paper, we focus on the latter methods. Two main approaches in this field are deductive proofs by means of theorem proving and algorithmic decision diagram-based methods. We provide a general overview of formal methods used in arithmetic circuits and systems verification. More specifically, we focus on floating point modeling and verification.

Arithmetic circuits modeling and verification methods fall into two categories: deductive proofs and decision diagram-based algorithms. Our experience with theorem-proving as the main approach to deductive proofs shows that it requires a high level of knowledge of the tools and it lacks automation potentials. Different proof systems like higher order logic (HOL) [10] differ in their logic level points of view.

Algorithmic decision diagram-based methods, on the other hand, are more promising since they provide an opportunity for automation. Model checking, equivalence checking, symbolic simulation, etc. are the most common methods. The binary decision diagram (BDD) served as a basis for the reduced ordered binary decision diagram (ROBDD). The ROBDD can model logical functions using a canonical representation [11]. Modeling integer domain functions was made feasible by multiterminal BDDs (MTBDDs) [12]. Similar to the ROBDD, the MTBDD uses the Shannon decomposition type. Ordered functional decision diagrams (OFDDs) and their extensions are used to minimize fixed polarity Reed–Muller (FPRM) expressions, which can represent Boolean functions canonically with a better performance compared to other binary diagrams [13–16]. Integer functions, however, cannot be modeled directly. One cannot even think of modeling floating point circuits.

Hierarchical decision diagrams were proposed in [17] and [18] as edge-valued binary decision diagrams (EVBDDs). This data structure can model integer functions; nevertheless, higher level arithmetic circuits like floating points are far from the scope of this approach. The binary moment diagram (BMD) and its extension, the *BMD, use the positive Davio decomposition type for mapping from the Boolean domain to integer range [19, 20]. More extension to BMDs resulted in K*BMD [21] and HDD [22] with more expressive power, but none of them can model floating point representation.

In order to leverage the expressive power of modeling methods, the Taylor expansion diagram (TED) was derived [23, 24]. Similar to other diagrams there are rules for ordering, normalization, and isomorph subgraph removal. Enhanced TED [25] and TED+ [26] are some extensions of this data structure. However, if a function has no Taylor expansion around zero, there would be no TED representation. Floating point modeling is still beyond the capabilities of the TED. Another representation trying to improve the performance of modeling is the Horner expansion diagram (HED) and its modular version, the M-HED [27, 28]. These representations, though powerful, are not capable of modeling floating points. Binary floating point formal modeling was studied in [29, 30]. A new data structure, namely the multiplicative power hybrid decision diagram (*PHDD), has been introduced, which can model binary floating point standards. Edges in this model have different weights. Basic arithmetic operations were modeled and the paper reported its performance for integer multiplier circuits. It was compared with BMDs in terms of speed and memory usage. DFP was not the subject of [29]; however, it could be used as a hint for DFP modeling.

Simulation-based verification of DFP square root operation was performed in [31] by verifying corner cases of the operation. Fused-Multiply-Add operation was verified functionally in [32] using constraint solving of corner cases. The authors of [33] integrated the works of [31] and [32] and proposed three engines to solve simulation-based verification constraints for basic DFP operations. None of the mentioned works addressed DFP verification formally, but rather functionally.

To the best of our knowledge there is no prior work on DFP modeling. Most of the data structures in the literature can model Boolean (BDD, ROBDD, EVBDD, FPRM, etc.) or integer (BMD, *BMD, TED, HED, etc.) functions. The PHDD can model only binary floating points, not decimal standards; thus, there is no prior work for modeling decimal floating point arithmetic and our research provides the first data structure to overcome this limitation. Because of this, there are no previous results in the literature to be compared with our experiments. However, we can use our data structure as an integer level decision diagram as well. In this case, comparison with some other methods is feasible. We will address this issue in Section 7.

4. Proposed DFP data structure

In this section we propose a novel data structure suitable for formal modeling of DFP arithmetic systems. Any proposed model for a DFP system, including ours, should be able to model sign, coefficient, and exponent efficiently. To do so we use different decomposition rules.

We assume that coefficient digits are uncompressed; i.e. each binary coded decimal (BCD) is coded with four bits and, as a result, the representation of a 3-digit BCD number needs 12 bits. This means that DPD encoding is neglected in our model. This is obviously not a problem, since DPD encoding is a simple AND/OR logic and could be easily verified.

4.1. Exponent modeling

For the modeling exponent, we note that the base of both arithmetic and calculation is 10 and the coefficient is multiplied by powers of 10. Normally decision diagrams work on a binary basis, but here we require a 10-ary basis. This is a must because we should model expressions like $10^{Exponent}$.

The decomposition rule for exponent modeling is Shannon decomposition, since it simplifies our exponent model. Keep in mind that in contrast to traditional binary decision diagrams, here Shannon decomposition is applied on a 10 basis, not a binary basis.

In Figure 2, ec_0 is bit 0 of the exponent, and so on. The value of each node is displayed close to it. Notice how edge weights are calculated based on powers of 10. Each bit of the exponent is modeled by its counterpart node in our data structure. A node is connected to other nodes by means of two edges. The real weight of an edge equals 10 to the power of the shown weight; for example, if an edge is shown with weight of 2 (ec_1 in Figure 2), then the real weight of that edge equals 10^2. Continuing node evaluation, the exponent value can be derived as follows.

Figure 2 illustrates how constituting parts of the exponent can be calculated at each level. Generalizing the procedure to other levels, it is possible to calculate the ith part of the exponent at level i as shown in Eq. (2).

$$10^{Exponent}\, at\, level\, i = (1+9ec_0)(1+99ec_1)(1+999ec_2)...(1+(10^i-1)ec_{i-1}) \tag{2}$$

In 64-bit format the exponent is 10 bits long. According to Eq. (2) and Figure 2, it is possible to derive the exponent value at the top node of the exponent model. This is done in Eq. (3):

$$\prod_{i=1}^{10}(1+(10^i-1)ec_{i-1}) \tag{3}$$

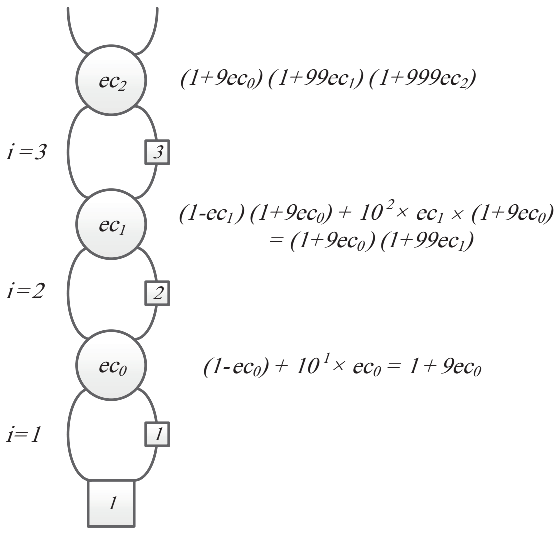
$(1+9ec_0)(1+99ec_1)(1+999ec_2)$
$(1-ec_1)(1+9ec_0) + 10^2 \times ec_1 \times (1+9ec_0) = (1+9ec_0)(1+99ec_1)$
$(1-ec_0) + 10^1 \times ec_0 = 1 + 9ec_0$
ec_2
ec_1
ec_0
i = 3
i=2
i=1
3
2
1
1

Figure 2. Exponent modeling using 10-based Shannon decomposition.

In order to model a biased exponent, we impose an appropriate weight on the top exponent node. Like before, the weights of edges in our exponent modeling are powers of 10; thus, 10^{-Bias} can be modeled with an edge whose weight is $-Bias$. According to Eq. (1) and the fact that $10^{Exponent-Bias} = 10^{Exponent} \times 10^{-Bias}$, Figure 3 illustrates how to incorporate bias into exponent modeling.

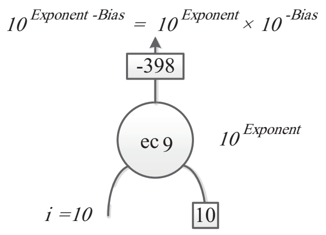

Figure 3. Exponent modeling incorporating bias.

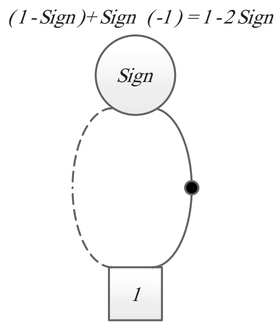

Figure 4. Sign modeling.

4.2. Sign modeling

The sign is modeled with a node with Shannon decomposition. A negative sign should be somehow inserted into the diagram. This is accomplished with an edge with weight of -1. This negative weight is displayed with a dot on the edge in our diagrams. Figure 4 shows further illustration.

4.3. Coefficient modeling

In DFP standard coefficients consist of BCD digits. These digits are compressed, but as previously described, we assume that they are not compressed; thus, four bits are assumed for each BCD digit. Without losing the generality, here we focus on 64-bit format. In the case of 32-bit or 128-bit format similar data structures can be derived. The only difference is in the size of the data structure, i.e. the number of nodes.

In 64-bit format there are 16 BCD digits constructing the coefficient. The coefficient is derived from Eq. (4).

$$Coefficient = d_0 + 10^1 d_1 + 10^2 d_2 + ... + 10^{15} d_{15} = \sum_{i=0}^{15} 10^i d_i \tag{4}$$

Since four bits are used for each BCD, we have Eqs. (5) and (6).

$$Coefficient = [cc_0 + 2cc_1 + 4cc_2 + 8cc_3] + 10^1[cc_0 + 2cc_1 + 4cc_2 + 8cc_3] + ... \tag{5}$$

$$Coefficient = \sum_{i=0}^{15} 10^i [cc_{4i+0} + 2cc_{4i+1} + 4cc_{4i+2} + 8cc_{4i+3}] \tag{6}$$

According to Eq. (6) it is evident that two parameters are key parameters in constituting the coefficient: 1- The i index, which generates powers of 10 and is related to each BCD level. 2- Powers of two, which are multiplied by coefficient bits. Thus, our model should cover both powers. One such node is depicted in Figure 5. In Figure 5 both the node and edge are weighted. Nodes similar to that of Figure 5 will be used to model the coefficient. Each node like in Figure 5 has an i index that determines the power of 10. Edges have a j index, which determines the power of two. The top node value is calculated as shown in Figure 5. The decomposition rule here is positive Davio, because we can directly derive our intended value without Shannon complementation.

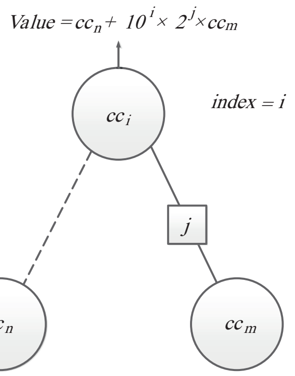

Figure 5. Coefficient modeling for a single node.

Based on our discussion and Figure 5, the coefficient model can be sketched as illustrated in Figure 6. It is easily verified that the top node value is that of Eq. (6). Remember that BCD digits are assumed to be uncompressed. Also keep in mind that the decomposition rule is positive Davio in coefficient modeling. Numbers close to each node in Figure 6 show powers of 10.

4.4. DFP number comprehensive model

We covered separate modeling of sign, coefficient, and exponents in the previous subsections. Now it is time to integrate them and render a comprehensive diagram that models all these aspects.

Both nodes and edges are weighted in this model. Coefficient bits start with *cc* and exponent ones start with *ec*. Both Shannon and positive Davio decomposition rules are utilized appropriately. The model is shown in Figure 7. Note the following:

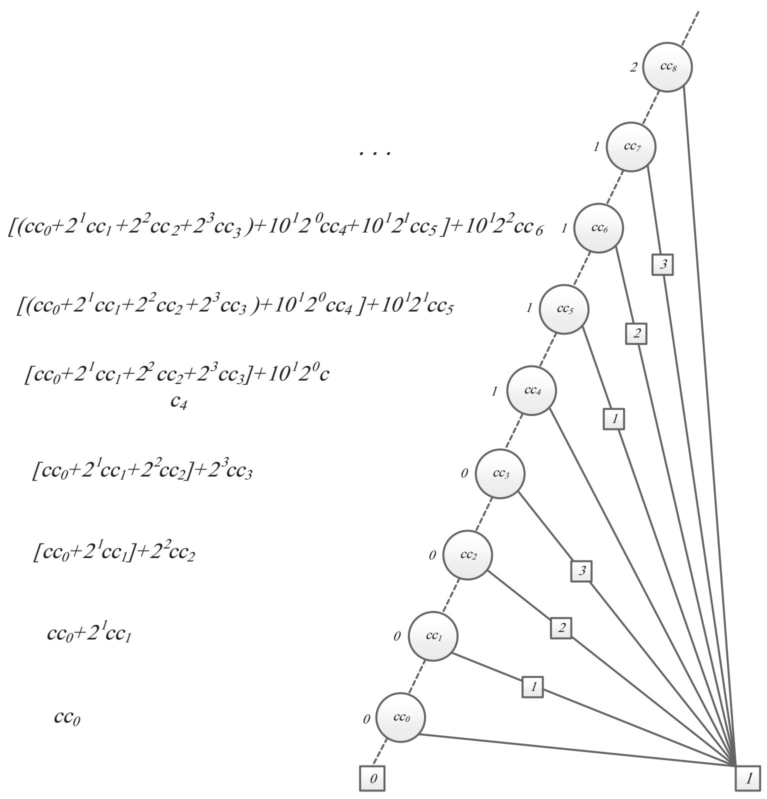
. . .
$[(cc_0+2^1cc_1+2^2cc_2+2^3cc_3)+10^12^0cc_4+10^12^1cc_5]+10^12^2cc_6$
$[(cc_0+2^1cc_1+2^2cc_2+2^3cc_3)+10^12^0cc_4]+10^12^1cc_5$
$[cc_0+2^1cc_1+2^2cc_2+2^3cc_3]+10^12^0cc_4$
$[cc_0+2^1cc_1+2^2cc_2]+2^3cc_3$
$[cc_0+2^1cc_1]+2^2cc_2$
$cc_0+2^1cc_1$
cc_0
cc_8
cc_7
cc_6
cc_5
cc_4
cc_3
cc_2
cc_1
cc_0
0
1

Figure 6. Coefficient modeling using positive Davio decomposition.

- Nodes with Shannon decomposition are sketched with bold circles.
- Positive Davio nodes are displayed with circles that are not bold.
- A thick line represents an edge with power of 10.
- A thin line represents an edge with a power of two.
- Negative weighted edge is depicted with a dot on that edge.

The computed value at some important nodes is shown for simplicity. In the next section we will focus on modeling canonicity.

5. Model canonicity

In the following discussions we assume that variable ordering is according to Eq. (7). Variable ordering becomes more important when we discuss model canonicity.

$$VariableOrdering: Sign < ec_9 < ... < ec_0 < cc_63 < ... < cc_0 \tag{7}$$

In this section we show that our proposed model is canonical. Note that canonicity of DFP modeling is basically different and more complex compared to traditional decision diagrams. Traditional decision diagrams represent a Boolean/integer function with Boolean/integer domain.

The model presented for a DFP number in Figure 7 is a canonical model because:

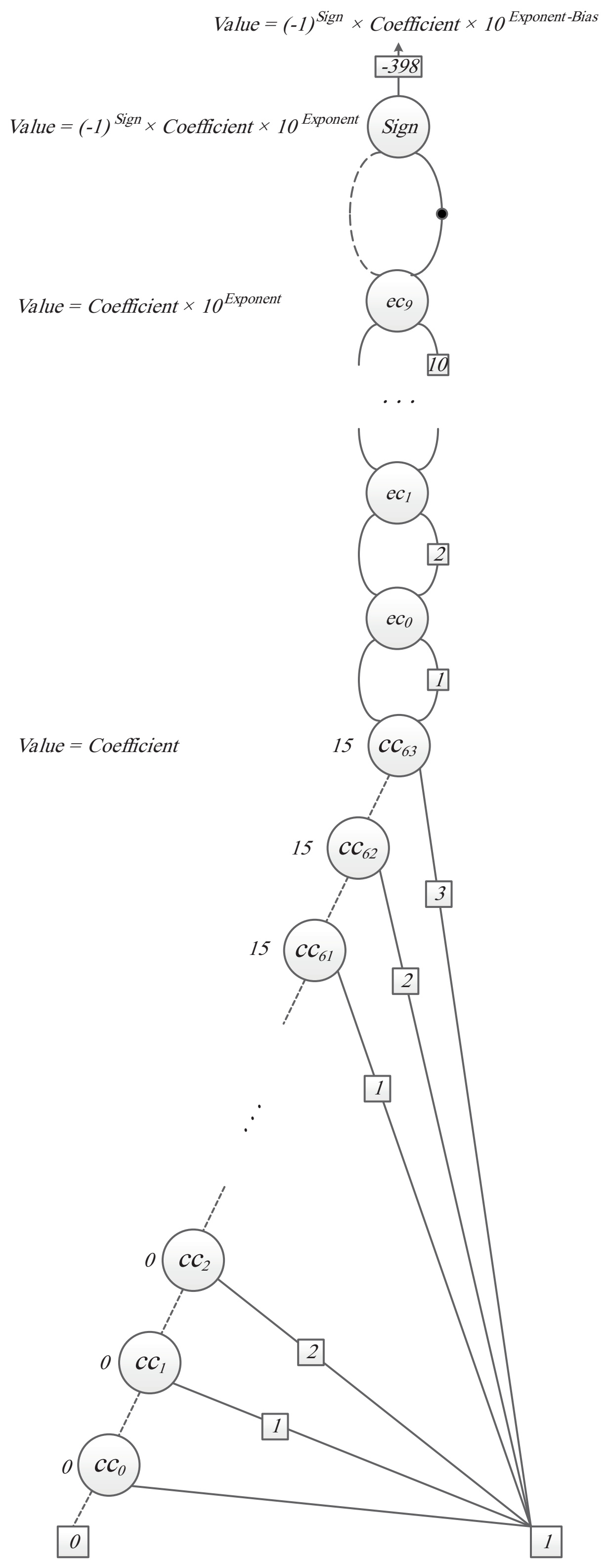
Value = (-1)^Sign × Coefficient × 10^Exponent-Bias
-398
Value = (-1)^Sign × Coefficient × 10^Exponent
Sign
Value = Coefficient × 10^Exponent
ec9
10
ec1
2
ec0
1
Value = Coefficient
15
cc63
15
cc62
15
cc61
3
2
1
0
cc2
2
0
cc1
1
0
cc0
0
1

Figure 7. DFP number comprehensive model.

- According to IEEE-754-R there is a unique interpretation of bits in each format. Each bit has a determined meaning and the DFP number value is determined based on the determined meaning of bits according to Eq. (1). On the other hand, there is always one and only one node related to each bit in each format. This means that the number of nodes equals the number of bits plus two (for terminal nodes 0 and 1) in Figure 7.

- There is a fixed variable ordering based on which our DFP model is constructed; thus, our model is an ordered model. This variable ordering was reported in Eq. (7).

- Edges and weights in the model determine the relation between nodes. This relation is fixed and defined based on node and edge weights at each level. If another relation replaces the defined relation, by changing weight of an edge or a node for instance, then according to Eqs. (2), (3), (4), (5), and (6) a different value is calculated, and as a result a different number is calculated. We conclude that by changing weights of each edge or node we cannot derive an identical value before the change.

- Putting it all together, assuming the previously defined variable ordering, there are fixed numbers of nodes and edges in the model of a DFP number. The relation between nodes is also determined and fixed. Changing either an edge connection or weight, and also changing either the existence or weight of a node and changing the decomposition rule of a node, will result in a different value; thus, there will be no two isomorph diagrams representing the same DFP number. Thus, our DFP number model is canonical.

6. Model extraction and verification

Verification of arithmetic circuits can be performed in two phases: 1- deriving a mathematical data structure appropriate for arithmetic operations, and then 2- extracting a mathematical model from the circuit to be compared with the model of phase 1. In this paper we present a method that models arithmetic operations using the DFP data structure efficiently. Step 1 of verification is performed using the DFP data structure of this paper. Extracting a similar model from a circuit for comparison with our model requires more research, which is our next step. We are currently developing a model for model extraction from hardware. That is our next step toward a complete formal verification methodology. Due to its complexity and variety it is out of scope of this paper and will be covered in our next studies.

7. Experimental results

In this section we provide our implementation results regarding complexity and performance of the proposed DFP data structure. In order to experimentally show that the proposed DFP data structure grows linearly for a DFP number, we measure the memory consumption of our data structure. We also report the run time of our modeling implementation, which supports the proven theorems in the preceding sections. The proposed model was implemented in C++. All experiments have been run on a PC with Core2Duo CPU, operating at 2.4 GHz, running Windows-7 x64 OS with 4 GB of RAM.

Modeling a single DFP number requires $m+n+3$ nodes where m and n are the number of coefficient and exponent bits, respectively. There are $m+n$ nodes for the coefficient and exponent part, one node for the sign bit, and two leaves for 1 and 0. Our modeling supports all standard formats according to IEEE-754-R, i.e. 32-bit, 64-bit, and 128-bit format. For measuring memory usage, we generated sufficiently large sets of DFP number data structures and monitored memory usage. In Figure 8 memory usage for all three formats is shown for different numbers of DFP numbers. It is evident that memory consumption of DFP number modeling grows linearly. Experimental memory usage for a single DFP number is reported in Table 1.

Table 1. Experimental memory usage of single DFP number.

Format length	DFP number node count	DFP number memory usage
32-bit	39	3.42 kB
64-bit	77	6.70 kB
128-bit	153	13.25 kB

Further illustration of the linear growth of our data structure can be accomplished by relative memory usage of data structure implementation for different standard formats. Inspecting the numbers in Table 1 reveals that for sufficiently large numbers of DFP data structures (minimizing implementation side effects) memory consumption is doubled with doubling format length. For instance, when we build the 32-bit data structure we create (28+8+3) nodes. Creating the 64-bit version requires (64+10+3) nodes. Thus, memory usage of the 64-bit relative to the 32-bit data structure is $77/39 \approx 1.97$, which is very close to our experimental result in Table 1, which is in fact $6.70/3.42 \approx 1.96$. Relative memory requirements of 128-bit and 32-bit formats approach $13.25/3.42 \approx 3.87$, which is again close to $(136+14+3)/(28+8+3) \approx 3.92$ as expected.

The run times of corresponding implementations are illustrated in Figure 9. It is obvious that for 128-bit format a longer construction time is required. The time unit of the modeling implementation time is milliseconds. All the experiments run in a short and reasonable amount of time.

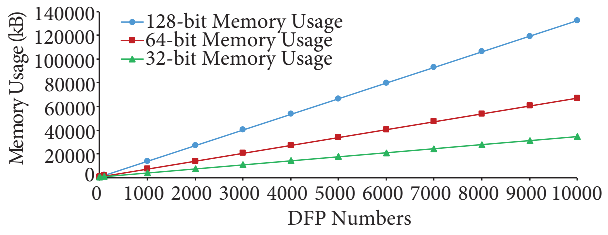

Figure 8. Single DFP number experiments: memory usage.

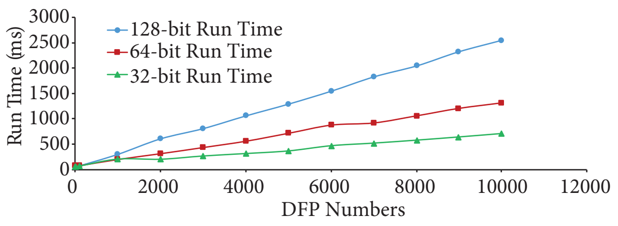

Figure 9. Single DFP number experiments: run time.

As described in the literature review, our presented model is the first that is capable of modeling DFP representation, so we cannot compare our memory usage or run time speed with previous works. Nevertheless our data structure can also model integer arithmetic. If we model integer multipliers with our DFP data structure, then we can compare its performance with other methods. In such a case some of the DFP model capabilities will be ignored, because there is no floating point number. However, we have still equal performance with *BMDs. In essence, our diagram and *BMD perform equally in terms of memory for integer multipliers. However, keep in mind that the DFP model can also represent floating point arithmetic, whereas *BMDs cannot. Thus, we have equal performance with better expressive power when compared to integer level diagrams. Comparisons with K*BMDs with Shannon decomposition are illustrated as well. It is obvious from Table 2 that our DFP diagram outperforms K*BMDs with the Shannon rule. K*BMDs with positive Davio expansion make no difference from *BMDs, so we do not mention numbers for that case.

Table 3 reports memory consumption of the DFP structure versus *BMD for several adders. In Table 3 memory usages for three adders are reported. It is evident that our data structure requires less memory than

*BMD. This is mainly due to negation and multiplicative powers of DFP diagram edges, which result in a more compact representation. This comes at the cost of more complex data structure and more processing time.

Table 2. Integer multiplier memory usage (number of nodes.

Circuit	DFP model	*BMD	K*BMD (Shannon)
Mult 16-bit	32	32	65551
Mult 32-bit	64	64	Blow-up
Mult 64-bit	128	128	Blow-up

Table 3. Integer adder memory usage comparison.

Circuit	DFP model	*BMD
Add 16-bit	0.37 MB	0.7 MB
Add 64-bit	0.93 MB	1.1 MB
Add 256-bit	5.23 MB	6.5 MB

8. Conclusion

In this paper we proposed a data structure that is able to model DFP numbers and DFP arithmetic operations. The proposed model made use of Shannon and positive Davio decomposition rules in order to increase expressiveness and size reduction of the model. We provided the model complexity for representing a DFP number. It was shown that the presented model grows linearly for a DFP number. Experimental results support our claim. It is obvious that our proposed data structure can be implemented in a short amount of time for DFP number modeling. The memory requirement for this modeling is also negligible, in the order of several kilobytes. It is evident that our experimental results agree with the provided mathematical discussions on the linear growth when format size is increased. We also showed that our model is canonical, so it can be used for formal verification purposes efficiently. In order to formally verify a DFP DSP system, a model extraction from a DFP arithmetic hardware is necessary. Then the extracted model and the proposed model can be checked for design errors. This is out of scope of this paper and will be our next step.

References

[1] Price D. Pentium FDIV flaw-lessons lLearned. IEEE Micro 1995; 15: 86-88.

[2] Sharangpani HP, Barton ML. Statistical Analysis of Floating Point Flaw in the Pentium Processor. Santa Clara, CA, USA: Intel Technical Report, 1994.

[3] IEEE Computer Society. IEEE Std 754-1985: IEEE Standard for Binary Floating-Point Arithmetic. New York, NY, USA: IEEE, 1985.

[4] IEEE Computer Society. IEEE Std 754-2000: IEEE Standard for Floating-Point Arithmetic. New York, NY, USA: IEEE, 2008.

[5] Duale AY, Decker MH, Zipperer HG, Aharoni M, Bohizic TJ. Decimal floating-point in z9: an implementation and testing perspective. IBM J Res Dev 2007; 51: 217-227.

[6] Lipetz D, Schwarz E. Self checking in current floating-point units. In: IEEE 2011 20th Symposium on Computer Arithmetic; 25–27 July 2011; Tubingen, Germany. New York, NY, USA: IEEE. pp. 73-76.

[7] Corena M. IEEE 754-2008 decimal floating-point for Intel architecture processors. In: IEEE 2009 19th Symposium on Computer Arithmetic; 8–10 June 2009; Portland, OR, USA. New York, NY, USA: IEEE. pp. 225-228.

[8] Gao S, Al-Khalili D, Langlois JMP, Chabini N. Decimal floating-point multiplier with binary-decimal compression based fixed-point multiplier. In: IEEE 2017 30th Canadian Conference on Electrical and Computer Engineering; 30 April–3 May 2017; Windsor, Canada. New York, NY, USA: IEEE. pp. 1-6.

[9] Ray S. Scalable Techniques for Formal Verification. Berlin, Germany: Springer, 2010.

[10] Camilleri A, Gordon M, Melham T. Hardware Verification Using Higher-Order Logic. Technical Report UCAM-CL-TR-91. Cambridge, UK: University of Cambridge, 1986.

[11] Bryant RE. Graph-based algorithms for Boolean function manipulation. IEEE T Comput 1986; C-35: 677–691.

[12] Fujita M, McGeer PC, Yang JCY. Multi-terminal binary decision diagrams: an efficient data structure for matrix representation. Form Method Syst Des 1997; 10: 149-169.

[13] Drechsler R, Theobald M, Becker B. Fast OFDD-based minimization of fixed polarity Reed-Muller expressions. IEEE T Comput 1994; 45: 1294-1299.

[14] Kebschull U, Schubert E, Rosenstiel W. Multilevel logic synthesis based on functional decision diagrams. In: Proceedings of the European Conference on Design Automation; 16–19 March 1992; Brussels, Belgium. New York, NY, USA: IEEE. pp. 43-47.

[15] Drechsler R, Sarabi A, Theobald M, Becker B, Perkowski MA. Efficient representation and manipulation of switching functions based on ordered Kronecker functional decision diagrams. In: Proceedings of 31st Design Automation Conference; 6–10 June 1994; San Diego, CA, USA. New York, NY, USA: IEEE. pp. 415-419.

[16] Drechsler R, Becker B. Ordered Kronecker functional decision diagrams, a data structure for representation and manipulation of Boolean functions. IEEE T Comput Aid D 1998; 17: 965-973.

[17] Lai YT, Sastry S. Edge-valued binary decision for multi-level hierarchical verification. In: Proceedings of 29th ACM/IEEE Design Automation Conference; 8–12 June 1992; Anaheim, CA, USA. New York, NY, USA: IEEE. pp. 608-613.

[18] Lai YT, Pedram M, Vrudhula SBK. Formal verification using edge-valued binary decision diagrams. IEEE T Comput 1996; 45: 247-255.

[19] Bryant RE, Chen YA. Verification of arithmetic circuits with binary moment diagrams. In: 32nd Design Automation Conference; 12–16 June 1995; San Francisco, CA, USA. New York, NY, USA: IEEE. pp. 535-541.

[20] Bryant RE, Chen YA. Verification of Arithmetic Functions with Binary Moment Diagrams. Technical Report CMU-CS-94-160. Pittsburgh, PA, USA: Carnegie Mellon University, 1994.

[21] Drechsler R, Becker B, Ruppertz S. K*BMDs: a new data structure for verification. In: Proceedings of European Design and Test Conference; 11–14 March 1996; Paris, France. New York, NY, USA: IEEE. pp. 2–8.

[22] Clarke EM, Fujita M, Zhao X. Hybrid decision diagrams, overcoming the limitations of MTBDDs and BMDs. In: Proceedings of IEEE International Conference on Computer-Aided Design; 5–9 November 1995; San Jose, CA, USA. New York, NY, USA: IEEE. pp. 159-163.

[23] Ciesielski M, Kalla P, Zhihong Z, Rouzeyre B. Taylor expansion diagrams: a new representation for RTL verification. In: IEEE 6th International High-Level Design Validation and Test Workshop; 9 November 2001; Monterey, CA, USA. New York, NY, USA: IEEE. pp. 70-75.

[24] Ciesielski M, Kalla P, Askar S. Taylor expansion diagrams: a canonical representation for verification of data flow designs. IEEE T Comput 2006; 55: 1188–1201.

[25] Lotfi-Kamran P, Massoumi M, Mirzaei M, Navabi Z. Enhanced TED: a new data structure for RTL verification. In: 21st International Conference on VLSI Design; 4–8 January 2008; Hyderabad, India. pp. 481-486.

[26] Lotfi-Kamran P, Hosseinabady M, Shojaei H, Massoumi M, Navabi Z. TED+: A data structure for microprocessor verification. In: Proceedings of Asia and South Pacific Design Automation Conference; 21 January 2005; Shanghai, China. New York, NY, USA: IEEE. pp. 567-572.

[27] Alizadeh B, Fujita M. Modular datapath optimization and verification based on modular-HED. IEEE T Comput Aid D 2010; 29: 1422–1435.

[28] Alizadeh B, Fujita M. Modular equivalence verification of polynomial datapaths with multiple word-length operands. In: IEEE 2011 International High Level Design Validation and Test Workshop; 9–11 November 2011; Napa Valley, CA, USA. New York, NY, USA: IEEE. pp. 22–40.

[29] Chen YA, Bryant RE. *PHDD: An efficient graph representation for floating point circuit verification. In: Proceedings of IEEE International Conference on Computer Aided Design; 9–13 November 1997; San Jose, CA, USA. New York, NY, USA: IEEE. pp. 2-7.

[30] Chen YA, Bryant RE. An efficient graph representation for arithmetic circuit verification. IEEE T Comput Aid D 2001; 20: 1443-1454.

[31] Sayed-Ahmed AAR, Fahmy HAH, Kuhne U. Verification of the decimal floating-point square root operation. In: IEEE 2014 19th European Test Symposium; 26–30 May 2014; Paderborn, Germany. New York, NY, USA: IEEE. pp. 1-2.

[32] Sayed-Ahmed AAR, Fahmy HAH, Samy R. Verification of the decimal floating-point fused-multiply-add operation. In: 9th IEEE/ACS International Conference on Computer Systems and Applications; 27–30 December 2011: Sharm El-Sheikh, Egypt. New York, NY, USA: IEEE. pp. 255-262.

[33] Sayed-Ahmed AAR, Fahmy HAH, Hassan MY. Three engines to solve verification constraints of decimal Floating-Point operation. In: Conference Record of the 44th Asilomar Conference on Signals, Systems and Computers; 7–10 November 2010; Pacific Grove, CA, USA. New York, NY, USA: IEEE. pp. 1153-1157.