Formal Hardware Verification based on Signal Correlation Properties

A PVS LIBRARY FOR REDUNDANT NUMBER REPRESENTATIONS

Nikhil Kikkeri and Peter-Michael Seidel
Southern Methodist University
Computer Science and Engineering
Dallas, TX 75205
{nikhil,seidel}@engr.smu.edu

Abstract

This paper deals with special challenges in the formal verification of high-performance circuits that involve redundant number representations and partial compressions. Our particular focus is the verification of circuits where symbolic consideration of number representations is not sufficient, but where additionally properties beyond operand values need to be considered to show correct operation. As a solution we consider theorem proving techniques in PVS and we propose a framework that allows for reasoning about signal correlation properties in redundant representations. We develop a library for redundant representations that includes support of basic circuits for partial compression and that verifies several fundamental properties regarding signal correlation and partial compression in the computed results. We outline the applicability of our library in the verification of several practical circuits.

1. Introduction

The use of redundant number representations is crucial in the design and implementation of many high-performance circuits. Their flexibility in representing a particular value can enable design options that keep computations local and that allow to significantly shorten critical paths (e.g. carry-save adders for binary addition). On the other hand some number properties (e.g. signs, leading zeros) are difficult to determine directly from some redundant number representations (e.g. borrow-save representations) and require compression of the redundant representation before they can be determined. Instead of full compressions (e.g. carry-propagate additions), partial compressions (e.g. half-adders, P- or N-recoders [5]) with much faster implementations are often sufficient to determine a particular property [6], which is why several practical high-performance implementations follow this approach [14, 9, 15].

Showing the correctness of implementations that involve redundant number representations and partial compressions requires keeping track of more information of an operand than just the value represented. On the other hand abstraction from the bit patterns of a particular number representation is necessary to cope with verification complexities. The combination of these two arguments imposes a major challenge for conventional approaches in formal verification of arithmetic hardware.

In [5] the concept of the *fraction range* of a redundant number representation has been introduced to allow capturing and reasoning on properties of redundant number representations beyond their values while still abstracting from the choice of the number itself. This concept has been used so show the correctness of significant parts of the implementations in [9, 15]. The correctness proofs on the leading zero prediction from [9] are cleaner, more elegant and easier to follow than the informal explanations for the functionality of comparable circuits in [13] (and the references that appear there), but they are also not fully formalized and their translation into a form that can be automatically checked imposes several challenges.

We focus on formal verification based on theorem proving and choose PVS [10, 16] as the implementation environment. PVS has been widely used for hardware verification [11], e.g. in verifying the VAMP microprocessor [1]. The PVS bitvectors library [4] is a fundamental tool for the verification of arithmetic circuits in PVS and the library for basic circuits from [2] can be of further assistance. We develop a library that extends the supported number representations from binary in [4] to the most popular redundant number representations of carry-save and borrow-save numbers [5], and that introduces concepts similar to that of fraction ranges from [5] to capture redundant properties of these numbers in an abstract way. We specify basic circuits

for the manipulation and partial compression of redundant number representations in PVS and formally show their effect on value and redundant features of the redundant inputs. Our library can assist in showing the correctness of high-performance circuits that make use of redundant number representations and partial compressions. We outline the applicability of our library in the verification of several practical cases.

In Section 2 we overview background information, we formally introduce the redundant number representations under consideration and show some of their basic properties. In Section 3 we introduce basic circuits for partial compression and we formulate and verify their partial compression properties on redundant number representations. In Section 4 we describe some details of the library implementation in PVS. In Section 5 we discuss the application of the proposed PVS library to the verification of practical implementations and finally conclude our work.

2 Number Representations

2.1 Notation

We denote binary strings in upper case letters (e.g. X,Y,Z). The value represented by a binary string is represented in italics (e.g. x, y, z). Let $\mathrm{X}_k\mathrm{X}_{k-1}\cdots\mathrm{X}_j \in \{0,1\}^{k-j+1}$ denote a binary string, also written as $\mathrm{X}[k:j]$. The weight associated with the bit X_i is 2^i. The binary value of a binary string is denoted by: $\langle \mathrm{X}[k:j]\rangle = \sum_{i=j}^{k} \mathrm{X}[i]\cdot 2^i$. The two's complement value of a binary string is denoted by: $[\mathrm{X}[k:j]] = -\mathrm{X}[k]\cdot 2^k + \sum_{i=j}^{k-1} \mathrm{X}[i]\cdot 2^i$.

2.2 Redundant number representations

Redundant number representations introduce multiple representations for each value represented (for a survey see for example [12]). The given flexibility can allow for optimizations in circuit design.

The most popular redundant number representations are redundant carry-save and borrow-save representations [5]. A number representation $\mathbf{F}[k:j]$ in *carry-save* (CS) format is compounding two binary strings: a carry-string $\mathrm{C}[k:j]$ and a sum-string $\mathrm{S}[k:j]$:

$$\mathbf{F}[k:j] = \begin{pmatrix} \mathrm{C}[k:j] \\ \mathrm{S}[k:j] \end{pmatrix} = \begin{pmatrix} \mathrm{C}[k], \mathrm{C}[k-1], \ldots, \mathrm{C}[j] \\ \mathrm{S}[k], \mathrm{S}[k-1], \ldots, \mathrm{S}[j] \end{pmatrix}$$

representing the value:

$$\begin{aligned} \langle \mathbf{F}[k:j]\rangle_{CS} &= \sum_{i=j}^{k} \langle \mathbf{F}[i]\rangle_{CS}\cdot 2^i \\ &= \sum_{i=j}^{k} (\mathrm{C}[i]+\mathrm{S}[i])\cdot 2^i \\ &= \langle \mathrm{C}[k:j]\rangle + \langle \mathrm{S}[k:j]\rangle. \end{aligned}$$

Each single CS digit is represented by a carry- and by a sum-bit and has a value from the range $\{0,1,2\}$. Note, that an n-digit carry-save representation can represent all values from the range $\{0,\ldots,2^{n+1}-2\}$.

Carry-save representations are redundant number representations, because a particular value can have multiple representations. Therefore, by converting a carry-save number representation to its corresponding value, structural information about the redundancy and about particular bits in the CS representation are lost.

A number representation in *borrow-save* (BS) format $\mathbf{F}[k:j]$ compounds a positively weighted sum-string $\mathrm{S}[k:j]$ and a negatively weighted borrow-string $\mathrm{B}[k:j]$:

$$\mathbf{F}[k:j] = \begin{pmatrix} \mathrm{B}[k:j] \\ \mathrm{S}[k:j] \end{pmatrix} = \begin{pmatrix} \mathrm{B}[k], \mathrm{B}[k-1], \ldots, \mathrm{B}[j] \\ \mathrm{S}[k], \mathrm{S}[k-1], \ldots, \mathrm{S}[j] \end{pmatrix}$$

representing the value:

$$\begin{aligned} \langle \mathbf{F}\rangle_{BS} &= \sum_{i=j}^{k} \langle \mathbf{F}[i]\rangle_{BS}\cdot 2^i \\ &= \sum_{i=j}^{k} (\mathrm{S}[i]-\mathrm{B}[i])\cdot 2^i \\ &= \langle \mathrm{S}[i]\rangle - \langle \mathrm{B}[i]\rangle. \end{aligned}$$

Each single BS digit is represented by a borrow- and by a sum-bit and has a value from the range $\{-1,0,1\}$. An n-digit borrow-save representation can represent all values from the range $\{-2^n-1,\ldots,2^n-1\}$ and most values from this range have redundant representations.

Any BS representation $\mathbf{G}[k:j]$ can be easily converted to a CS representation $\mathbf{F}[k:j]$ of the same value by using the following correspondence:

$$\left\langle \begin{pmatrix} \mathrm{B}[k:j] \\ \mathrm{S}[k:j] \end{pmatrix} \right\rangle_{BS} = \left\langle \begin{pmatrix} \overline{\mathrm{B}[k:j]}, 1 \\ \mathrm{S}[k:j], 1 \end{pmatrix} \right\rangle_{CS}.$$

2.3 Basic Properties

In this section we are discussing basic properties of redundant carry-save and borrow-save representations. The values and value ranges have already been discussed in the previous section. Here we review the concept of fraction ranges from [5]. The consideration of fraction ranges allows the expression of gradual restrictions in the flexibility of redundant representations based on signal correlation properties. Informally, the fraction range of a set of carry-save representations is defined as the interval of fractions occurring within the numbers when a binary point is virtually shifted to any position within the number representations.

For a formal definition of fraction ranges, fractions of redundant representations need to be defined.

Definition 1 *For a carry-save representation* $\mathbf{F}[k:j]$ *and for all integers* t *with* $j \leq t \leq k$*, the fraction stating at position* t *is defined by:*

$$\begin{aligned} frac_t(\mathbf{F}[k:j]) &= \langle \mathbf{F}[t:j] \rangle_{CS} \cdot 2^{-t-1} \\ &= \sum_{i=j}^{t} \langle \mathbf{F}[i] \rangle_{CS} \cdot 2^{i-t-1} \\ &= \sum_{i=j}^{t} (\mathrm{C}[i] + \mathrm{S}[i]) \cdot 2^{i-t-1} \\ &= \langle \mathrm{C}[t:j] \rangle \cdot 2^{-t-1} + \langle \mathrm{S}[t:j] \rangle \cdot 2^{-t-1}. \end{aligned}$$

For borrow-save representations $\mathbf{F}[k:j]$*, fractions are defined correspondingly by replacing the evaluation function for carry-save numbers* $\langle \rangle_{CS}$ *with the evaluation function for borrow-save numbers* $\langle \rangle_{BS}$*.*

The fraction range is defined for a set of redundant number representations.

Definition 2 *For any set of* n*-digit redundant representations* Q *the fraction range* $FR(Q)$ *is defined as the smallest interval* $[a, b]$*, such that for all* t *with* $0 < t \leq n$ *and for all* $\mathbf{F}[n-1:0] \in Q : frac_t(\mathbf{F}[n-1:0]) \in [a, b]$*.*

Note, that by applying the definition of fractions for CS and BS numbers, the definition of fraction ranges applies to CS and BS numbers correspondingly.

It can be easily shown that for any set of carry-save numbers fraction range lies within the interval $[0, 2)$ and for any set of borrow-save numbers fraction range lies within the interval $(-1, 1)$.

3 Basic Circuits for Partial Compression

The conversion of a redundant number representation into a non-redundant (e.g. binary or two's complement) form is a conceptually slow operation involving long computation chains for carry- or borrow-propagations of at least logarithmic depth. For the extraction of particular features of the non-redundant form, the redundant representation does not necessarily have to be fully compressed. Constant-delay circuits implementing a partial compression are sufficient to allow for extracting various properties of a redundant representation.

Figure 1 gives an overview of the basic circuits for partial compression that we are considering. Each of these basic circuits is just used for a single digit position and applied concurrently to each digit of a redundant number representation. In their parallel application each of the basic circuits for partial compression meets two properties: (i) The value of the operand is preserved from input to output representation; and (ii) the redundancy of the representation is reduced.

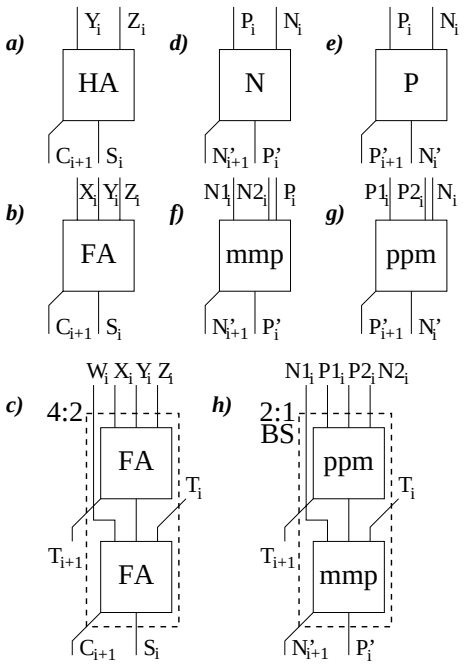

Figure 1. Basic Partial Compression Circuits

Value preservation properties (i) are used in the following for the specification of the functionality of each of the basic partial compression circuits.

Redundancy in the representations of input and output operands is measured considering fraction ranges. Their reduction is discussed to address the redundancy reduction property (ii).

3.1 Specification

Half-Adder (Fig. 1a). A *half-adder* has two input bits Y_i, Z_i and computes two output bits C_{i+1}, S_i, so that: $2C_{i+1} + S_i = Y_i + Z_i$. The logic functions for C_{i+1} and S_i can be specified as: $C_{i+1} = Y_i \wedge Z_i$, $S_i = Y_i \oplus Z_i$.

Full-Adder (Fig. 1b). A *full-adder* has three input bits X_i, Y_i, Z_i and computes two output bits C_{i+1}, S_i, so that: $2C_{i+1} + S_i = X_i + Y_i + Z_i$. The logic functions for C_{i+1} and S_i can be specified as: $C_{i+1} = (X_i + Y_i + Z_i \geq 2)$, $S_i = X_i \oplus Y_i \oplus Z_i$.

4:2 Adder (Fig. 1c). A *4:2-adder* can be built from two full-adders as depicted in Figure 1c). It has four primary input bits W_i, X_i, Y_i, Z_i and one intermediate carry-in T_i and computes one intermediate carry-out $T_{i+1} = (X_i + Y_i + Z_i \geq 2)$ and two primary output bits C_{i+1}, S_i, so that

$$2C_{i+1} + S = W_i + X_i + Y_i + Z_i + T_i - 2T_{i+1}.$$

N-Recoder (Fig. 1d). An *N-recoder* has two input bits P_i, N_i and computes two output bits N'_{i+1}, P'_i, so that: $P'_i - 2N'_{i+1} = P_i - N_i$. The logic functions for P'_i and N'_{i+1} can be specified as: $N'_{i+1} = N_i \wedge \overline{P_i}$, $P'_i = P_i \oplus N_i$.

P-Recoder (Fig. 1e). A *P-recoder* has two input bits P_i, N_i and computes two output bits N'_i, P'_{i+1}, so that: $2P'_{i+1} - N'_i = P_i - N_i$. The logic functions for P'_{i+1} and N'_i can be specified as: $P'_{i+1} = N_i \wedge \overline{P_i}$, $N'_i = P_i \oplus N_i$.

mmp-Recoder (Fig. 1f). An *mmp-recoder* has three input bits $P_i, N1_i, N2_i$ and computes two output bits N'_{i+1}, P'_i, so that: $P'_i - 2N'_{i+1} = P_i - N1_i - N2_i$.

ppm-Recoder (Fig. 1g). A *ppm-recoder* has three input bits $P1_i, P2_i, N_i$ and computes two output bits P'_{i+1}, N'_i, so that: $2P'_{i+1} - N'_i = P1_i + P2_i - N_i$.

2:1 BS-Adder (Fig. 1h). A *2:1-BS Adder* has two BS input digits $(P1_i, N1_i)$ and $(P2_i, N2_i)$ and a temporary carry-in T_i and computes one temporary carry-out $T_{i+1} = (P1_i + P2_i - N2_i \geq 1)$ and two primary output bits N'_{i+1}, P'_i, so that:

$$P'_i - 2N'_{i+1} = P1_i - N1_i + P2_i - N2_i + T_i - 2T_{i+1}.$$

3.2 Properties and Theorems

We are interested in the value preservation and partial compression properties of the above circuits.

Value Preservation Properties. Value Preservation from input to output operands is achieved for parallel application of basic partial compression circuits to whole operands and follows from the specification of each basic circuit for partial compression. Considering the example of an HA-line that operates on an n-bit carry-save operand

$$\mathbf{F}[\mathbf{n-1:0}] = \begin{pmatrix} \mathrm{Y}[n-1:0] \\ \mathrm{X}[n-1:0] \end{pmatrix}$$

with carry-string $\mathrm{Y}[n{-}1{:}0]$ and sum-string $\mathrm{X}[n{-}1{:}0]$ yields the carry-save representation

$$\mathbf{G}[\mathbf{n:0}] = HA(\mathbf{F}[\mathbf{n-1:0}]) = \begin{pmatrix} \mathrm{C}[n:1], 0 \\ 0, \mathrm{S}[n-1:0] \end{pmatrix}$$

with carry-string $(\mathrm{C}[n{:}1], 0)$ and sum-string $(0, \mathrm{S}[n{-}1{:}0])$ that has the same value:

$$\left\langle \begin{matrix} \mathrm{Y}[n-1:0] \\ \mathrm{X}[n-1:0] \end{matrix} \right\rangle_{CS} = \left\langle \begin{matrix} \mathrm{C}[n:1], 0 \\ 0, \mathrm{S}[n-1:0] \end{matrix} \right\rangle_{CS},$$

but is partially compressed (as we will see later). Similarly, we have for the operands in P-Recoding:

$$\left\langle \begin{matrix} \mathrm{N}[n-1:0] \\ \mathrm{P}[n-1:0] \end{matrix} \right\rangle_{BS} = \left\langle \begin{matrix} 0, \mathrm{N}'[n-1:0] \\ \mathrm{P}'[n:1], 0 \end{matrix} \right\rangle_{BS},$$

for N-Recoding:

$$\left\langle \begin{matrix} \mathrm{N}[n-1:0] \\ \mathrm{P}[n-1:0] \end{matrix} \right\rangle_{BS} = \left\langle \begin{matrix} \mathrm{N}'[n:1], 0 \\ 0, \mathrm{P}'[n-1:0] \end{matrix} \right\rangle_{BS},$$

and for the 2:1 BS-Adder operands:

$$\left\langle \begin{matrix} \mathrm{N1}[n-1:0] \\ \mathrm{P1}[n-1:0] \end{matrix} \right\rangle_{BS} + \left\langle \begin{matrix} \mathrm{N2}[n-1:0] \\ \mathrm{P2}[n-1:0] \end{matrix} \right\rangle_{BS} = \left\langle \begin{matrix} \mathrm{N}'[n:1], 0 \\ T[n], \mathrm{P}'[n-1:0] \end{matrix} \right\rangle_{BS}$$

In the following we will further extend the notation to apply a whole set of CS operands A instead of a single n-bit operand as the argument of a HA line, resulting in the image of the HA line considering the set of input operands:

$$HA(A) = \{\mathbf{G[n:0]} = HA(\mathbf{F[n-1:0]}) \mid \mathbf{F[n-1:0]} \in A\}.$$

This notation applies to describe the image of a set of BS operands considering P-recoder, N-recoder and 2:1 BS-adder lines correspondingly.

Partial Compression Properties. We overview the partial compression properties of the above circuits that can be described using the framework of fraction ranges from [7]. The theorems to state the corresponding reductions in the fraction ranges are listed in the following. An intuitive proof for most of the theorems can be found in [9].

Theorem 1 *For any set of CS representations R and any set of BS representations B:*

- *HA lines:*

  $$FR(Q) = [a, b] \Longrightarrow FR(HA(Q)) = [a/2, b/2 + 1/2].$$

- *P-, N-Recoder lines:*

  $$FR(R) = [c, d] \Longrightarrow FR(P(R)) = [c/2 - 1/2, d/2],$$
  $$FR(R) = [c, d] \Longrightarrow FR(N(R)) = [c/2, d/2 + 1/2].$$

- *2:1 BS-Adder lines:*
  $FR(R) = [c, d] \Longrightarrow$
  $FR(ADD2BS(R, R)) = [c/2 - 1/2, d/2 + 1/2].$

In the following we consider formalizing redundant number representations and the concept of fraction ranges in PVS with the ultimate goal to formally verify the above theorems on the fraction range reductions. Generally, fraction ranges indicate a measure of the correlation between bits in the number representations of a set of operands. With the above theorems the increase in the correlation between bits in output operand representations can be measured independent of the values considered.

4 Implementation and Formal Verification in PVS

The implementation of the proposed PVS library involves three parts: (i) defining redundant number representation types in PVS and defining value and projection functions for redundant number representations, (ii) defining signal correlation measures similar to fraction ranges and (iii) specifying the implementations of basic circuits for partial compression. Based on the implementation we describe and prove basic properties of redundant number representations and basic circuits partial compression in PVS.

4.1 Implementation in PVS

Redundant Number Representations in PVS. The PVS library for redundant number representations is based on the PVS *bitvectors* library [4]. The library makes use of bit vectors to specify the carry-save and borrow-save numbers. Each of these number representations is specified as a record with two fields. Hence in PVS carry-save and borrow-save representations are represented as a new types called CSVec(n) and BSVec(n):

$$\begin{aligned} CSVec(n) : TYPE &= [\#C : bvec[n], S : bvec[n]\#], \\ BSVec(n) : TYPE &= [\#B : bvec[n], S : bvec[n]\#], \end{aligned}$$

where C is the Carry part, B is the Borrow part and S is the Sum part of the carry-save and borrow-save vectors and each of these is a bitvector of length n. As in the case with binary representations, CS and BS representations also need to be converted to their values. Our library provides PVS expressions "CSVec2nat" and "BSVec2int" to achieve these conversions.

$$\begin{aligned} CSVec2nat(n : nat, csv : CSVec(n))\ nat\ = \\ bv2nat(csv`C) + bv2nat(csv`S) \\ BSVec2int(n : nat, bsv : BSVec(n))\ int\ = \\ bv2nat(bsv`C) - bv2nat(bsv`S) \end{aligned}$$

The library also provides functions for extracting projections of the CS and BS Vectors.

Fraction Ranges in PVS. The introduction of fraction ranges in PVS involved several challenges. Arithmetic other than on integers is not well supported in PVS.

Moreover, a fraction range is described as an interval that requires simultaneous consideration of its two delimiters, an upper and a lower bound which complicates formulation and manipulation in PVS.

For a simplified handling in PVS, we have considered upper and lower bound of the fraction range interval separately. Each of the two bounds is stated as a separate predicate on the redundant number representation argument and a boundary description. To avoid arithmetic on fractions, the values to be compared for predicate evaluation are scaled to integers. This is achieved by representing a fraction range boundary for a n-digit CS representation as a $n+1$-digit bit vector that is aligned with the CS representation at its least significant digit.

In the case of the upper boundary, $U\_Boundary?$ is the predicate which tests the n-digit carry-save number representation csv against the $n+1$-bit upper boundary vector $ubound$. In PVS the predicate is stated as follows:

U_Boundary(n,csv,ubound): bool =
IF FORALL(i:below(n)):
CSVec2nat(i+1,CSVProj(n,csv,i,0)) ≤ ubound ^ (n,n-i-1)
THEN TRUE
ELSE FALSE
ENDIF

The predicate for the lower boundary is defined correspondingly with replacing the $\leq$ condition with $\geq$.

Note, that the fraction range boundary pattern always has to be one bit longer than the CS argument that it is applied to (even if subranges are considered), and that the actual comparison with fragments of the number representations involve projections of both the boundary pattern and the CS representation.

Basic Circuits for Partial Compression in PVS. Specification of basic circuits are based on gate level implementations of the specifications from Section 3. The implementations for HA-lines and FA-lines are also considered in the basic circuits library in [2]. Based on a single basic HA/FA implementation, they are defined recursively for wider operands. We follow this approach and also initially define all partial compression circuit implementations for wider operands recursively from right to left. In the context of [2], this recursive definition is sufficient, because input and output operands are supposed to be accessed as a whole or directly converted into their values. Because the value functions from the bitvectors library are defined recursively in the same way, their evaluation does not cause any complications.

In our setting single bits or subranges of the input and output operand representation need to be extracted and individually processed. The above specification requires solving a recurrence equation for each of these tasks. We get around this complication by showing lemmas for the closed form representations of single bits of the circuit computations. These lemmas can then be used for compact substitution and resolution of many expressions.

4.2 Formal Verification in PVS.

The main properties that we have to verify in PVS are the value preservation of basic circuits from section 3.2 and the fraction range reductions from theorem 1.

The value preservation is easily shown based on the closed form description that we have derived for each parallel application of basic circuits. Each theorem on fraction range reductions is split into two versions: one for the upper fraction range boundary and one for the lower fraction range boundary.

As an example we consider the theorem on the upper boundaries for HA lines in the following. In PVS the theorem states as follows:

Theorem 2 *Let $n > 0$, input vector $csin \in CSVec(n)$, input upper fraction range boundary pattern $ubound \in CSVec(n+1)$, HA line output CS vector $csout = HA(csin) \in CSVec(n+1)$ and the modified upper fraction range boundary pattern of the output $mubound \in CSVec(n+2)$ being defined as*

mubound :=

ubound(n) o (NOT (ubound(n))) o ubound^(n-1,0),

where "o" is the concatenation operator and " ^ " is the bitvector extraction operator from the PVS bitvectors library. Then,

$$U\_Boundary?(n, csin, ubound) \Longrightarrow U\_Boundary?(n+1, csout, mubound),$$

Observe that in the above theorem the definition of $mubound$ calculates the modified upper fraction range boundary as $\langle ubound \rangle + 2^n$ in a bitwise fashion. Considering that the modified upper boundary pattern is wider by one bit and thus the reference binary radix point is shifted by one bit position to the left this corresponds to the computation of $b/2 + 1/2$ as modified upper bound in the first part of theorem 1.

Further three challenges in proving this type of theorem are given by: (i) the accessibility of closed form expressions that make local computations independent of prior bits in recursive definitions, (ii) the correspondence of properties of bit, CS or BS vector projections as arguments and as function values, (iii) the bit length alignment between carry-, borrow-, and sum-strings in recursive definitions and applications in the fraction range theorems.

The value-preservation lemmas for whole operands also had to be modified to match the case of projections of inputs and outputs without involving unrelated bits of the recursive definitions. In this contest also some additional lemmas regarding the concatenation and extraction of bitvectors were needed as these are not known to be available in the PVS bitvectors library. With these additional lemmas discharging the proof in PVS was a matter of using a couple of case-splits and about fifteen rewrite rules and the expansion of the predicate definition. The other parts of Theorem 1 have been proven correspondingly with a very similar proof structure.

More details on the library, its implementation and verification can be found on our website at [8].

5 Applications and Conclusions

To show the practicality of the proposed PVS library, we have applied the library in the verification of four different circuits that involve redundant number representations and/or partial compressions. We only give some details on the verification of one of the cases (leading zero prediction from redundant number representations (RNR)) and refer to [8] for more details on the other applications.

Leading Zero Prediction from RNR. We consider the implementation of approximately counting the number of leading zeros from BS representations in Nielsen *et al.* [9]. The method is based on P- and N-recoding of a borrow-save encoded string followed by a bitwise XOR-operation to a binary vector. The number of leading zeros in the binary vector almost equals the number of leading zeros in the binary representation of the absolute value of the number represented by the borrow-save encoded string. We summarize the main details of the application of this reduction below.

The input consists of a borrow-save encoded digit string $\mathbf{F}[\mathbf{1}:\text{–}\mathbf{52}]$. The borrow-save encoded string $\mathbf{F'}[\mathbf{2}:\text{–}\mathbf{52}] = P(N(\mathbf{F}[\mathbf{1}:\text{–}\mathbf{52}]))$ is computed and the borrow and sum vectors of the result are applied to a bitwise XOR-operation, where $P()$ and $N()$ denote P-recoding and N-recoding. The application of the technique is based on the following claim.

Theorem 3 *[9] Suppose the borrow-save encoded string $\mathbf{F'}[\mathbf{2}:\text{–}\mathbf{52}]$ is of the form $\mathbf{F'}[\mathbf{2}:\text{–}\mathbf{52}] = 0^k \cdot \sigma \cdot t[-1:-54+k]$, where $\cdot$ denotes concatenation of strings, 0^k denotes a block of k zeros, $\sigma \in \{-1, 1\}$, and $t \in \{-1, 0, 1\}^{54-k}$. Then the following holds:*

1. *If $\sigma = 1$, then the value represented by the borrow encoded string $\sigma.t$ satisfies:*

$$\sigma + \sum_{i=1}^{54-k} t[i] \cdot 2^{-i} \in (\frac{1}{4}, 1).$$

2. *If $\sigma = -1$, then the value represented by the borrow encoded string $\sigma.t$ satisfies:*

$$\sigma + \sum_{i=1}^{54-k} t[i] \cdot 2^{-i} \in (-\frac{3}{2}, -\frac{1}{2}).$$

The implication of Theorem 3 is that after PN-recoding, the number of leading zeros in the borrow-save encoded

string $\mathbf{F}'[-\mathbf{2} : \mathbf{53}]$ (denoted by k in the claim) can be used as an approximate normalization shift amount to bring the normalized result into one of two binades (i.e. in the positive case either $(\frac{1}{4}, \frac{1}{2})$ or $[\frac{1}{2}, 1)$, and in the negative case after negation either $(\frac{1}{2}, 1)$ or $[1, \frac{3}{2})$). This is used to prepare for the normalization shift in parallel with the significand addition in the N-path of the IEEE floating-point adder implementation in [15].

We formulate and verify the above theorem as an application of our library to demonstrate its use in the verification of circuit implementations that are of practical interest.

The other four types of circuits that we have considered as case studies for the verification with our library are: (i) a circuit to test a BS representation for zero, (ii) a circuit for the fast detection of the sign from signed redundant number representations like BS (signed sticky bit), (iii) implementations for Booth Recoding from redundant number representations following the description from [6].

The circuits specified in PVS can be synthesized to correct Verilog implementations by using the translation tools described in [3].

The use of redundant number representations and of partial compressions is far more popular than in the few case studies presented. For example the whole field of on-line arithmetic involves redundant representations and their partial compressions and could make use of our library for their verification. Moreover, implementations for division and square root computations on binary operands often use redundant representations and partial compressions for intermediate results.

We hope that our library will be applied and can also be found useful in the verification of several of these cases and improve the confidence in the corresponding designs.

Conceptually, the contribution in the proposed methods is the utilization of bit correlation properties and their combination with symbolic reasoning in theorem proving (in PVS). Bit correlation properties are captured by the pattern vectors for upper and lower bounds to represent fraction ranges. In the theorems these vectors can be applied and manipulated symbolically. The case study of the circuit for leading zero prediction from redundant number representations shows that the restriction of the (symbolic) boundary patterns can allow the determination of single bit conditions in the result with an accuracy up to one bit without any restrictions on the inputs of the computations (neither operands nor boundary patterns at the input). In the verification of the case studies this has been shown to be a powerful concept that might be applicable to other cases where symbolic reasoning should be combined with conditions on variable bits of the computation.

References

[1] C. Berg, S. Beyer, C. Jacobi, D. Kroening, and D. Leinenbach. Formal verification of the VAMP microprocessor. In *Proc. Symp. on the Effectiveness of Logic in Coputer Science (ELICS02)*, pages 31–36, 2002.

[2] C. Berg, C. Jacobi, and D. Kroening. Formal verification of a basic circuits library. In *Proc. of the IASTED International Conference on Applied Informatics, Innsbruck (AI 2001)*. ACTA Press, 2001.

[3] S. Beyer, C. Jacobi, D. Kroening, and D. Leinenbach. Correct hardware by synthesis from PVS. Internal Report, Saarland University, accessible from: http://busserver.cs.uni-sb.de/publikationen/BJKL02.pdf, 2002.

[4] R. Butler, P. Miner, M. Srivas, and D. Greve. A new bitvectors library for PVS. Technical Report TM-110274, NASA Langley Research Center, 1996.

[5] M. Daumas and D. Matula. Recoders for partial compression and rounding. Technical Report 97-01, Laboratoire de l'Informatique du Parallélisme, Lyon, France, 1997.

[6] M. Daumas and D. Matula. A Booth multiplier accepting both a redundant or a non-redundant input with no additional delay. In *IEEE International Conference on Application-specific Systems, Architectures and Processors*, pages 205–214, 2000.

[7] M. Daumas and D. Matula. Further reducing the redundancy of a notation over a minimally redundant digit set. *Journal of VLSI Signal Processing*, 33:7–18, 2003.

[8] N. Kikkeri and P.-M. Seidel. A PVS library for Redundant Number Representations and Partial Compressions. http://engr.smu.edu/˜ seidel/verification/correlation/.

[9] A. Nielsen, D. Matula, G. Even, and C. Lyu. An IEEE compliant floating-point adder that conforms with the pipelined packet-forwarding paradigm. *IEEE Transactions on Computers*, 49(1):33–47, January 2000.

[10] S. Owre, J. M. Rushby, and N. Shankar. PVS: A prototype verification system. In *11th International Conference on Automated Deduction (CADE)*, volume 607 of *LNAI*, pages 748–752. Springer, 1992.

[11] S. Owre, J. M. Rushby, N. Shankar, and M. K. Srivas. A tutorial on using PVS for hardware verification. In *Theorem Provers in Circuit Design (TPCD '94)*, volume 901 of *LNCS*, pages 258–279. Springer, 1994.

[12] B. Parhami. *Computer Arithmetic: Algorithms and Hardware Designs*. Oxford, 2000.

[13] M. Schmookler and K. Nowka. Leading zero anticipation and detection-a comparison of methods. In *Proceedings 15th IEEE Symposium on Computer Arithmetic*, pages 7 – 12, 2001.

[14] P.-M. Seidel. High-speed redundant reciprocal approximation. *INTEGRATION, the VLSI Journal*, 28:1–12, 1999.

[15] P.-M. Seidel and G. Even. Delay-optimized implementation of IEEE floating-point addition. *IEEE Transactions on Computers*, 53(2):97–113, 2004.

[16] N. Shankar. Specification and verification using PVS, April 1992.