Noname manuscript No.
(will be inserted by the editor)

Bounded Model Checking for Fixed-Point Digital Filters

Renato B. Abreu · Mikhail Y. R. Gadelha · Lucas C. Cordeiro · Eddie B. de Lima Filho · Waldir S. da Silva Jr

Received: date / Accepted: date

Abstract Currently, digital filters are employed in a wide range of signal processing applications, using fixed- and floating-point processors. Regarding the former, some filter implementations may be highly prone to errors, due to problems related to finite word-length. In particular, signal processing modules may produce overflows and unwanted noise, which are caused by quantization and round-off effects, during accumulative-addition and multiplication operations. As a consequence, the system output may overflow or even keep oscillating, which compromises the expected system performance. The present paper addresses this problem and proposes a new methodology for verifying digital filters, called Digital Systems Verifier, which is based on state-of-the-art bounded model checkers that support full C and employ solvers for boolean satisfiability and satisfiability modulo theories. In addition to verifying overflow and limit-cycle occurrences, the present approach can also check output errors and time constraints, based on discrete-time models implemented in C. Experiments conducted during this work show that the proposed methodology is effective, when finding realistic design errors with respect to fixed-point implementations of digital filters. Going further than previous attempts, the present results suggest that the proposed method, in addition to helping de-

R. Abreu
Nokia Institute of Technology, Manaus, Amazonas, Brazil.
E-mail: renato.abreu@indt.org.br

M. Gadelha
University of Southampton, Southampton, UK.
E-mail: myrg1g14@soton.ac.uk

L. Cordeiro and W. Júnior
Federal University of Amazonas, Manaus, Amazonas, Brazil.
E-mail: lucascordeiro@ufam.edu.br and E-mail: waldirjr@ufam.edu.br

E. Filho
Science, Technology and Innovation Center for the Industrial Pole of Manaus, Manaus, Amazonas, Brazil.
E-mail: eddie@ctpim.org.br

signers in determining the number of bits for fixed-point representations, can also aid in defining details about filter realization and structure.

Keywords Digital Filters · Finite Word-length · Formal Methods · Bounded Model Checking.

1 Introduction

In digital signal processing, a digital filter is a system that performs operations on discrete signals, in order to modify or improve some of their aspects. It can be classified into two types: infinite impulse response (IIR) or finite impulse response (FIR), differing by the presence or not of feedback, respectively. One of the most common procedures performed by digital filters, for instance, is to narrow or shape signal bandwidths, with the goal of discarding undesired information.

Digital filters have been used in a great variety of applications, mainly due to their reduced computational complexity and flexibility, which is reinforced by the availability of digital signal processor (DSP) and field programmable gate array (FPGA) devices. Regarding the first, they are split into two categories: fixed- and floating-point, which refer to the format used for storing and processing numerical-data representations. In fixed-point arithmetic, gaps between adjacent number representations are normally equal; floating-point arithmetic, in turn, results in nonuniform gaps, which are about ten million times smaller than a given number magnitude, depending on the floating point precision [1].

Typically, dynamic range, precision, ease of development, and cost considerations are used for determining the most interesting approach. For example, fixed-point devices provide low-cost products and normally incur higher development costs, which are suitable for high-volume applications; however, floating-point devices result in higher product costs, while presenting shorter development cycles and higher precision, which may be acceptable for high-value applications. In summary, the data to be processed and the target application define the need for fixed- or floating-point processors.

It is worth noticing that even with the current increased-availability of floating-point devices, which may also include optimized modules for complex-number manipulation, the high speed of fixed-point processors, in combination with their reduced cost, still make them a good choice for embedded digital filter design.

Both fixed- and floating-point implementations suffer from quantization errors; however, the latter normally presents better precision and higher dynamic range. Indeed, larger gaps in fixed-point approaches result in smaller signal-to-noise ratio (SNR) figures, and, due to that, nonlinearities, round-off errors, and overflows are much more pronounced, all caused by consecutive multiplication and addition operations using finite word-length, which might affect the desired filter behavior. For instance, regarding direct form realizations, only a small change on filter coefficients, due to quantization, has the potential to cause large changes in pole and zero locations [2].

In addition, different filter types present different problems. For example, IIR filters might suffer from serious oscillations in their outputs, even for zero input signals, which is a phenomenon known as *limit cycle* [3–6]. In summary, *limit cycles* are oscillations that appear due to rounding, truncation or overflow (nonlinearities), regarding internal filter computations, even if a given filter is stable, due to the feedback branch. FIR filters, in turn, do not suffer from such limit cycle effects, but might have other issues caused by finite word-length limitations (*e.g.*, frequency response modification).

There are many studies about quantization and limit cycle effects in digital filters, along with techniques to reduce their effects [7] or guarantee their absence [8–11]. Generally, limit cycles can be avoided by increasing the system word-length or applying scaling and saturation; however, those solutions present some drawbacks. The first may extrapolate the representation provided by the underlying architecture, and the second has the potential to decrease the SNR. As a consequence, the related error magnitude should then be verified, in order to ensure that it is at an acceptable level. It is also possible to use noise shaping [12], where quantization errors go through a feedback loop and, consequently, a very high SNR ratio is obtained, which greatly reduces limit cycle occurrences.

Another important property, which arises when implementing digital filters for real-time applications, is the time constraint [13]. In a real-time system, which may consist of other tasks beyond simple filtering, high sample rates will require more processing resources; therefore, the use of high sample rates may be insufficient to execute tasks and may also degrade the expected system performance. Indeed, time constraints verification is of paramount importance for real-time systems and, although not directly related to digital filters, it is often addressed during their design, in order to ensure the desired system behavior.

Normally, filter designers employ advanced tools for defining filter parameters, according to the desired operation in time and frequency domains, and use simulation software for validating their behavior, along with extensive testing. However, in most cases, floating-point arithmetic is considered during calculations, which can lead to wrong assumptions about filter performance.

There are a few tools for simulating systems using fixed-point arithmetic [14–16], which can be employed during filter design. As an example, Sung and Kum [17] proposed search algorithms to determine the minimum word-length bound, through a simulation-based approach. Nguyen, Menard, and Sentieys [18] also proposed faster word-length optimization algorithms, when compared to Sung and Kum's approach, which are based on iterative stochastic local searches. However, simulation (and testing) can lead to a limited number of scenarios and inputs, which normally do not exploit all possible behaviors that a system can exhibit and take long to complete. Analytical approaches may also be employed [19], which significantly reduces evaluation times. Hence, just performing frequency domain graphical analysis and simulations might not be enough to discover possible problems related to finite word-length, as well as filter time constraints.

Recently, Cox *et al.* [20,21] proposed the verification of fixed-point implementations of IIR digital filters, which is based on bounded model checking (BMC) [22] and applies modern satisfiability modulo theory (SMT) [23] solvers, in order to check for verification conditions. The main idea behind SMT-Based BMC approaches is to consider counterexamples of a particular length k and then generate an SMT formula, which is satisfiable if and only if such a counterexample exists [24,25].

1.1 Contributions

The present paper addresses the digital-filter verification problem and describes the use of a general purpose SMT-based bounded model checker for C programs, in order to verify potential problems caused by fixed-point arithmetic, on recursive filters.

Regarding the proposed methodology, in addition to detecting overflow and limit cycle issues, the following contributions can also be found: the associated processing time is considered during filter function unrolling, which checks the maximum acceptable time of filter operations, the output error magnitude is compared with a floating-point model, in order to ensure that the system error is within an acceptable margin, and BMC is exploited to verify the actual digital filter C code, which is intended to be embedded into micro-controllers and DSPs. It is worth noticing that the latter is closer to real implementations, where specific C constructs (*e.g.*, pointer arithmetic and comparisons) are used for implementing digital filters. The resulting C code also gives the possibility to apply other (state-of-the-art) model checking techniques (for C programs) to general filter verification.

Additionally, the application of SMT-based BMC approaches, regarding digital filter design, is not well known amongst DSP developers, which also opens a great opportunity, since SMT-based BMC approaches do not require proof (*i.e.*, they are algorithmic methods rather than deductive) and are very fast, if compared with other rigorous methods, such as theorem proving. Besides, one can easily specify a large number of digital filter properties and realization forms, in order to ensure correctness in computer-based systems (*e.g.*, DSP and FPGA).

This article is a substantially revised and extended version of the work published in previous conferences [26,27]. The major differences are the following: (i) in this version, an enhanced fixed-point library for arithmetic operations was implemented, in order to decrease verification-time figures, (ii) other filter structures in cascade, parallel, and transposed direct forms are supported, (iii) a maximum-error verification procedure was added, (iv) a unified tool, named as Digital Systems Verifier (DSVerifier[1]), which developed for checking

[1] Available at http://dsverifier.org/

different design problems in fixed-point digital filters, was built, (v) two different state-of-the-art BMC tools were used in DSVerifier as back-ends (*i.e.*, CBMC [28] and ESBMC [24]), and (vi) the performed experiments are based on a set of publicly available benchmarks. All benchmarks, tools, and results of the present evaluation are available on a supplementary web page[2].

The remainder of this paper is organized as follows. Section 2.1 gives a brief introduction about digital filter realizations, emphasizing some aspects related to implementation in fixed-point processors, along with state-of-the-art verification schemes presented in the related literature. Fundamentals about SMT-based BMC approaches, which were used in this work to the verification of digital filters, are tackled in section 2.2. Section 3 describes the research design and respective methodology, while section 4 presents the implemented methods to verify overflow, limit cycle, time constraint, and maximum error. Then, some experimental results for a set of digital filters, obtained with the proposed approach, are shown in section 5. Finally, section 6 presents the conclusions of this work, highlighting the importance of SMT-based BMC approaches for the verification of fixed-point filters.

2 Background and literature review

2.1 Verification of Digital Filters

As already mentioned, finite word-length effects are of paramount importance for digital filter design, given that numerical implementations of the related algorithms may suffer from deterioration in performance. The following sections will tackle filter implementation structures and some state-of-the-art verification algorithms, with the goal of introducing the basic problem.

2.1.1 Fixed-Point Filters Realization

Digital filters can be defined as linear time-invariant discrete-time systems, which are described by a difference equation as

$$y(n) = -\sum_{k=1}^{N} a_k y(n-k) + \sum_{k=0}^{M} b_k x(n-k), \tag{1}$$

where $y(n)$ is the output in instant n, $y(n-k)$ is the output k steps in the past, $x(n-k)$ is the input k steps in the past, a_k are the coefficients for the past outputs, b_k are the coefficients for the past inputs, N is the feedback filter order, and M is the feedforward filter order. The proper design of a digital filter consists in finding suitable values for coefficients a_k and b_k, which produce the expected frequency response.

Digital filters are usually classified according to their ideal frequency domain characteristics, whose basic forms are low-pass, high-pass, band-pass,

[2] http://dsverifier.org/

and band-reject. It is out of scope for this paper to show IIR and FIR filters design methods, because this is a huge topic that is covered in standard digital signal processing books [2], [29].

There are many ways to implement (1) in hardware, using FPGA and application specific integrated circuits, or in software, through a programmable digital computer, depending on the desired realization structure for the target system. The commonly known Direct Form I, Direct Form II, and Transposed Direct Form II structures are shown in Fig. 1, where z^{-1} is defined as the backward-shift operator, that is, a unit delay. These second-order structures are often employed as basic building blocks of high-order systems, in cascade and parallel [2].

When implementing fixed-point digital filters, coefficients and results of intermediate computations are affected by quantization and round-off errors. Here, the round-off quantizer $Q(x)$ was considered [2,29], whose maximum error caused by rounding is defined to be 2^{-b-1}, where b is the number of bits of the fractional part. If the result from an addition or multiplication exceeds the amount of bits available for the number representation, it is assumed that an *overflow* occurs. However, for limit cycle verification, overflows can naturally happen; as the two's complement arithmetic is adopted, results wrap around when overflow events occur. Fig. 2 shows the behavior of the round-off quantizer and the effect of a two's complement overflow wrapping around.

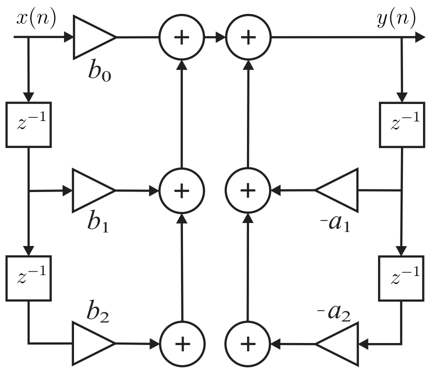
In order to obtain a realistic model of the finite precision system, each numeric value in the system is quantized, which includes inputs, coefficients, and results of arithmetic operations. Fig. 3 shows this approach for a single-pole filter.

Typically, numbers in fixed-point format are represented with a pair of digits, which are separated by a decimal point. The digits to the left and right represent the integer and fractional parts, respectively, and the two's complement is used to represent signed numbers, in fixed-point processors. In this system, the real number X, described by the $\langle k, l\rangle$ fixed-point position number $(b_{k-1}\ b_{k-2}\ ...\ b_1\ b_0\ \cdot\ b_{-1}\ b_{-2}\ ...\ b_{-l})$, can be represented as

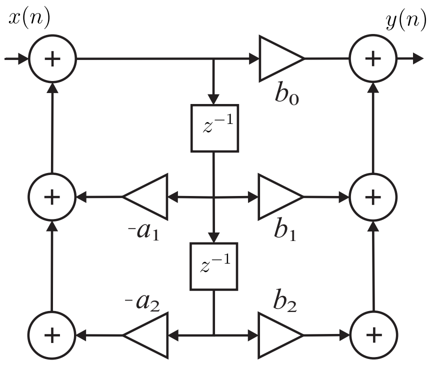
$$X = -b_{k-1}2^{k-1} + \sum_{i=k-2}^{-l} b_i 2^i. \tag{2}$$

The most significant bit $-b_{k-1}$ is used for the sign. Thus, the maximum representable value, which consists of an integer part with k bits and a fractional part with l bits, is $2^{k-1} - 2^{-l}$, and the minimum value is -2^{k-1}. In Fig. 3, the quantizer Q rounds numbers inside this range. If a number does not fit into this interval, then it indicates an *overflow*, which can be handled in the following ways, by the verification module: detect it as failure, wrap around results, or saturate to a maximum/minimum value. Although saturating implementations are popular in DSPs, we handle both saturating and wrap-around math.

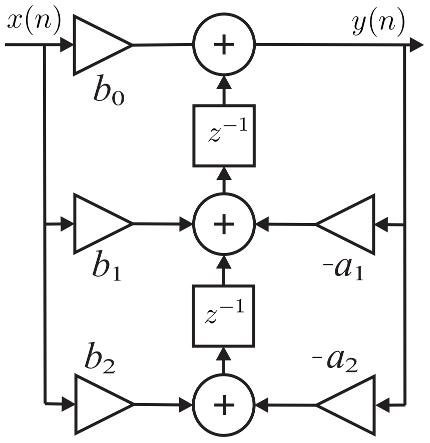
Given that fixed-point implementations introduce quantization effects and reduce dynamic range figures, traditional simulation and test steps for system verification may not be enough, because they normally exploit only a limited



(a) Direct Form I structure.



(b) Direct Form II structure.



(c) Transposed Direct Form II structure.

Fig. 1 Second-order modules.

number of scenarios and variable values. Moreover, filter designers usually

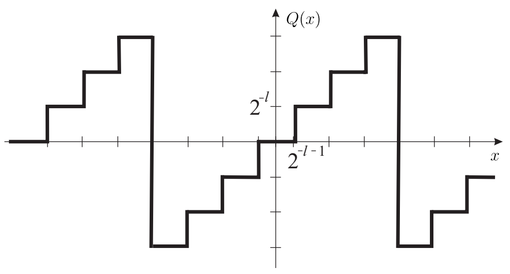


Fig. 2 Round-off quantizer of l precision bits, with wrap around overflow.

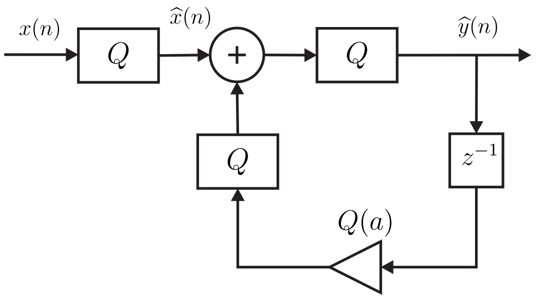


Fig. 3 Realistic model of a single-pole quantized filter.

adopt floating-point tools for evaluating their projects, which has the potential to let some failures go unnoticed. As a result, one can argue that detecting problems caused by fixed-point implementations, regarding digital filters, is a challenge that deserves formal verification-methods.

2.1.2 Finite word-length effects in digital filters

Target systems for implementing digital signal processing applications, like DSP and FPGA devices, are not capable of computing with infinite precision and commonly use finite word-length, which leads to deterioration in performance. Such an effect has two separate origins: quantization and roundoff errors during computations [30].

One of the most known problems related to finite word-length is the overflow/underflow condition, which can compromise results in recursive filters. As stated in the related literature [2,29], a filter output can be computed with the convolution sum, given as

$$y(n) = \sum_{k=-\infty}^{\infty} x(n-k)h(k), \tag{3}$$

where $x(n)$ is the input, $h(k)$ is the impulse response of the filter, and $y(n)$ is the output. Given that the present approach addresses *bounded-input bounded-output* (BIBO) stable systems, and that the input is in the interval $[-x_{max}, x_{max}]$, the output is then limited in amplitude [2,29], according to

$$|y(n)| \leq x_{max} \sum_{k=-\infty}^{\infty} |h(n)|\,, \tag{4}$$

which implicitly employs the L-1 norm, given by

$$||h||_1 = \sum_{k=-\infty}^{\infty} |h(n)|. \tag{5}$$

This way, overflows can be avoided by using a representation capable of accommodating the signal range related to $y(n)$. Since such a rule is very conservative and the roundoff noise may be unnecessarily increased [31], for instance, the L-2 norm may also be employed, but it does not guarantee the complete absence of overflow conditions [32].

It is worth noticing that even taking into account what was presented above, any intermediate addition, in a digital filter, may still overflow; however, as long as two's complement arithmetic is used, it does not affect the final result of summations. Indeed, since the 2's complement is a modulo arithmetic, if intermediate operations overflow, the associated system will still produce correct outputs, as long as the final result falls into the expected range, even if one of the operands has already overflowed, due to a multiplication. Nonetheless, that implies the addition of more than two numbers, as long as their output is representable. Such a property is commonly known as the Jackson's rule [33,34] and is extensively used in digital filters, in order to simplify designs and minimize word lengths, given that all quantizers are configured to wrap around.

Regarding Fig. 1, an important observation must be made. Parts (a) and (c) of the same figure show that multiplier outputs are directly fed to unique equivalent adders (disregarding delays), which means that rule is valid. Part (b), in turn, shows two equivalent adders (input and output), connected through a multiplier (b_0), which also means that rule is valid; however, the output of the equivalent input adder must not overflow. If it is not avoided, the result of the equivalent output adder may be incorrect. Indeed, each adder result may be seen as the output of a separate filter, which will produce correct results as long as its final computation does not overflow.

In summary, if each component filter is correctly designed and two's complement with wraparound is employed, the jackson's rule will apply (as done here). Indeed, practical designs take into consideration the output of existing equivalent adders, in such a way that the resulting filter will not suffer from overflow, independently of intermediate operations. Nonetheless, if the result saturates [9], such an assumption is no longer applicable and each node must be evaluated for avoiding overflow conditions [20,21].

In an ideal stable filter, the output should asymptotically approach a steady-state level, which is determined by the filter transfer function [35]. However, if a limit cycle problem exists, it manifests itself either as a steady oscillation or a nonzero level in the output, even for a zero level input. Such an effect is normally caused by round-off errors and overflows, during filter operation, when a feedback loop is present. Limit cycles may be dominant for low inputs, but their importance is diminished with increasing amplitudes [36].

Limit cycles can be avoided, for instance, when minimum L_2 sensitivity realizations are employed [37], by using magnitude truncation [38], by controlled rounding [39], or with specific filter structures [40]. Also, there are works, available in the related literature, that tackle this problem specifically for saturation arithmetic [11].

Another approach for dealing with fixed-point implementations relies on word-length optimization, in such a way that hardware requirements are minimized, while trying to maintain a desired performance measure. Sung and Kum [17] used search algorithms that keep a desired signal-to-quantization-noise ratio (SQNR); however, results are dependent on the signal input samples, which must be carefully chosen.

Nguyen, Menard, and Sentieys [18] also proposed word-length optimization algorithms, in which the absence of overflows is ensured, in the first step of the algorithm, through analytical determination of data behavior.

There has been great interest also in the computation [41] and analysis [42, 43] of the roundoff noise, in order to study its interaction with the filter output and even provide design tools for minimizing noise power or pole sensitivity.

The finite word-length problem can also be treated by means of a unifying approach [32], which is capable of describing equivalent realizations and measures for designing optimal systems. Although it addresses both implementation- and realization-level, a great deal of effort must still be employed, for the development of suitable optimization algorithms.

Indeed, there is a great deal of techniques available for digital filter analysis and design, which can be applied to many scenarios; however, only a few works (*e.g.*, Hildare's approach [32]) present a more general methodology, as proposed here.

2.2 Bounded Model Checking (BMC)

The Bounded Model Checking (BMC) techniques, based on Boolean Satisfiability (SAT) [22] or Satisfiability Module Theories (SMT) [23], have been successfully applied for verifying single- and multi-threaded programs, and also for finding subtle bugs in real programs [24,25,28,44]; however, applications aiming to ensure correctness of digital filters are only recent [20,21].

The basic idea of BMC is to check (the negation of) a given property at a given depth. Supposing a transition system M, a property ϕ, and a bound k, BMC unrolls the system k times and translates it into a verification condition (VC) ψ, in such a way that ψ is satisfiable if and only if ϕ has a counterexample, of depth less than or equal to k. Given that, standard SMT/SAT solvers can be used to check whether ψ is satisfiable.

In BMC of digital filters, the bound k limits the number of loop iterations and recursive calls in the filter implementation. BMC thus generates VCs that reflect the exact path in which a statement is executed, the context in which a given function is called, and the bit-accurate representation of expressions [24]. It is worth noticing that the validity proof, for VCs arising from programs, is still a major performance bottleneck, despite attempts to cope with increasing system complexity, by applying SMT solvers.

In this work, the Efficient SMT-Based Bounded Model Checker (ESBMC) tool is used as the verification engine, since it was one of the most efficient BMC tools, for bit-vector programs, in the last software verification competitions [45–47]. In ESBMC, the associated SMT-based BMC problem is formulated by constructing the logical formula

$$\psi_k = I(s_0) \wedge \bigvee_{i=0}^{k} \bigwedge_{j=0}^{i-1} \gamma(s_j, s_{j+1}) \wedge \overline{\phi(s_i)}, \tag{6}$$

where ϕ is a safety property (*e.g.*, overflow represented as an *assert*-statement in the C source-code level), I is the set of initial states of M, and $\gamma(s_j, s_{j+1})$ is the transition relation of M between time steps j and $j+1$. Hence, $I(s_0) \wedge \bigwedge_{j=0}^{i-1} \gamma(s_j, s_{j+1})$ represents executions of the filter function, unrolling up to length i. The above VC ψ_k can then be satisfied if and only if, for some $i \leq k$, there exists a reachable state, at time step i, in which ϕ is violated. If (6) is satisfiable, then the SMT solver provides a satisfying assignment, from which values of program variables can be extracted, in order to construct a counterexample. The latter, for a property ϕ, is then defined as a sequence of states $s_0, s_1, \ldots, s_k$, with $s_0 \in S_0$ and $s_k \in S$, and $\gamma(s_i, s_{i+1})$, for $0 \leq i < k$. If (6) is unsatisfiable, then one can concluded that no error state is reachable, in k steps or less.

2.2.1 Related Work on Model-Checking

There are some model-checking tools that have been used for verifying real-time systems, but not digital filters. An example is UPPAAL [48], which is a model checker based on the timed automata theory, that is, it is applied to real-time systems modeled via a timed automata network. Another similar tool is Open-Kronos [49], which is able to check reachability of timed automata and emptiness of timed Büchi automata. The CPN tools [50] are also applied to the verification of real-time systems, which are modeled via a colored (timed and untimed) Petri net.

As a remark, at the time the present paper was written, only recent initiatives had used SMT-based BMC techniques to verify finite word-length effects, on digital filter properties and realizations [20, 21, 26], which is the focus of this work. For instance, the one presented by Cox *et al.* [20, 21] tackled verification techniques regarding bit-precise analysis and real-valued error approximations, based on empirical evaluations. In particular, the present approach differs from the latter in the following aspects: Cox *et al.* did not consider the time and error verifications, which are of paramount importance in filter design, they did not exploit different filter design techniques in their benchmarks, there is no check related to the actual C implementation that is going to be embedded into the hardware platform, and, finally, they do not exploit different state-of-the-art BMC tools for C programs, which have considerably evolved over the last years. Indeed, Cox *et al.* use the ABC model checker [51], which incorporates a BMC implementation as part of the portfolio solver; however, ABC does not noticeably change results [21].

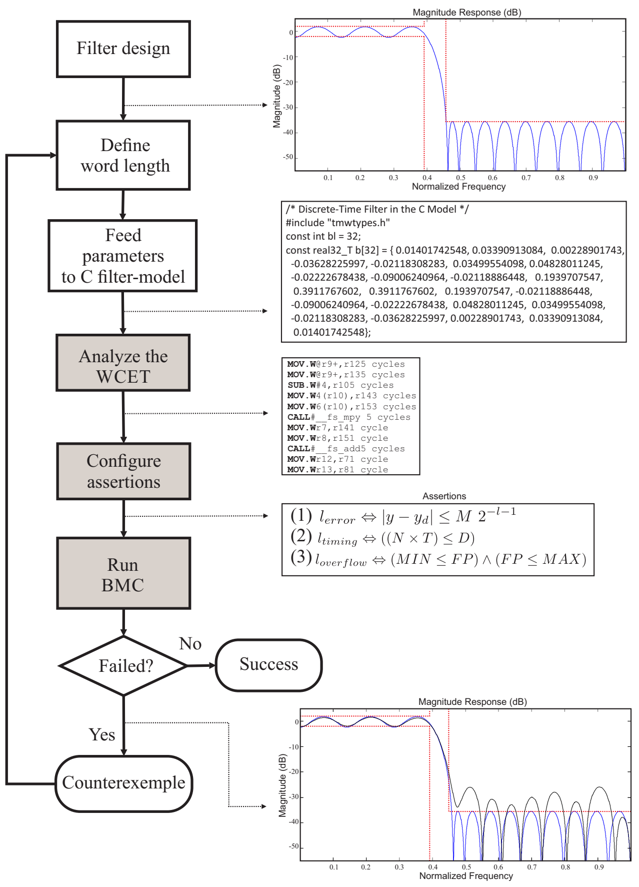
3 Research design and methodology

The main goal of the present work is to provide a general scheme for digital filter verification, based on DSVerifier [52]. Such an approach is composed of digital-filter verification rules and the verification engine itself.

The following steps for design and verification of digital filters are proposed, as shown in Fig. 4. First, filter parameters are designed, using the preferred methods [2,29] and tools [53] (cf. Filter design). After that, the output range for a given input is estimated (*e.g.,*based on the criteria proposed by Carletta *et al.* [54]), in order to define the word-length for representing fixed-point numbers (cf. Define word length). Once the word-length is defined, the respective design parameters are fed to the filter model implemented in C language, which include number of bits, realization, and sample time (cf. Feed parameters to C filter-model). Then, one can perform a time analysis regarding filter operations, considering a specific microprocessor architecture, including instruction-set architecture and CPU clock (cf. Analyze the WCET). Finally, assertions are added to the given model, in order to check properties related to time constraints, overflow, limit cycle, and output errors (cf. Configure assertions). The unwinding bound to run the BMC procedure is determined according to the filter function, based on the computation of output samples (cf. Run BMC).

If any property violation is found, then DSVerifier reports a counterexample, which contains system inputs that lead to the respective system failure (cf. Counterexample). A successful verification result is reported if the system is safe with respect to ϕ, up to bound k (cf. Success).

In Fig. 4, white boxes represent actions to be performed by the filter designer, while gray boxes represent actions to be performed automatically by DSVerifier (*i.e.*, without filter designer interventions). One can notice that if an under or overflow occurs, a high output error is found. Such information is extracted from a counterexample, which is provided by the verification engine, and then the word-length is increased, which is followed by another call to the verification engine. Alternatively, if a time constraint violation is found (which is also extracted from the counterexample), then it indicates that the filter complexity and the word-length must be decreased, in order to achieve a performance improvement.



```
/* Discrete-Time Filter in the C Model */
#include "tmwtypes.h"
const int bl = 32;
const real32_T b[32] = { 0.01401742548, 0.03390913084,  0.00228901743,
  -0.03628225997, -0.02118308283,  0.03499554098, 0.04828011245,
  -0.02222678438, -0.09006240964, -0.02118886448,   0.1939707547,
   0.3911767602,   0.3911767602,   0.1939707547, -0.02118886448,
  -0.09006240964, -0.02222678438,  0.04828011245,  0.03499554098,
  -0.02118308283, -0.03628225997, 0.00228901743,  0.03390913084,
   0.01401742548};
```

```
MOV.W@r9+,r125 cycles
MOV.W@r9+,r135 cycles
SUB.W#4,r105 cycles
MOV.W4(r10),r143 cycles
MOV.W6(r10),r153 cycles
CALL#__fs_mpy 5 cycles
MOV.Wr7,r141 cycle
MOV.Wr8,r151 cycle
CALL#__fs_add5 cycles
MOV.Wr12,r71 cycle
MOV.Wr13,r81 cycle
```

(1) $l_{error} \Leftrightarrow |y - y_d| \leq M\ 2^{-l-1}$
(2) $l_{timing} \Leftrightarrow ((N \times T) \leq D)$
(3) $l_{overflow} \Leftrightarrow (MIN \leq FP) \wedge (FP \leq MAX)$

Fig. 4 The proposed approach implemented in DSVerifier.

It is worth noticing that the timing verification is performed inside the DSVerifier context, by exploiting exhaustive checking via non-deterministic inputs. Indeed, such a verification is very important and must be included in the tool set, since it aids in determining the tradeoff among filter property conformance, fixed-point representation, and hardware requirements.

Fig. 5 shows an algorithm for a Direct Form I filter, which is implemented as C source-code in DSVerifier, as shown in Fig. 6, so that Eq. 6 can be derived in ESBMC (or CBMC) and checked by an SMT (or SAT) solver. In Fig. 6, the functions `fxp_add`, `fxp_mult`, and `fxp_sub` take two input arguments and return the respective addition, multiplication, and subtraction result in `fxp32_t` format, which is internally defined in DSVerifier as `int32_t`. The addition and multiplication blocks also include the quantization effect on the result, considering the fixed-point representation used for the system. The quantizer can be configured to saturation, wrap-around or throw an error, when the result of an operation exceeds its representable limits. Similarly, DSVerifier also im-

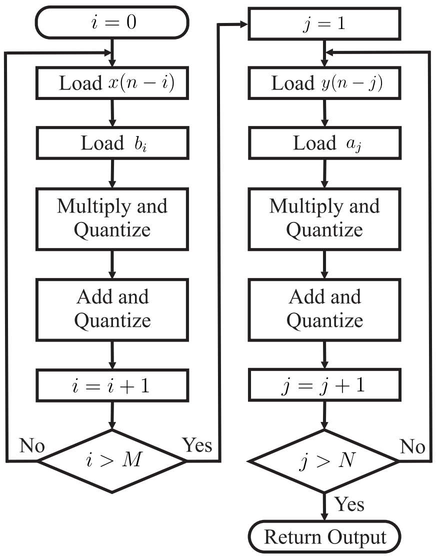


Fig. 5 Algorithm of a Direct Form I filter.

```
fxp32_t iirOutFixed(fxp32_t y[], fxp32_t x[], fxp32_t a[],
                    fxp32_t b[], int N, int M) {
  fxp32_t *a_ptr, *y_ptr, *b_ptr, *x_ptr;
  fxp32_t sum = 0;
  a_ptr = &a[1];
  y_ptr = &y[N - 1];
  b_ptr = &b[0];
  x_ptr = &x[M - 1];
  int i, j;
  for (i = 0; i < M; i++) {
    sum = fxp_add(sum, fxp_mult(*b_ptr++, *x_ptr--));
  }
  for (j = 1; j < N; j++) {
    sum = fxp_sub(sum, fxp_mult(*a_ptr++, *y_ptr--));
  }
  return sum;
}
```

Fig. 6 C code for a Direct Form I filter.

plements the filter functions in Direct Form II and Transposed Direct Form II, using the C language and a fixed-point library[3].

In addition, one may notice that the code in Fig. 6 employs pointer arithmetic, instead of normal indexing. From the verification point of view, pointers correspond directly to memory addresses, thus avoiding indexing complexity, while array indexing involves multiplying indexes by the element size and adding such a result to the array base address, which has a higher cost, in

[3] The DSVerifier source code can be downloaded from http://dsverifier.org/

terms of verification time. Currently, ESBMC, CBMC, and other state-of-the-art verifiers, regarding C programs, are able to efficiently and effectively support pointer arithmetic and comparisons [55].

The main goal of the present study is not proposing advanced methods for digital-filter parameter verification, but instead incorporating (some) techniques and then provide a general framework for system verification, based on DSVerifier. This way, the employed verification is robustly implemented and enables an iterative scheme for designing and testing digital filter modules.

3.1 Illustrative Example

As a toy example, which is supposed to fail for illustrative purposes, a single-pole system, described by the difference equation

$$y(n) = -a\ y(n-1) + x(n), \tag{7}$$

is presented. This is a *bounded-input bounded-output* (BIBO) stable system [2, 29], in which the output is limited in amplitude, according to (4). Regarding (7), with $a = -1/2$, it can be shown that the summation of the impulse response norm converges to 2 using geometric series (*i.e.*, $\sum |h_k| = \frac{1}{1-0.5} = 2$). In this particular example, if an input in the range $[-1, 1]$ is considered, the output will be $[-2, 2]$ (*i.e.*, the input range is simply multiplied by $\sum |h_k|$). Given that, one could choose to represent the fixed-point number using two bits for the integer part, including the sign, and four bits for the fractional one, according to the classical method described by Carletta *et al.* [54]. It is worth noticing that this is not the only criteria, since there may be restrictions regarding the target hardware, which may even allow a larger number of bits. As a result, the designer must take into account system restrictions and minimum required number of bits. The resulting range for this particular format would then be $[-2, 1.9375]$, with an error of ± 0.03125.

The proposed method can then be employed for verifying the proposed setup. The coefficients of (7) are used for the filter model in C language, with the previously defined numeric representation. In DSVerifier, the user must provide the specification in a ANSI-C file, as shown in Fig. 7. This file contains the digital-system specification (ds), with numerator (ds.b = {1}) and denominator (ds.a = {1.0, 0.5}), and the implementation specification itself (impl), which contains the number of bits in the integer (impl.int_bits = 2) and fractional (impl.frac_bits = 4) parts, and the input range (impl.min = -1 and impl.max = 1).

The user must also define the realization form (*e.g.*, DFI) and the verification configuration parameters (*e.g.*, solver and timeout). As an example regarding the verification of overflow occurrences, with a timeout of 1 hour and a bound of 10 cycles, the following parameters must be provided to DSVerifier:

```
dsverifier  <filename>  --realization DFI  --property OVERFLOW
--x-size 10 --timeout 1h --solver boolector
```

```
#include<dsverifier.h>

digital_system ds = {
    .a = { 1.0, 0.5 },      /* denominator        */
    .a_size = 2,            /* denominator length */
    .b = { 1 },             /* numerator          */
    .b_size = 1             /* numerator length   */
};

implementation impl = {
    .int_bits = 2,          /*  integer bits      */
    .frac_bits = 4,         /*  precision bits    */
    .min = -1.0,            /*  minimum input     */
    .max = 1.0,             /*  maximum input     */
};
```

Fig. 7 A digital-system verification input file for DSVerifier.

If DSVerifier is run, by taking into account the input range $[-1, 1]$, then it quickly shows a counterexample in which the system gets an overflow, for a particular sequence of inputs. For instance, it can be easily shown that an input sequence $x = \{1, 1, 1, 1, 1, 1\}$ leads to an overflow in the output, as reported in Table 1.

The solver adoption depends basically on its coverage and verification speed. In tests performed with many applications, Boolector [56] and Z3 [57] presented large coverage and reasonable verification times. In addition, all C/C++ language structures used in filter implementations are handled by Boolector and Z3, which makes them good choices. Of course, if filters are developed with different tools or libraries, which are deeper supported by another solver, the designer has the flexibility to change it.

Additionally, an unwinding bound of $k = 10$ to unroll all loops and recursive functions is set, so that a property violation is exposed for this particular digital filter implementation; however, since DSVerifier entirely relies on the BMC technique, it is susceptible to exhaustion of time (or memory) limits, when checking digital filters whose unwinding bounds are too large (see our experimental evaluation). Finally, the timeout is an optional parameter to limit the verification time.

Once again, there may be hardware and application restrictions, which will influence the choice regarding realization forms. For example, if there are time restrictions and many logic cells are available, one may choose to use a parallel structure. Besides, if truncation noise is a concern (*i.e.*, the associated SNR), *e.g.*, when processing audio, a DFI is preferred. Finally, if memory cells are a scarce resource, TDFII and DFII are viable options.

For this particular case, one could easily infer about overflow conditions by simply analyzing the impulse response summation or simulating a constant step input; however, it is worth noticing that impulse response summations are useful to infer about intermediate operations.

Table 1 *Overflow* occurrence in the toy example.

n	**1**	**2**	**3**	**4**	**5**	**6**
$x(n)$	1	1	1	1	1	1
$y(n)$	1.0000	1.5000	1.7500	1.8750	1.9375	1.96875

As a consequence of the overflow reported by DSVerifier, the designer then has to re-define the word length. In this particular example, if the number of bits in the integer part (*e.g.*, impl.int_bits = 2) is increased, then the user can rectify the single-pole system implementation and, for this particular word-length, the verification is successfully completed.

DSVerifier might also be used via Graphical User Interface (GUI), which is available in the DSVerifier website[4]. More details about the DSVerifier GUI and command-line can be found in the article written by Ismail *et al.* [52].

The next section presents the methods, which were employed in the proposed approach, for implementing the digital filter verifications shown in section 2.1.2. In addition, there is a deeper discussion about specific features of each verification procedure.

4 Methods

4.1 Arithmetic Underflow and Overflow Verification

During the design of fixed-point filters, one needs to specify the number of integer and fractional bits. First, the output range of the filter, for a given input, must be estimated; such a procedure typically relies on analytical- [58] or simulation-based approaches [7,17,18]. Thus, the designer should specify a suitable word-length for representing variables, considering also quantization errors in the system response.

In the present work, assertions are coded in the quantizer block and the verification engine is configured to use non-deterministic inputs, in the specified range, in order to detect overflows in digital filters, for a given fixed-point word-length. For any addition or multiplication results, during filter operation, if there exists a value that exceeds the range representable by the fixed-point approach, an assert statement detects that as an under or overflow violation. As a consequence, a literal $l_{overflow}$ is generated, with the goal of representing the validity of each addition and multiplication operation, according to the constraint

$$l_{overflow} \Leftrightarrow (MIN \leq FP) \wedge (FP \leq MAX), \tag{8}$$

where *FP* is the fixed-point approximation [24], for the result of adders and multipliers, and *MIN* and *MAX* are the minimum and maximum values representable for the given fixed-point bit format, respectively (as previously described in section 2.1.1). One can notice that constraints are typically included

[4] http://www.dsverifier.org/downloads

into the verification engine simply as an *assert*-statement in the C source-code level (*e.g.*, assert($(MIN \leq FP)$ && $(FP \leq MAX)$)).

In contrast to our toy example, regarding cases where intermediate operations cause overflow, the filter defined by

$$y(n) = 0.703125\ y(n-1) - 0.5\ y(n-2) + \\ 0.75\ x(n) - 0.703125\ x(n-1) + 0.75\ x(n-2) \tag{9}$$

can illustrate such situations. In this example, it is considered that the fixed-point values are represented with 2 bits for the integer part and 6 bits for the fractional one. The summation $\sum |h_k|$ for this filter converges to 1.8279; this way, if inputs in the range $[-1, 1]$ are applied to this system, it then provides output values that are representable in the fixed-point range $[-2, 1.984375]$. When a bounded verification of this system is performed, with implementations in Direct Form I and Transposed Direct Form II structures, the proposed tool issues a counterexample that causes an overflow on an intermediate operation. Fig. 8 illustrates the overflow on the summation after the first stage on the Direct Form I implementation, when the counterexample input is applied to the filter. For the Direct Form II filter, the verification successfully finishes without issuing any overflow failure, due to operation execution-order in this particular case. As a consequence, such observations then reinforce the application of BMC to digital filters, as an auxiliary verification tool.

Indeed, as already mentioned for saturation arithmetic, overflows in intermediate operations are considered errors; however, if wrap-around is employed, the system will still arrive at the correct result. This way, if wrap-around is used, overflow assertions for intermediate operations are simply disabled.

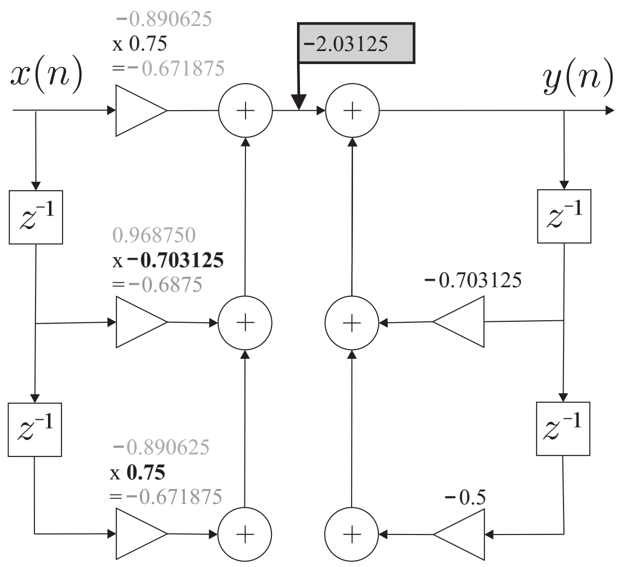


Fig. 8 Overflow in a Direct Form I filter.

4.2 Limit Cycle Verification

In order to verify the presence of limit cycles, in a particular fixed-point filter realization, the quantizer block routine is configured by setting a flag variable on it, in order to enable wrap around on overflows. The expected behavior will be as shown in Fig. 2, which means that the verification engine is not expected to detect overflow failures, as in the previous case. Additionally, the filter is configured to use a zero input signal and a non-deterministic initial state, for previous outputs. The filter execution is then unrolled, for a bounded number of entries, and an assert statement is added to detect a failure, if a set of previous outputs states (that repeats during the zero-input response) is found. One can notice that this method is different from the one presented by Cox *et al.* [20,21], which aims at finding a limit cycle by comparing input and output windows, within a bounded number of steps.

As an example, the same system described by (7) is considered. Here, such a filter is also modeled using 2 bits for the integer part and 4 bits for the fractional one (as in the previous case), but with a zero input signal. If the verification engine is executed for the implemented model, then it finds a particular initial condition that leads the system to a limit cycle. In Table 2, the system response, for that particular condition, is presented. Columns y_2 and y_{10} represent the filter response, in binary and decimal formats, respectively. Due to the rounding procedure, which was applied to the fractional part of the fixed-point number, one can notice that, in Table 2 and for $a = 0.5$, the output starts repeating after $n = 2$. Similarly, for $a = -0.5$, the output keeps in a nonzero steady-state value, instead of decaying towards zero.

Table 2 Limit cycle in the toy example.

$a = 0.5_{10} = 0.1000_2$			$a = -0.5_{10} = 1.1000_2$		
n	y_2	y_{10}	n	y_2	y_{10}
-1	0.0010	0.125	-1	0.0010	0.125
0	1.0001	-0.0625	0	0.0001	0.0625
1	0.0001	0.0625	1	0.0001	0.0625
2	1.0001	-0.0625	2	0.0001	0.0625
3	0.0001	0.0625	3	0.0001	0.0625

4.3 Time-Constraint Verification

There are efficient structures for implementing digital filters, such as the Lattice form and filtering methods based on the Fast Fourier transform [59], which aim to reduce the number of arithmetic operations and computational costs. However, the time-domain convolution methods, based on direct forms, are still prevalent, both in hardware and software implementations, due to their simplicity.

In real-time applications, a given filter receives data at the same rate it processes and outputs it. As a result, time constraints verifications become necessary, especially in high-order filters, which present many arithmetic operations and higher group delays.

Differently from Cox *et al.* [20,21], in which time constraint verifications are not supported, the proposed approach uses filter models for verifying the maximum acceptable time for filter operations, in order to tradeoff among filter property conformance, fixed-point representation, and hardware requirements.

As an example, an IIR filter function was implemented and compiled to run on a MSP430G2231, which is an ultra-low-power 16-bit RISC-CPU based microcontroller [60]. Since the filter implementation in such architecture is straightforward, it is assumed here that the timing behavior is repeatable and predictable.

So, with the assembly file generated from the compilation, this can be compared with the source code using the `__asm`*("block identifier")* function[5] [61], with the goal of identifying instructions for each program segment. After that, a worst case execution time (WCET) analysis (using BMC) can be performed in the filter function, considering the number of cycles required to execute instructions and iterations.

BMC is required since one needs to compute the worst case timing behavior of the filter; otherwise, a high burden on filter testing, with the goal to achieve the longest path, must be placed. Kim *et al.* [62] describes a method using static analysis and the model-checking technique to check program-segment timing, similarly to what is done here.

The code fragment shown in Fig. 9(a) is used to perform multiplications involving coefficients b_k and previous entries, in (1). Fig. 9(b) shows the code in Fig. 9(a) converted into some assembly instructions, using the compiler CCS v4 [63].

One can then realize that each instruction takes a different number of clock cycles to execute and, based on that information, it is possible to compute the number of clock cycles that will be needed for each operation. For MSP430G2231, the internal frequency is up to 16 MHz, which gives a cycle time of 2187.5 ns. Once the total processing time for associated instructions is available, then it can be used to increment a timer variable and add an assert statement, in order to detect any time-constraint violation.

The constraint value can be easily estimated, based on the sample rate of the system; for instance, if it operates using a sample rate of 48 KHz (which is commonly used in digital audio systems), then it means that after each 20.8μs window, new data are obtained from the system input, and the filter function has to process output samples within this time. Formally, a literal l_{timing} is generated to represent the validity of the time response, with a constraint

$$l_{timing} \Leftrightarrow ((N \times T) \leq D), \tag{10}$$

[5] In the GCC compiler, the mapping back to C code, using the `__asm`*("block identifier")* function, can be obtained by running `gcc -O2 -S -c <file>`

```
    sum += *b_ptr++ * *x_ptr--;
```

(a)

```
    MOV.W  @r9+,r12  5 cycles
    MOV.W  @r9+,r13  5 cycles
    SUB.W  # 4,r10   5 cycles
    MOV.W  4(r10),r14  3 cycles
    MOV.W  6(r10),r15  3 cycles
    CALL # __fs_mpy  5 cycles
    MOV.W  r7,r14    1 cycle
    MOV.W  r8,r15    1 cycle
    CALL # __fs_add  5 cycles
    MOV.W  r12,r7    1 cycle
    MOV.W  r13,r8    1 cycle
```

(b)

Fig. 9 (a) C code fragment of the digital filter. (b) Assembly instructions of the code fragment shown in (a).

where N is the number of cycles spend by the filter, T is the cycle time, and D is the deadline.

4.4 Error Verification

As widely known, computation using finite word-length leads to rounding and truncation errors [29,64]. In the present work, impairments present on a filter output, due to coefficient rounding and arithmetic-operation results, are considered. As shown in Fig. 2, the quantization error E, due to the rounding of a number represented with l bits of precision, is

$$-2^{-l-1} \leq E \leq 2^{-l-1}. \qquad (11)$$

One may notice that the accuracy on the computation of IIR and FIR filters is limited by the word-length specified in the digital system realization. As already mentioned, coefficient rounding changes pole and zero positions and also modifies the frequency response, which cause variations on the filter output that can also be observed in time domain.

Floating-point variables provide a better approximation of rational numbers, when compared with fixed-point representations presenting the same number of bits, since they cover a larger dynamic range; however, practical implementations of digital filters are typically realized in fixed-point representation. Additionally, encoding floating-point arithmetic into the BMC framework leads to large formulas and, consequently, a high time and memory consumption for verification. Considering that, typical verification engines support fixed-point representation, using bit-vector and rational arithmetic.

Regarding the bit-vector arithmetic, which was applied to this work, it assumes that integral and fractional number parts have the same bit-widths, before and after the radix point. Thus, for 64-bits *double* variables, 32-bits are used for representing the fractional part, and the remaining 32-bits are used to represent the integer one, including sign.

Based on what was presented above, the verification of output error bound (defined by the designer) is proposed, as opposed to the framework developed by Cox *et al.* [20,21], in which error verifications are not supported. For this purpose, the output of a filter, implemented with reduced-word-length fixed-point representation, is compared with a reference output of the same structure, which was implemented using double precision variables. It is important to notice that both models present quantization errors; however, since in reference models full *double* variables are used, the error amplitude is much lower than in fixed-point ones. Thus, considering that the number of precision bits l_d in reference models is higher than in designed models, the quantization error E_r is given by

$$-2^{-l-1} + 2^{-l_d-1} \leq E_r \leq 2^{-l-1} - 2^{-l_d-1}. \tag{12}$$

The expression above shows that the error, computed with the proposed method, is affected by the precision of the reference model. In the implemented system, 64-bits fixed-point variables were used for the reference model, with 32 precision bits. The experiments were executed on models with less than 16 bits for the fractional part, which provides a good approximation for error calculations.

For the reduced fixed-point models, the computed values are saturated to the maximum representable number on overflow, or to the minimum on underflow; the same input is applied to both models. The input vector uses non-deterministic values from $\{2^{k-1} - 2^{-l}, 2^{-l}, 0, -2^{-l}, 2^{k-1}\}$, that is, values for maximum and minimum amplitudes of the input signal. So, errors due to quantization and saturation are explored.

During filter operation unrolling, the cumulative error can increase beyond the quantization-error interval. The following literal is then generated, in order to check whether the output error is in accordance with an acceptable margin:

$$l_{no\_error} \Leftrightarrow |y - y_d| \leq M \cdot 2^{-l-1}, \tag{13}$$

where y is the output of the designed system, y_d is the output of the double precision system, and $M\ 2^{-l-1}$ is the tolerance margin. The verification process searches for the negation of this literal and, when a counterexample is found, it indicates that the output error is higher than expected for that filter realization.

5 Results and discussion

This section is split into five parts. The experimental setup is described in section 5.1, while the other sections present verification results and considerations for digital filter benchmarks, which are all realistic examples extracted from the literature [2,21,29,59].

The proposed methodology checks the actual C code of digital filters, which are intended to be embedded into micro-controllers and DSPs. Such an approach is very close to real implementations, where specific C constructs (*e.g.*, pointer arithmetic and comparisons) are used to realize (1), which makes VCs harder to be checked. Additionally, the present work innovates on exploring different verification techniques, which are used to detect overflows and limit cycles, in digital filters.

It is worth noticing that the present work also proposes a set of necessary checks, by exploiting state-of-the-art bounded model checkers, with the goal of aiding the choice of system representation and structure, while meeting the design specifications.

5.1 Experimental Setup

Table 3 describes some filters chosen with different design types, number of feedback coefficients N, number of forward coefficients M, input range, and word-length. One can notice that column *Bits* indicates the word-length for the integer and fractional parts of fixed-point representations, including the sign. Besides, the word-length for fixed-point representations is estimated based on both the $\sum |h_k|$ summation and the input range, in order to obtain optimized filters, in terms of reduced number of bits. The column *Bound* shows the number of consecutive entries applied to the filter (filter function unwinding), which is empirically determined to find violations of the associated property, up to given bound. Indeed, for each applied entry, the filter function is executed, in order to compute output samples. From the verification perspective, all important information about the digital filters benchmarks are described in Table 3.

In Table 3, filters from 1 to 11 are verified in three different realizations: Direct Form I (DFI), Direct Form II (DFII) and Transposed Direct Form II (TDFII), in order to investigate how filter structures can interfere in the occurrence of some failures. Many of the selected test cases use second order structures, because such a realization is widely used and can be applied as a building block to form higher-order systems, as referred in section 2.1.1.

Regarding the error verification experiments, they were fair enough to define a range larger than the quantization error interval, as given by (11). Due to that, an acceptable margin of two times the precision was fixed, that is, from -2^{-l+1} to 2^{-l+1}. Note that all filter operations are represented in our verification engine as fixed-point arithmetic. The error verification only checks whether the output of a filter, implemented with reduced-word-length fixed-point representation, is within certain bounds, when compared to a double-precision reference output.

When evaluating time constraints, restrictions related to a 16 MHz processor, operating on a system with sample rate equal to 48 KHz, were considered.

Table 3 Tested digital filters

#	**Filter**	N	M	$\sum \lvert h_k \rvert$	**Input**	**Bits**	**Bound**
1	LP-IIR	2	1	2	$[-1, 1]$	$< 2, 4 >$	6
2	LP-Butterworth-IIR	3	3	1.2	$[-1.6, 1.6]$	$< 2, 5 >$	6
3	LP-IIR	3	1	4	$[-1, 1]$	$< 3, 4 >$	6
4	LP-IIR	3	1	1.56	$[-1, 1]$	$< 2, 4 >$	6
5	HP-ChebyshevI-IIR	3	3	1.33	$[-1, 1]$	$< 2, 6 >$	6
6	BP-Elliptic-IIR	3	3	1.24	$[-1, 1]$	$< 2, 8 >$	6
7	BS-Butterworth-IIR	3	3	1.81	$[-1.1, 1.1]$	$< 2, 8 >$	6
8	BP-Elliptic-IIR	5	5	0.91	$[-1.1, 1.1]$	$< 1, 7 >$	10
9	HP-Butterworth-IIR	5	5	1.58	$[-1.27, 1.27]$	$< 2, 6 >$	10
10	BP-ChebyshevI-IIR	5	5	1.51	$[-1.33, 1.33]$	$< 2, 6 >$	10
11	HP-Elliptic-IIR	7	7	5.39	$[-1, 1]$	$< 3, 11 >$	14
12	HP-Cascade-IIR	6	6	12.4	$[-1, 1]$	$< 5, 5 >$	14
13	BS-Cascade-IIR	9	9	2.45	$[-1, 1]$	$< 3, 5 >$	19
14	LP-Parallel-IIR	6	6	16.6	$[-1, 1]$	$< 5, 5 >$	13
15	LP-Cascade-IIR	6	6	7.64	$[-1, 1]$	$< 4, 4 >$	13
16	Multiband-Parallel-IIR	9	9	2.75	$[-1, 1]$	$< 4, 4 >$	19
17	LP-FIR	1	31	1.94	$[-1, 1]$	$< 2, 6 >$	31
18	Multiband-FIR	1	9	2.18	$[-1, 1]$	$< 2, 6 >$	13
19	Multiband-FIR	1	25	1.47	$[-1, 1]$	$< 2, 8 >$	25
20	LP-WiFi-FIR	1	21	2.92	$[-1, 1]$	$< 3, 5 >$	21

Such a rate was adopted due to its use as a standard audio sampling rate, commonly employed in professional equipment. One can notice that the system sample rate does not interfere in overflow and limit cycle conditions, because this is just a consequence of the fixed-point arithmetic.

Regarding overflow and limit cycle verifications, results for FIR filters are not presented here, since, in this kind of system, such issues can be easily avoided. By definition, FIR filters are completely immune to limit cycles [2] (there is no feedback), and overflows can be prevented by applying a conservative criterion based on the filter-impulse-response absolute sum, in order to determine the word-length. Indeed, FIR systems, checked for those properties, finished successfully or timed out without finding a single counterexample. It is worth noticing that overflow checking for intermediate operations was enabled, in order to guide a possible use of saturation arithmetic. As a result, the number of VCs are substantially increased, since it depends on the filter function unwinding bound. For instance, if the unwinding bound is set to k, then the verification engine produces k VCs to check for overflows, which are harder to be checked by an SMT (or SAT) solver.

Here, DSVerifier[6] was employed and configured to use the SMT solver Z3 v4.0 [57], with the bit-vector arithmetic enabled. Cox *et al.* [20, 21] observed that integer arithmetic is identical to that of bit-vector, in terms of precision; however, there were tradeoffs in performance with (slight) advantage to bit-vector arithmetic.

For each benchmark, the verification engine was invoked by manually setting file name, timeout, and bit-vector-arithmetic solver[7]. Note that array bounds, pointer safety, and division by zero assertions are disabled, since the present work is focused on checking finite word-length problems, as previously described in section 3. The above ESBMC call is thus used for checking safety properties related to arithmetic under and overflow, when `<file>` refers, for instance, to *verify_overflow.c*. In order to check for limit cycle, timing constraints, and quantization error, the following files can be referenced by the above ESBMC call, respectively: *verify_limitcycle.c*, *verify_timing.c*, and *verify_error.c*. Each file contains the necessary filter calls and assertions used for property verification.

All experiments were conducted on an otherwise idle Intel Core i7 – 2600, with a clock of 3.40 GHz and 24 GB of RAM, running Fedora 64-bits. For all digital filters, the individual time limit has been set to 3600 seconds, except for cascade/parallel systems and FIR filters, in which the timeout is 7200 seconds. The presented elapsed times were measured using the *time* command.

5.2 General results

The parameters related to the chosen digital filters are used as inputs to the C filter-model. Tables 4, 5, and 6 summarize the results obtained with DSVerifier, by presenting them as *TRUE* or *FALSE*, i.e., whether the verification has finished successfully or not, respectively. The verification time is also shown for each type of failure assertion, and, when a particular verification exceeds the related time limit, the test case is tagged as *Timeout*.

One can notice that the proposed system is able to detect failures in various digital filter types, which are implemented with different structures, orders, or fixed-point formats. However, the verification time tends to be higher for high order filters and long word-length formats, since those lead to harder VCs. In particular, the verification time tends to be higher when checking overflows, in filters that contain a high number of forward and feedback coefficients, as can be seen from test case 9 onward. In addition, timeout events occurred when checking for limit cycles, with digital filters that present long word-length for the fractional part representation (*e.g.*, test case 11). The error verification also leads to high verification times, since it needs to calculate the output twice,

[6] DSVerifier is an internal module for ESBMC and CBMC, which is available at *http://dsverifier.org/*, together with the benchmarks, so that other researchers can reproduce the results

[7] In particular, the ESBMC tool was invoked as follows: `esbmc <file> --no-bounds-check --no-pointer-check --no-div-by-zero-check --timeout` *1h* `--z3-bv`

Table 4 Summary of results for the tested IIR filters

			Overflow		Limit Cycle		Timing		Error	
#	**Filter**	**Type**	**Result**	**Time**	**Result**	**Time**	**Result**	**Time**	**Result**	**Time**
1	LP-IIR	DFI	FALSE	3	FALSE	8	TRUE	1	TRUE	6
		DFII	FALSE	2	FALSE	13	TRUE	<1	TRUE	4
		TDFII	FALSE	2	FALSE	8	TRUE	<1	TRUE	4
2	LP-Butterworth-IIR	DFI	FALSE	2	TRUE	417	FALSE	<1	TRUE	31
		DFII	FALSE	1	TRUE	709	FALSE	1	FALSE	5
		TDFII	TRUE	30	FALSE	447	FALSE	<1	TRUE	32
3	LP-IIR	DFI	TRUE	10	FALSE	21	TRUE	<1	TRUE	14
		DFII	TRUE	12	FALSE	55	TRUE	1	TRUE	10
		TDFII	TRUE	39	FALSE	135	TRUE	<1	TRUE	12
4	LP-IIR	DFI	TRUE	4	TRUE	88	TRUE	<1	TRUE	11
		DFII	TRUE	5	TRUE	106	TRUE	<1	TRUE	9
		TDFII	TRUE	11	FALSE	101	TRUE	<1	TRUE	10
5	HP-ChebyshevI-IIR	DFI	TRUE	10	TRUE	941	FALSE	<1	TRUE	48
		DFII	TRUE	27	TRUE	1776	FALSE	1	TRUE	57
		TDFII	TRUE	14	FALSE	331	FALSE	<1	TRUE	49
6	BP-Elliptic-IIR	DFI	TRUE	9	FALSE	37	FALSE	1	TRUE	53
		DFII	FALSE	2	FALSE	70	FALSE	<1	FALSE	16
		TDFII	TRUE	11	FALSE	105	FALSE	1	TRUE	46
7	BS-Butterworth-IIR	DFI	FALSE	3	FALSE	437	FALSE	<1	FALSE	17
		DFII	FALSE	2	FALSE	112	FALSE	<1	FALSE	11
		TDFII	TRUE	117	FALSE	321	FALSE	<1	TRUE	73
8	BP-Elliptic-IIR	DFI	FALSE	3	TRUE	8	FALSE	1	FALSE	168
		DFII	FALSE	2	TRUE	14	FALSE	1	FALSE	79
		TDFII	FALSE	2	TRUE	12	FALSE	<1	FALSE	147
9	HP-Butterworth-IIR	DFI	FALSE	15	FALSE	1060	FALSE	1	FALSE	125
		DFII	FALSE	4	FALSE	1506	FALSE	1	FALSE	145
		TDFII	Timeout	3600	FALSE	1485	FALSE	<1	Timeout	3600
10	BP-ChebyshevI-IIR	DFI	TRUE	96	Timeout	3600	FALSE	1	FALSE	255
		DFII	FALSE	11	FALSE	2395	FALSE	1	FALSE	106
		TDFII	TRUE	139	FALSE	2092	FALSE	<1	FALSE	238
11	HP-Elliptic-IIR	DFI	FALSE	12	Timeout	3600	FALSE	1	Timeout	3600
		DFII	FALSE	4	Timeout	3600	FALSE	1	Timeout	3600
		TDFII	FALSE	5	Timeout	3600	FALSE	1	Timeout	3600

that is, for fixed- and floating-point arithmetic formats. Apart from that, time constraints are easily verified, since this procedure consists of only checking the time response of a sequential piece of code.

Another important observation regarding verification times, as noted during tests conducted for validating the proposed methodology, is that failing test cases tend to be quickly checked. The main reason is that the model-checking algorithm stops a verification procedure whenever it finds a counterexample; however, cases where no defect is found tend to have high verification times, or even produce a timeout. In such scenarios, a wide set of non-deterministic inputs is applied, which generates verification conditions that are hard to be checked and makes this procedure very time-consuming. That can be noticed, for instance, on the limit cycle verification procedure for test cases 9, 10, and 11, and also in test case 9, during the overflow check. In the error verification of test case 9, the result indicates a failure for Direct Form I and Direct Form II implementations, in less or equal than 15 seconds; for the Transposed Direct Form II, the verification gets a time-out, without finding an error beyond the fixed limits. It indicates that it is hard to find an input that produces a high error value for this particular structure, which suggests that such a realization may properly operate most of the time; however, it cannot be guaranteed, since the model checking did not finish.

The timing verification reported failures for all cases with order higher than 2, which present more than 3 forward or feedback coefficients. Actually, considering the modeled structures implemented in C, only filters 1, 3, and 4 meet the time constraint requirements, when running on the specified platform.

5.3 Filter-structure considerations

Although different filter structures lead to the same final filters, as long as the same coefficients are used, the internal computation procedures are completely different. It can be seen, especially in overflow and limit cycle verifications, that a filter may fail or pass according to its implementation structure. That occurs due to the order of intermediate operations, which changes from one structure to another. The timing constraint verification is also affected by the filter structure, since the number of additions and multiplications are different in each form. For instance, test case 4 should not be implemented using the Transposed Direct Form II, in order to avoid limit-cycle occurrences. Regarding test cases 6 and 7, however, there is no viable option, unless the designer uses some technique (see section 2.1.2), in order to prevent limit cycles.

For instance, in test case 2, the verification detects overflow failures for the Direct Form I and Direct Form II structures, in addition to a limit cycle failure for the Transposed Direct Form II structure; the remaining properties finished successfully. On this particular case, the designer intending to implement this filter, in Direct Form I or II, would need to use an accumulator with a higher number of bits for the integer part, two's complement arithmetic, or other technique, when saturation arithmetic is employed, for preventing overflow. When using the Transposed Direct Form II, the designer could modify the overflow mode of the system, in order to perform saturation arithmetic for solving limit cycle oscillations. The techniques to prevent overflow and limit cycle, as referred by Proakis *et al.* in [2], generally increase noise levels. As a result, after performing such modifications, the designer should necessarily run the verification engine another time, in order to ensure that the output error is within an acceptable margin.

Table 5 describes the results obtained for filters implemented in cascade and parallel forms. Those systems use second order building blocks on their

Table 5 Summary of results for the tested cascade and parallel filters

			Overflow		Limit Cycle		Timing		Error	
#	**Filter**	**Type**	**Result**	**Time**	**Result**	**Time**	**Result**	**Time**	**Result**	**Time**
12	HP-Cascade-IIR	TDFII	Timeout	7200	Timeout	7200	FALSE	2	FALSE	1584
13	BS-Cascade-IIR	DFII	FALSE	259	Timeout	7200	FALSE	7	Timeout	7200
14	LP-Parallel-IIR	DFII	TRUE	911	FALSE	3656	FALSE	3	Timeout	7200
15	LP-Cascade-IIR	DFII	FALSE	261	FALSE	4059	FALSE	3	FALSE	189
16	Multiband-Parallel-IIR	TDFII	Timeout	7200	FALSE	6668	FALSE	<1	Timeout	7200

Table 6 Summary of results for the tested FIR filters

			Timing		Error	
#	**Filter**	**Type**	**Result**	**Time**	**Result**	**Time**
17	LP-FIR	DFI	FALSE	1	Timeout	7200
18	Multiband-FIR	DFI	FALSE	1	TRUE	7915
19	Multiband-FIR	DFI	FALSE	1	Timeout	7200
20	LP-WiFi-FIR	DFI	FALSE	1	Timeout	7200

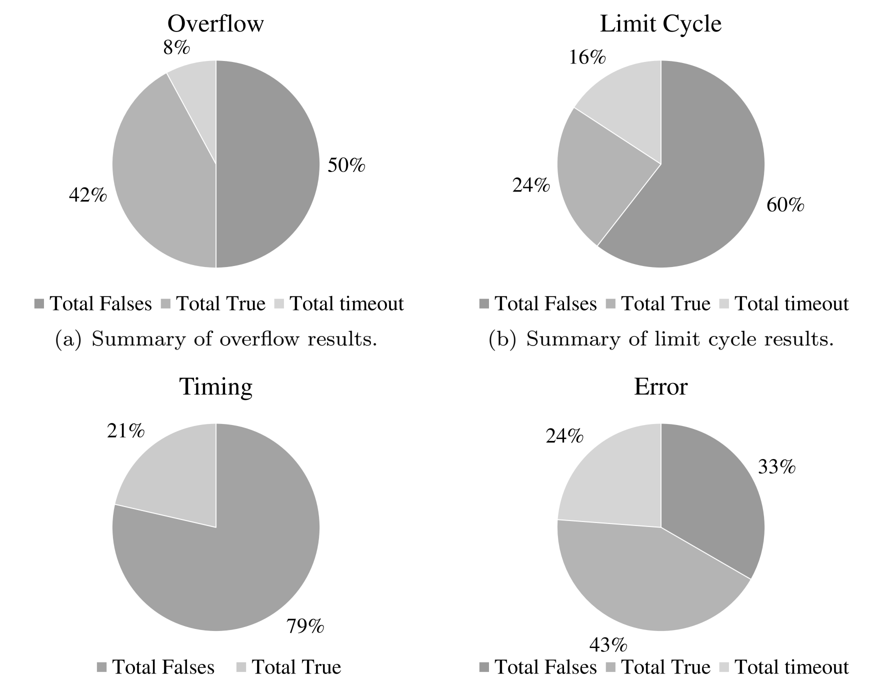
structures, as mentioned in section 2.1.1. The results show that the number of bits, used on those implementations, is not enough for avoiding problems caused by the reduced word-length, as overflow and error violations are found in most cases. Here, checking procedures for error, overflow, and limit cycle are very time-consuming, due to the high order of such systems. Given that, an alternative choice for the test cases that timeout is, at least, ensuring the conformance of the second order blocks separately, since those structures can be checked faster, as shown in Table 4.

5.4 FIR filter results

The results collected from the FIR filter experiments are described in Table 6. The high-order systems cause long verification times, due to the search for a counterexample that produces a high output error. The timing verifications are also quickly performed, as in the other presented cases. Multiband filters 18 and 19 were also verified for these properties and, as can be seen in test case 19, the timing failure indicates that the particular implementation must be simplified.

5.5 Final considerations

Fig. 10 shows a summary of all obtained results, classified by verification category. Fig. 10(a) presents results regarding arithmetic under- and overflows, which show that DSVerifier was able to find property violations in 50% of the benchmarks. In addition, only 8% of the cases timed out, due to the high



(a) Summary of overflow results. (b) Summary of limit cycle results.

(c) Summary of timing constraints results. (d) Summary of quantization errors results.

Fig. 10 Summary of the results of all benchmarks, per verification category.

number of generated VCs, regarding the system under verification. In particular, a VC is generated for each arithmetic expression of the digital filter implementation, considering non-deterministic inputs.

Fig. 10(b) presents results regarding limit cycle verification. In summary, DSVerifier was able to find property violations in 60% of the benchmarks and timed out in 16% of the them, given that limit cycle verifications of high-order filters, with long word-lengths, led to a large state-space to be explored by the chosen SMT solver. One can notice that the results of both categories do not include verification of FIR filters, due to the already stated reasons.

Fig. 10(c) presents results regarding timing-constraint verifications, which show that DSVerifier was able to find timing violations in 79% of the benchmarks. The verification of timing constraints is fast (in some cases, it takes less than 1 second), and no benchmark, in this category, timed out.

Finally, Fig. 10(d) presents results regarding quantization errors, where DSVerifier was able to find property violations in 33% of the benchmarks and timed out in 24% of them, for reasons previously stated for overflow verification.

In summary, the presented results show that the proposed verification methodology covers a large set of checks, which are required by common designs, in order to define the fixed-point representation format. Besides, it is also useful in determining the filter structure for system realization, *i.e.*, in observing whether the system implementation is feasible, given the project constraints.

It has also been shown that DSVerifier can be a very effective tool for verifying digital filters, aiding DSP engineers to automatically discover low-level properties violations, which are hard to found by traditional simulation tools (*e.g.*, Matlab [65]), since they depend on a set of input stimuli, in order to improve the state space coverage, and require a significant manual intervention from designers. In particular, simulations tools [17,66] might neglect some failures, due to low coverage achieved during simulations (coverage problem) [20, 21]. Our verification methodology then replaces typical validation processes currently used by DSP engineers.

6 Conclusions

In this work, a new approach to detect failures in fixed-point digital filters, using state-of-the-art BMC tools, was proposed. It allows the designer to formally check a given implementation for a specific bit-width, in addition to define the word-length to properly represent numeric data. In particular, the proposed approach supports the designer in detecting problems caused by the finite word-length, such as overflows, limit cycles, and output error, in digital filters. The experimental results show that failures can be detected in low and medium order digital filters, with arbitrary bit-width. However, the verification of high order filters, with long word-length, tends to be a hard problem, due to the large state-space exploration.

There is also a contribution with a new method based on WCET analysis, together with BMC, which is used to verify time constraints, in digital filters. Since the tested digital-filter models are implemented in C language, the proposed approach could also use other state-of-the-art software verifiers for C programs, by taking advantage of their robustness and efficiency. Additionally, the resulting filter C code (with specific word-length format) could be implemented directly into DSPs or converted into hardware description languages, which are supported by typical FPGAs.

Finally, the tool set developed here, which was named as DSVerifier, can help system designers, when defining adequate representation and structure, in order to meet functional and performance requirements. However, if such requirements are not met, then the user has to manually modify the digital filter representation and/or structure. In future work, the counterexample, generated on the verification of a failing design, will be used to automate fault localization [67], regarding digital filters implementations.

7 Competing interests

The authors declare that they have no competing interests.

8 Authors' contributions

RA carried out the implementation of the DSVerifier approach and its experimental evaluation; he also drafted the manuscript. MG carried out the implementation of the ESBMC tool to support specific functions of the DSVerifier approach; he also drafted the manuscript. LC carried out the implementation of the ESBMC tool to support fixed-point arithmetic for digital filters; he also drafted the manuscript. EL defined the verification algorithms for overflow, limit cycle, and error; he also drafted the manuscript. WS provided the background in digital signal processing; he also drafted the manuscript.

9 Authors' information

RA received the B.Sc. degree in electrical engineering from the Federal University of Viçosa, Minas Gerais, Brazil, in 2007. He is currently working on the MS degree in electrical engineering at the Federal University of Amazonas, Brazil. He is also a senior researcher at Nokia Institute of Technology. He has a background in electrical engineering with interests in systems verification, PHY layer development, test and measurement systems.

MG was born in Manaus, AM, Brazil, in 1988, received the B.Sc. degree in computer engineering and the M.Sc. degree in electrical engineering from the Federal University of Amazonas (UFAM), in 2010 and 2013, respectively. He worked developing mobile applications for Maemo and Meego in 2009 and developing SBTVD (Brazilian Digital TV standard) in 2011. His work focuses on software verification, bounded model checking, mobile applications development, and embedded systems.

LC received the B.Sc. degree in electrical engineering and the M.Sc. degree in informatics from the Federal University of Amazonas (UFAM), in 2005 and 2007, respectively. He received the Ph.D. degree in computer science from the University of Southampton in 2011. Since 2011 he has been an adjunct professor in the Electrical and Computer Engineering Department at UFAM. His work focuses on software verification, model checking, satisfiability modulo theories, and embedded systems.

EL received the B. Sc. degree in electrical engineering from the Federal University of Amazonas (UFAM), Manaus, AM, Brazil, in 1999, and the M. Sc. and D. Sc. degrees in electrical engineering from the Federal University of Rio de Janeiro (COPPE/UFRJ), Rio de Janeiro, RJ, Brazil, in 2004 and 2008, respectively. Since 2007, he has been with the Science, Technology and Innovation Center for the Industrial Pole of Manaus (CT-PIM), Manaus, AM, Brazil, working with digital TV, middleware and embedded systems. His research interests include digital signal processing, channel/source coding, video/image processing, embedded systems, and cognitive radio.

WS received the B.Sc. degree in electrical engineering from the Federal University of Amazonas (UFAM), Manaus, AM, Brazil, in 2000. He received the M.Sc. and D.Sc. degree in electrical engineering from the Federal University of Rio de Janeiro (COPPE/UFRJ), Rio de Janeiro, RJ, Brazil, in 2004 and 2010, respectively. Since 2006, he has been an adjunct professor in the Electrical and Computer Engineering Department (DEC) at UFAM. His research interests are in the fields of pattern recognition, digital signal processing and data compression.

Acknowledgements This research was supported by CNPq 475647/2013-0 and FAPEAM 062.01722/2014 grants. Additionally, the authors would like to thank the PPGEE/UFAM for supporting the development of this project.

References

1. Institute of Electrical and Electronics Engineers (2008) IEEE Standard for Floating-Point Arithmetic (IEEE Std 754-2008). IEEE, New York.
2. Proakis JG, Manolakis, DG (1996) Digital Signal Processing: Principles, Algorithms, and Applications. Prentice Hall, Upper Saddle River.
3. Parker SR, Hess SF (1971) Limit-Cycle Oscillations in Digital Filters. IEEE Trans Circuit Theory 18(6):687-697.
4. Bauer P, Leclerc LJ (1991) A computer-aided test for the absence of limit cycles in fixed-point digital filters. IEEE Trans Signal Process 39(11):2400-2410.
5. Shafic EM, Sandberg IW (1995) A study of bounds on limit cycles in digital filters. Circuits, Systems and Signal Processing 14(6):725-734.
6. Campo J, Cruz-Roldan F, Utrilla-Manso M (2006) Tighter limit cycle bounds for digital filters. IEEE Signal Process Lett 13(3):149-152.
7. Claasen TACM, Mecklenbrauker WFG, Peek JBH (1976) Effects of Quantization and Overflow in Recursive Digital Filters. IEEE Trans Acoust, Speech, Signal Process 24(6):517-529.
8. Vaidyanathan PP, Liu V (1987) An improved sufficient condition for absence of limit cycles in digital filters. IEEE Trans Circuits Syst 34(3): 319-322.
9. Ahn CK (2013) IOSS Criterion for the Absence of Limit Cycles in Interfered Digital Filters Employing Saturation Overflow Arithmetic. Circuits Systems and Signal Processing 32(3):1433-1441.
10. Kawamata M (2008) On the Absence of Limit Cycles in State-Space Digital Filters With Minimum L2-Sensitivity. IEEE Trans Circuits Syst II, Exp Briefs 55(1):4650.
11. Ahn CK (2011) Criterion for the elimination of overflow oscillations in fixed-point digital filters with saturation arithmetic and external disturbance. International Journal of Electronics and Communications 65(9):750-752.
12. Constantinides GA, Cheung PYK, Luk W (2000) Roundoff-noise shaping in filter design. In: IEEE International Symposium on Circuits and Systems, Geneva, May 2000. Proceedings of ISCAS 2000, vol 4, pp 57-60.
13. Henzel N (2005) Digital Filter Design with Constraints in Time and Frequency Domains. In: 4th International Conference on Computer Recognition Systems, Rydzyna Castle, Poland, May 2005. Advances in Soft Computing, vol 30, pp 169-176.
14. SYNOPSYS®(2013) SPW. `http://www.synopsys.com/Systems/BlockDesign/DigitalSignalProcessing/Pages/Signal-Processing.aspx`. Accessed 20 Nov 2013.
15. MathWorks®(2013) Fixed-Point Designer. `http://www.mathworks.com/products/simfixed`. Accessed 20 Nov 2013.
16. Wang CC, Shi C, Brodersen RW, Marković D (2011) An Automated Fixed-Point Optimization Tool in MATLAB XSG/SynDSP Environment. ISRN Signal Processing 2011:1-17.
17. Sung W, Kum KL (1995) Simulation-Based Word-Length Optimization Method for Fixed-Point Digital Signal Processing Systems. IEEE Trans Signal Process 43(12):3087-3090.

18. Nguyen HN, Menard D, Sentieys O (2011) Novel algorithms for word-length optimization. In: European Signal Processing Conference, Barcelona, Spain, September 2011. Proceedings of EUSIPCO 2011, pp 1944-1948.
19. Menard D, Rocher R, Sentieys O (2008) Analytical Fixed-Point Accuracy Evaluation in Linear Time-Invariant Systems. IEEE Trans Circuits Syst I, Reg. Papers 55(10):3197-3208.
20. Cox A, Sankaranarayanan S, Chang BYE (2012) A Bit Too Precise? Bounded Verification of Quantized Digital Filters. In: 18th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, Tallinn, Estonia, April 2012. Lecture Notes in Computer Science, vol 7214. Springer, Heidelberg, pp 33-47.
21. Cox A, Sankaranarayanan S, Chang BYE (2014) A bit too precise? Verification of quantized digital filters. Software Tools for Technology Transfer 16(2):175-190.
22. Biere A (2009) Bounded model checking. in: Handbook of Satisfiability, IOS Press.
23. Barrett CW, Sebastiani R, Seshia SA, Tinelli C (2009) Satisfiability modulo theories. in: Handbook of Satisfiability, IOS Press.
24. Cordeiro L, Fischer B, Marques-Silva J (2012) SMT-Based Bounded Model Checking for Embedded ANSI-C Software. IEEE Trans Softw Eng 38(4):957-974.
25. Cordeiro L Fischer B (2011) Verifying multi-threaded software using SMT-based context-bounded model checking. In: International Conference on Software Engineering, Honolulu, USA, May 2011. Proceedings of ICSE 2011, pp 331-340.
26. Abreu RB, Cordeiro LC, Filho EBL (2013) Verifying Fixed-Point Digital Filters using SMT-Based Bounded Model Checking. In: XXXI Brazilian Telecommunications Simposyum, Fortaleza, Brazil, September 2013. Proceedings of SBrT 2013, pp 1-5.
27. Bessa I, Abreu R, Chaves Filho JE, Cordeiro L (2014) SMT-Based Bounded Model Checking of Fixed-Point Digital Controllers. In: 40th Annual Conference of the IEEE Industrial Electronics Society, Dallas, UA, October 2014. Proceedings of IECON 2014, pp 295-301.
28. Clarke E, Kroening D, Lerda F (2004) A tool for checking ANSI-C programs. In: International Conference on Tools and Algorithms for the Construction and Analysis of Systems, Barcelona, Spain, March 2004. Lecture Notes in Computer Science, vol 2988. Springer, Heidelberg, pp 168-176.
29. Oppenheim AV, Schafer RW, Buck JR (1999) Discrete-Time Signal Processing. 2nd ed. Prentice Hall, Upper Saddle River.
30. Gevers M, Li G (1993) Parametrizations in Control, Estimation, and Filtering Problems: Accuracy Aspects. Springer-Verlag, London.
31. Mills W, Mullis C, Roberts RA (1978) Digital filter realizations without overflow oscillations. IEEE Trans Acoust, Speech, Signal Process 26(4):334-338.
32. Hilaire T (2011) Towards tools and methodology for the fixed-point implementation of linear filters. In: Digital Signal Processing Workshop and Signal Processing Education Workshop, Sedona, USA, January 2011. Proceedings of DSP/SPE 2011, pp 488493.
33. Jackson LB, Kaiser JF, McDonald HS (1968) An Approach to the Implementation of Digital Filters. IEEE Trans Audio and Electroacoust AU-16(3):413-421.
34. Dattorro J (1988) The Implementation of Recursive Digital Filters for High-Fidelity Audio. J Audio Eng Soc 36(11):851-878.
35. Brubaker T, Gowdy J (1972) Limit Cycles in Digital Filters. IEEE Trans Autom Control 17(5):675-677.
36. Hanselmann H (1987) Implementation of digital controllers - a survey. Automatica 23(1):7-32.
37. Yamaki S, Abe M, Kawamata M (2008) On the Absence of Limit Cycles in State-Space Digital Filters With Minimum L_2-Sensitivity. IEEE Trans Circuits Syst II, Exp Briefs 55(1):46-50.
38. Auer E (1987) Digital filter structures free of limit cycles. In: IEEE International Conference on Acoustics, Speech, and Signal Processing, Dallas, USA, April 1987. Proceedings of ICASSP 1987, vol 12, pp 904-907.
39. Ritzerfeld JHF, Mollova GS (1997) Controlled rounding in low noise digital filter structures. In: ProRISC Workshop on Circuits, Systems and Signal Processing, Mierlo, The Netherlands, September 1987. Proceedings of the ProRISC Workshop on Circuits, Systems and Signal Processing, pp 433-437.

40. Kwan H (1985) A multi-output second-order digital filter without limit cycle oscillations. IEEE Trans Circuits Syst 32(9):974-975.
41. López JA, Caffarena G, Carreras C, Nieto-Taladriz O (2008) Fast and accurate computation of the roundoff noise of linear time-invariant systems. IET Circuits, Devices and Systems 2(4):393-408.
42. Jackson L (1970) Roundoff-noise analysis for fixed-point digital filters realized in cascade or parallel form. IEEE Trans Audio Electroacoust 18(2):107-122.
43. Hilaire T, Ménard D, Sentieys O (2007) Roundoff noise analysis of finite wordlength realizations with the implicit state-space framework. In: 15th European Signal Processing Conference, Poznan, Poland, September 2007. Proceedings of EUSIPCO 2007, pp 1019-1023.
44. Merz F, Falke S, Sinz C (2012) LLBMC: Bounded Model Checking of C and C++ Programs Using a Compiler IR. In: Verified Software: Theories, Tools, Experiments, Philadelphia, USA, January 2012. Lecture Notes in Computer Science, Vol 7152. Springer Heidelberg, pp 146-161.
45. Cordeiro L, Morse J, Nicole D, Fischer B (2012) Context-Bounded Model Checking with ESBMC 1.17. In: 19th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, Tallinn, Estonia, March 2012. Lecture Notes in Computer Science, vol 7214. Springer, Heidelberg, pp 534-537.
46. Morse J, Cordeiro L, Nicole D, Fischer B (2013) Handling Unbounded Loops with ESBMC 1.20. In: 19th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, Rome, Italy, March 2013. Lecture Notes in Computer Science, vol 7795. Springer, Heidelberg, pp 619-622.
47. Morse J, Ramalho M, Cordeiro L, Nicole D, Fischer B (2014) ESBMC 1.22 - (Competition Contribution). In: 20th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, Grenoble, France, April 2014. Lecture Notes in Computer Science, vol 8413. Springer, Heidelberg, pp 405-407.
48. Carlson J, Hakansson J, Pettersson P (2005) SaveCCM: An Analysable Component Model for Real-Time Systems. In: International Workshop on Formal Aspects of Component Software, Macao, China, October 2005. Proceedings of FACS 2005, vol 160, pp 127-140.
49. Tripakis T, Yovine S, Bouajjani A (2005) Checking Timed Buchi Automata Emptiness Efficiently. Formal Methods in System Design 26(3):267-292.
50. Jensen K, Kristensen LM (2009) Coloured Petri Nets: Modelling and Validation of Concurrent Systems. Springer.
51. Brayton RK, Mishchenko A (2010) ABC: An Academic Industrial-Strength Verification Tool. In: 22nd International Conference on Computer-Aided Verification, Edinburgh, July 210. Lecture Notes in Computer Science, vol 6174. Springer, Heidelberg, pp 24-40.
52. Ismail H, Bessa I, Cordeiro L, de Lima Filho EB, Chaves Filho E (2015) DSVerifier: A Bounded Model Checking Tool for Digital Systems. In: 22nd International SPIN Symposium on Model Checking of Software, Stellenbosch, South Africa. Lecture Notes in Computer Science, vol 9232. Springer, Heidelberg, pp 126-131.
53. MathWorks®(2013) Open Filter Design and Analysis Tool. `http://www.mathworks.com/help/dsp/ref/fdatool.html`. Accessed 21 Nov 2013.
54. Carletta J, Veillette R, Krach F, Fang Z (2003) Determining appropriate precisions for signals in fixed-point IIR filters. In: Design Automation Conference, Anaheim, USA, June 2003. Proceeding of the Design Automation Conference, pp 656-661.
55. Beyer D (2015) Software Verification and Verifiable Witnesses - (Report on SV-COMP 2015). In: 21st International Conference on Tools and Algorithms for the Construction and Analysis of Systems, London, UK, April 2015. Lecture Notes in Computer Science, vol 9035. Springer, Heidelberg, pp 401-416.
56. Brummayer R, Biere A (2009) Boolector: An efficient SMT solver for bit-vectors and arrays. In: International Conference on Tools and Algorithms for the Construction and Analysis of Systems, York, UK, March 2009. Lecture Notes in Computer Science, vol 5505. Springer, Heidelberg, pp 174-177.
57. de Moura LM, Bjorner N (2008) Z3: An Efficient SMT Solver. In: 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, Budapest, Hungary, April 2008. Lecture Notes in Computer Science, vol 4963. Springer, Heidelberg, pp 337-340.

58. Balakrishnan V, Boyd S (1992) On computing the worst-case peak gain of linear systems. Systems & Control Letters 19(4):265-269.
59. Diniz PSR, da Silva EAB, Netto SL (2010) Digital Signal Processing: Systems Analysis and Design. 2nd ed. Cambridge University Press.
60. Texas Instrument$^{\mathrm{TM}}$(2013) MSP430G2231 Mixed Signal Controller. `http://www.ti.com/lit/ds/symlink/msp430g2231-ep.pdf`. Accessed 21 Nov 2013.
61. GNU (2015) The GNU C Reference Manual. `http://www.gnu.org/software/gnu-c-manual/`. Accessed 15 Jun 2015.
62. Kim S, Patel HD, Edwards SA (2009) Using a Model Checker to Determine Worst-case Execution Time. Technical Report. Computer Science Department, Columbia University.
63. Texas Instrument$^{\mathrm{TM}}$(2013) Code Composer Studio$^{\mathrm{TM}}$Integrated Development Environment for MSP430. `http://www.ti.com/tool/ccstudio-msp430`. Accessed 19 Nov 2013.
64. Akbarpour B, Tahar S (2007) Error Analysis of Digital Filters using HOL Theorem Proving. Journal of Applied Logic 5(4):651-666.
65. Davis TA (2010) MATLAB primer. 7th ed. CRC Press.
66. Luengo D, Oses D, Martino L (2014) Monte carlo limit cycle characterization. In: IEEE International Conference on Acoustics, Speech and Signal Processing, Florence, Italy, May 2014. Proceedings of ICASSP 2014, pp 8043-8047.
67. Könighofer R, Bloem R (2011) Automated error localization and correction for imperative programs. In: International Conference on Formal Methods in Computer-Aided Design, Austin, USA, October 2011. Proceedings of FMAC, pp 91-100.