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

Verification of Fixed-Point Digital Controllers Using Direct and Delta Forms Realizations

Iury V. Bessa · Hussama I. Ismail · Lucas C. Cordeiro · João E. C. Filho

Received: date / Accepted: date

Abstract The extensive use of fixed-point digital controllers demands a growing effort to prevent design errors that appear in the discrete-time domain. The present article describes a novel verification methodology, which employs Bounded Model Checking (BMC) based on Satisfiability Modulo Theories (SMT) to verify the occurrence of the design errors, because of the finite word-length (FWL) format, in fixed-point digital controllers. Here, the performance realizations of the digital controllers realizations that use delta operators are compared to those that use traditional direct forms. The experimental results show that the delta-form realization substantially reduces the digital controllers' fragility when compared to the direct-form realization. Additionally, the proposed methodology can be very effective and efficient to verify real-world digital controllers, where conclusive results are obtained in nearly 98% of the benchmarks.

Keywords fixed-point digital controllers · direct and delta forms · formal methods · bounded model checking

1 Introduction

Digital controllers are now widely used by the control engineering community because of various advantages over analog controllers such as improved reliability, sensitivity, flexibility, and cost. However, there are some disadvantages in the use of digital controllers, for example, errors that are introduced during the quantization process. In this context, there are some initiatives to solve problems that appear in the discrete-time domain; in particular, problems related to the finite word-length (FWL) effects [1, 2].

I. Bessa and H. Ismail
Graduate Program in Electrical Engineering, Federal University of Amazonas, Brazil
E-mail: {iurybessa, hussamaibrahim}@ufam.edu.br

L. Cordeiro and J. Filho
Electronic and Information Research Center, Federal University of Amazonas, Brazil E-mail: lucascordeiro, jo_edgar}@ufam.edu.br

Digital controllers are generally implemented in microcomputers, microprocessors, digital signal processors [3], and field programmable gate arrays [4]. According to the hardware choice, the format and arithmetic used to represent and manipulate numbers might change (*e.g.*, number of bits, fixed- or floating-point arithmetic); these representations directly influence the precision and performance of the digital controller.

Floating-point processor has a numerical representation that approximates the real number values and offers an enhanced precision if compared to fixed-point processors. In particular, floating-point processors can represent numbers with smaller magnitude than the fixed-point processors; however, fixed-point processor is the fastest and cheapest solution and consequently, it is largely used in practice. The scenario aforementioned thus requires better understanding and handling of typical problems in the realizations of digital controllers so that quantizations and FWL effects can potentially be reduced when designing a digital controller. Several factors can influence, intensify, or attenuate these effects (*e.g.*, via the use of realization structures such as direct and delta forms as well as the definition of the number of bits and sample rate). The possible influences of these effects bring important stability and eigenvalues sensitiveness issues to the digital controller, which have been previously investigated by several authors [5–9].

To avoid performance degradation, control engineers usually invest in time and effort at the design phase, solving problems caused by the FWL effects with more robust and arduous solutions. Previous studies propose appropriate scaling using special metrics [8, 10]. Others developed different methodologies to estimate the optimal word-length to avoid FWL effects [11–14]. There are some initiatives that propose more complex controllers to maintain the performance inside an error bound or uncertainty bounds (*e.g.*, the robust and non-fragile controllers [15, 16]). Automated verification tools have also been applied to find design errors in discrete-time systems (*e.g.*, UPPAAL [17], Open-Kronos [18], CPN [19], and Maellan [20]). However, there is still a gap in formal verification of embedded systems; in particular in digital controllers, which are in a continuous interaction with the environment.

Differently from others, this study presents a novel methodology to formally verify the occurrence of design errors in realization of digital controllers. In particular, BMC based on the SMT is used to verify five types of properties, which include overflow, limit cycle, time constraints, stability, and minimum-phase. Additionally, six different realizations structures are considered, which include three direct forms and three delta forms.

The main objective of this research is thus to demonstrate that an SMT-based BMC method can be a very powerful tool in the design and verification of digital controllers, aiding the control engineer with an efficient verification tool that is more reliable and less laborious than traditional simulation tools (*e.g.*, Matlab [21] and LabVIEW [22]), as simulation tools depend on a set of input stimuli to improve the state space coverage and significantly require manual intervention from designers. In particular, simulations tools [14, 23] either generate false alarms or neglect some failures because of the low coverage achieved during simulations (coverage problem) [24, 25]. Our verification methodology then replaces typical validation processes currently used by control engineers.

In summary, this article describes three important contributions. It marks the first study that applies an SMT-based BMC technique to verify the occurrence of

various design errors related to FWL and quantization effects in numerical fixed-point digital controllers. Second, we demonstrate with the aid of an SMT-based BMC tool that the use of the delta-form realization presents a higher maintenance capability of important properties (*e.g.*, stability and minimum phase) than the direct-form realization using an appropriate FWL format. Finally, we show that our verification methodology can be very effective and efficient to verify real-world digital controllers than other existing approaches.

This article is a substantially revised and extended version of the study published in previous conferences [26,27]. The major difference is that the present article includes more benchmarks to confirm the methodology feasibility. The delta operator properties and their advantages in relation to the shift operator are better investigated with more benchmarks. Furthermore, the experiments are performed in a newer Efficient SMT-Based Context-Bounded Model Checker (ESBMC) version with a different SMT-solver (Boolector [28]), resulting in better performance when compared to the SMT solver Z3 [29]. Last but not least, we also describe the stability and minimum-phase verification for delta-form digital controllers.

The remainder of the article is organized as follows: Section 2 provides fundamental aspects of digital controllers, and issues related to their implementation, especially in fixed-point architectures. Section 3 presents the proposed methodology for verification of digital controllers and the respective techniques involved. Section 4 presents the results of our experiments using several benchmarks, comparing the performance of delta and direct forms implementations. In Section 5, we discuss the related studies and we conclude and describe future research in Section 6.

2 Fixed-Point Digital Controllers

This section introduces the main concepts about digital controllers, including their representations and realizations; in particular, the fixed-point representation and the related problems. The delta-form realization is also presented and its advantage is conceptually compared to the direct form.

2.1 Fixed-Point Digital Controllers Representations and Implementations

A digital controller is a linear time-invariant causal discrete-time dynamic system, which deals with discrete numerical signals; its implementation is a program executed by a microprocessor. There are various mathematical controller representations (*e.g.*, transfer functions, state equations, and difference equations). These representations are studied in several books on signals and systems (*e.g.*, [30,31]). An important representation for digital controllers is the difference equation, which can be described as

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

where $y(n)$ and $x(n)$ are the output and input, respectively, in instant n [31]. Using the z-transform in Eq. (1), the digital controller can be represented by a transfer

function

$$H(z) = \frac{b_0 + b_1 z^{-1} + ... + b_M z^{-M}}{1 + a_1 z^{-1} + ... + a_N z^{-N}} \tag{2}$$

where z is called forward-shift operator and z^{-1} is called backward-shift operator. There are many ways to implement a digital controller in software; the realization structure of controllers influences their performance. Different realizations of digital controllers are studied in [1, 32].

In this study, however, the delta form is considered and its performance is compared to the direct form, from the formal verification perspective. Others types of realizations to implement a digital controller are explained by Åström and Wittenmark, and Hilaire *et al.* [32, 33].

The direct realization uses directly the coefficients in Eq. (1) in its implementation. The advantage of this implementation is that states variables are derivations of delayed inputs and outputs, using the shift operator. However, direct-form implementations make the controller extremely sensitive to numerical errors, which are strongly evident in fixed-point implementations; as a result, it may specially harm the system's stability and performance. The different direct forms (*e.g.*, DFI, DFII, and TDFII) might present different numerical performance, given that these realizations implement the same controller, but with different operations numbers and orders. Figures 1, 2, and 3 show three different direct representations; in these structures, the gains a_i and b_i are the coefficients and the z^{-1} represents the shift operations in Equations (1) and (2).

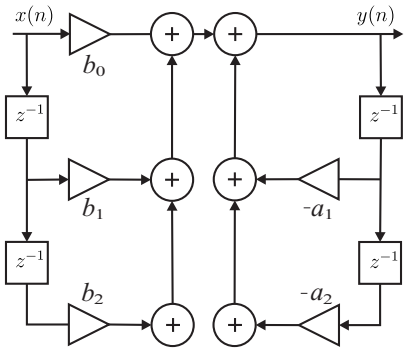

Fig. 1: Direct Form I Realization

The delta form, originally proposed by Middleton and Goodwin [8], represents a viable alternative to prevent the violation of the system's properties from the numerical errors perspective, which are typically caused by the quantizations and FWL effects.

2.1.1 Implementation of Digital Controllers Using the Delta Forms

Middleton and Goodwin proposed an alternative to minimize FWL effects in fixed-point digital controller, using the delta operator (δ), which is defined by

$$\delta = \frac{q - 1}{\Delta} \tag{3}$$

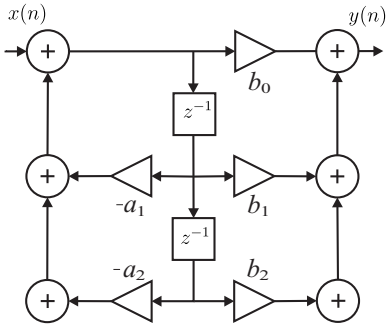

Fig. 2: Direct Form II Realization

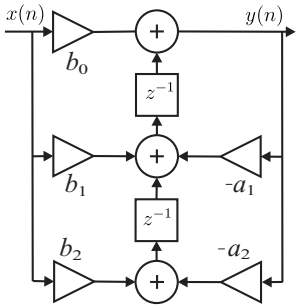

Fig. 3: Transposed Direct Form II Realization

where q is a shift operator (*e.g.*, the z) and δ is the Euler approximation to a derivative [8]. A discrete system represented by Eq. (1) in z-domain presents different coefficients in δ-domain, which can be described by the following transfer function

$$H(\delta) = \frac{\bar{b_0} + \bar{b_1}\delta^{-1} + ... + \bar{b_M}\delta^{-M}}{1 + \bar{a_1}\delta^{-1} + ... + \bar{a_N}\delta^{-N}}, \tag{4}$$

and each coefficient $\bar{a_i}$ with $0 \leq i \leq N$ and similarly for $\bar{b_j}$ with $1 \leq j \leq M$ can be described by

$$\bar{a_i} = \binom{k+i}{i} \sum_{k=i}^{N} a_k \Delta^i. \tag{5}$$

A system in delta form has exactly the same behavior as a continuously sampled system. However, this realization presents an improved round-off performance, a sensitive-less controller, more accurate coefficients representation, and a greater region of convergence so that the discrete stability domain is close to the continuous domain.

The delta operator is equivalent to the shift operator and all analysis done for the shift operator can be translated into the delta form. However, the delta operator is the forward-difference approximation of the differential continuous operator; therefore, the delta form is closer to the continuous behavior than the ordinary direct form, which uses the shift operator. Furthermore, the z-domain region of convergence (*i.e.*, the region where the stability is guaranteed) is only the unitary

radius circle and the δ-domain region of convergence is a circle inversely proportional to the sample period; it means that the set of coefficients values for which a polynomial is stable may be much greater with the delta operator.

2.2 Problems Related to Fixed-Point Implementation

Implementations of digital controllers are subject to FWL effects; these effects are amplified in a fixed-point processor. The FWL effects are related to small imprecisions or functional problems such as instability. The most common error sources are round-offs and quantizations [31]. Quantization occurs during the analog-to-digital conversion, which consists of approximating analog signal values to quantized (discrete) values. This process generates a rounding error, whose maximum value is 2^{-b-1}, where b is the number of bits in the fractional part.

A realistic model of a FWL system must include the quantization of every numerical value, including each arithmetic result (sums and products), input signals, and system coefficients; the last are narrowly related to the system's dynamics. These accumulated errors might affect the position of the poles and zeros of the digital controller (mainly in direct forms) and make the controller lose the stability or the minimum phase characteristics; in the control literature, this is called controller's fragility [1].

In the design phase, control engineers avoid poles and zero positions that might be fatally affected by the FWL effects (*i.e.*, they seek positions slightly away from the unitary circle); however, sometimes, it cannot be done easily or simply it may not be desired in the context. An example is given by resonant controllers, in which the poles must be positioned next to the stability border. The fixed-point arithmetic influence in these controllers is studied by Harnefors and Peretz *et al.* [10, 34].

A particular fixed-point representation $< k, l >$, where k is the number of bits of the integer part and l is the number of bits of the fractional part, can only represent values in the range from $2^{k-1} - 2^{-l}$ to -2^{k-1}. An overflow occurs when an addition or multiplication operation returns a result outside the range of representable values. A microprocessor generally handles an overflow via wrap-around (*i.e.*, allow the numerical representation wrapping it) or saturation (*i.e.*, hold the maximum representation). These round-offs or overflow errors might lead to periodic oscillations called limit cycles. There are books that explain completely the fixed-point theory and operations, as in Granas and Dugundji [35]. Additionally, problems related to FWL effects on fixed-point format can be extensively found in the literature [1, 31, 36, 37]. Understanding the presence of these phenomena might degrade the controller's performance; this study proposes a verification methodology that improves the design process to guarantee that a designed controller is immune to FWL effects.

2.2.1 Round-off Effect and Poles and Zeros Sensitivity

The problems related to the precision of digital controllers are well-known [1,5,6,8, 38]. There are essentially two alternatives to handle them: the first one is a simple but expensive technique that consists of increasing the word-length, using more expensive and improved hardware, with the drawback of increasing the product

price; the second one consists of optimizing and minimizing the FWL effects via deep knowledge and understanding of the phenomena related to round-off effects with the drawback of it being a quite laborious task. Some authors give an overview of the main techniques to minimize the performance degradation in digital filters, controllers, and identifiers, because of quantizations and effects of round-offs, and how to achieve optimal realization [1, 2, 39].

The round-off effects may be especially harming when coefficients are affected, because they might displace poles and zeros and consequently change the desired system's behavior [5]. Although, optimal FWL techniques, improved hardware, and special representation (with larger number of bits for coefficients) have been used, the poles and zeros sensitivity to FWL still presents a challenge. There are several tools and techniques to handle model uncertainty and perturbation (*e.g.*, $H\infty$ control), producing optimal and robust controllers. However, Keel and Bhattacharyya showed that these controllers may still present an undesired property [40]: fragility or sensitivity to implementation aspects (*e.g.*, the FWL representation of the digital controllers). The fragility of digital systems may be reduced by generating only small output errors, or may be very harmful by affecting the system's stability. To minimize controllers' fragility, some techniques such as the nonfragile control were proposed [2].

2.2.2 Arithmetic Overflow

The arithmetic overflow is a well-known problem in digital controllers. The overflow may occur in any node of the digital controller realization, when an operation result exceeds the limited range of the processor's word-length, resulting in undesirable nonlinearities at the output. In particular, these events may be avoided with a correct scaling of the input signals. Jackson *et al.* show that if the computation result of a digital controller, implemented with two-complement arithmetic, does not exceed the numerical range, then the intermediate steps of the computation may be allowed to overflow any number of times without causing an erroneous accumulation [41]; the overflow distortions because of the sums can be recovered by subsequent additions. The overflow can be avoided if one ensures that it will not occur in the final node, which can be done by scaling the multipliers' inputs. The necessary and sufficient condition to avoid overflow for any input signal is to ensure that the output will be bounded by a v_m, where v_m represents the bound of the processor's representative range.

Lemma 1 *Consider a digital controller with output $y(n)$ and input $x(n)$, bounded in magnitude by v_m, for all n, multiplied by a scale factor λ. The output $y(n)$ will be also bounded by v_m for all n for values for λ such that*

$$\lambda \leq \frac{1}{\|h(n)\|_1}, \tag{6}$$

where the $h(n)$ is the impulse response of the digital controller, and $\|\bullet\|$ is defined by $\|u(n)\| = \sum_{k=0}^{\infty} |u(k)|$.

This condition is rarely used in practice as it generally results in a very stringent scaling, which may be harmful to the accuracy of the output signal, once the word-length is finite. There are previous studies that present efforts to optimize the

number of bits used in the fixed-point format [12, 14, 42–48] or to present more relaxed norms [36, 49, 50] to minimize round-off effects. In any case, the accuracy is directly related to the dynamical behavior and to the stability of the system, which might present fragility because of the FWL effects and round-offs. Some researchers focus their efforts on the design phase, developing controllers with an adequate performance after FWL effects [2, 39, 51–53]; other authors investigate different implementation forms [54, 55].

2.2.3 Limit Cycle

The limit cycle oscillations (LCOs) in digital controllers are defined by the presence of oscillations occurring in the output, even when the input sequence is a constant value [1]. These oscillations may be very harmful to the control system, because of the signal-to-noise ratio of the controller's output, degrading control actions and causing damages to the physical plant (especially in mechanical systems), harming surround products, and increasing the material losses [56].

Some authors have studied the consequences of limit cycles; Peterchev and Sanders show that the presence of limit cycle oscillations in pulse-width modulation (PWM) power converters can increase the energy waste and decrease the lifespan of the electronic devices [57]. The same problem was also studied for resonant controllers by Peretz and Ben-Yaakov [34].

A LCO may be classified as granular or overflow limit cycles. The granular limit cycles are autonomous oscillations, originating from quantization performed in the least significant bits [36]. The absence of overflow limit cycles may be assured by preventing overflows or by treating overflows via saturation, holding the maximum (or minimum) value achieved. The granular LCOs are most hardly avoided by means of the following lemma, which is proved by Vaidyanathan and Liu [58]:

Lemma 2 *Consider any digital controller with transfer function $C(z)$, which may be represented in state-space by means of the following equations*

$$\begin{cases} x(n+1) = Ax(n) + Bu(n) \\ y(n) = Cx(n) + Du(n) \end{cases} \tag{7}$$

where $x(n)$, $u(n)$, $y(n)$, and (A, B, C, D) are respectively the state vector, the input, the output, and the state-space realization. A condition necessary and sufficient to ensure that the realization (A, B, C, D) is completely free of self-sustained oscillations is some diagonal matrix that exists with positive diagonal entries such that the following matrix Q is

$$Q = T - A^t * T * A \succeq 0. \tag{8}$$

This condition is not so easily met; there is intensive research effort for systems free of oscillations. Since the 1960s, different initiatives have been proposed to test the limit cycle absence. In particular, the initial initiatives proposed tests for the absence of LCOs based on frequency domain conditions [59, 60] and in Lyapunov functions [58, 61–64]. The first study that proposed the usage of exhaustive search to ensure the absence of LCOs in digital filters and controllers implementations appeared only after the 1990s [65–68], and some related research is used to optimize the search space by means of bounds and heuristics [68–70]. More recently, Monte Carlo algorithms are used to detect LCOs [23].

2.2.4 Sampling effect

The sample time is an important factor to be considered in a digital controller design. Obviously, the choice of a slow sampling implies difficulties in the signal reconstruction, increasing the reconstruction error. The minimum criterion is the Shannon's theorem below:

Theorem 1 *Any signal $y(t)$ that presents frequency components in the interval $\left[-\frac{\omega_s}{2}, \frac{\omega_s}{2}\right]$ can be exactly reconstructed, from the sample values $y(kT)$, where T is the sampling period and $\omega_s = \frac{2\pi}{T}$,*

$$y(t) = \sum_{k=-\infty}^{\infty} y(kT) \frac{\sin \frac{\omega_s}{2}(t - kT)}{\frac{\omega_s}{2}(t - kT)}. \tag{9}$$

This theorem is not practical; the sum in Eq. (9) is complicated to be evaluated and may accumulate large errors after the signal reconstruction. For this reason, polynomial interpolation are more commonly used in signal reconstruction (*e.g.*, zero-order hold); for this approach, a superior sample rate is necessary, which is extended to almost ten times the bandwidth to avoid the loss of information [38]. Furthermore, with the use of delta operator, small sample times can approximate the convergence region from the continuous one [8].

However, the use of excessively high sampling might result in undesirable performance, leading to greater numerical errors because of the FWL effects. In a real-time system, which may consist of other tasks beyond the control actions, high sample rates will require more processing resources; therefore, the use of high sample rates may be insufficient to execute the tasks and may degrade the system's performance. Recommended sample rates are between 10 and 50 times of the bandwidth [38].

3 Verification of Fixed-point Digital Controllers

This section describes our verification methodology for realization of digital controllers, which uses BMC based on SMT. In particular, we check for properties related to overflow, limit cycle, time constraints, stability, and minimum phase using direct- and delta-forms realizations.

3.1 Bounded Model Checking (BMC)

Formal verification of software and systems is an important task to ensure that a designed model meets its requirements; this task may be very hard and laborious with the complex growth of the systems. As model-checking techniques do not require proofs (algorithmic rather than deductive in nature), they have an important place in the verification paradigm, whenever systems are becoming more complex, especially with the growth of the cyber-physical systems, which demand short development cycles and high-level of reliability [71].

The International Competition on Software Verification (SV-COMP) is an annual competition of software verification tools, which evaluates and presents several testing and verification techniques [72] . One important verification technique,

which is presenting many of interesting results over the last years, is the BMC. The BMC techniques, based on Boolean Satisfiability (SAT) [73] or SMT [74], have been successfully applied for verifying single- and multi-threaded programs, and also for finding subtle bugs in real programs [75–78]; however, applications aiming to ensure correctness of discrete-time systems are only recent [24–27, 79].

The basic idea of BMC is to check for the negation of a given property at a given depth. Supposing a transition system M has 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. SMT solvers such as Z3 [29] and Boolector [28] can be used to check whether ψ is satisfiable. In the BMC of digital controllers, the bound k limits the number of loop iterations and recursive calls in the controller's realization, leaving enough that real errors can be found. BMC thus generates VCs that reflect the exact path in which an instruction is executed, the context in which a given controller function is called, and the bit-accurate representation of arithmetic expressions [76]. BMC tools are, however, susceptible to exhaustion of time or memory limits for digital controllers with loops whose bounds are too large or cannot be determined statically. From our benchmarks, we noted that a bound of $k = 10$ is enough to find many errors in digital controllers implementation.

In this study, we use the ESBMC tool as the verification engine, as it represents one of the most efficient BMC tools that participated in the last software verification competitions [80–82]; in particular, ESBMC is the most efficient tool to reason about programs that make use of bit-vector arithmetic according to the SV-COMP 2015 edition.

ESBMC is an SMT-based bounded model checker for C/C++ programs. ESBMC finds property violations such as pointer safety, array bounds, atomicity, overflows, deadlocks, data race, and memory leaks in single- and multi-threaded software (with shared variables and locks). It also verifies programs that make use of bit-level, pointers, structs, unions, and fixed-point arithmetics; it has been used in a previous studies to verify properties of digital filters [79] and digital controllers [26]. Inside ESBMC, the associated problem is formulated by constructing the following logical formula

$$\psi_k = I(S_0) \wedge \bigvee_{i=0}^{k} \bigwedge_{j=0}^{i-1} \gamma(s_j, s_{j+1}) \wedge \overline{\phi(s_1)} \tag{10}$$

where ϕ is a property (*e.g.*, overflow and limit cycle) and S_0 is a 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, $I(S_0) \wedge \bigwedge_{j=0}^{i-1}$ represents the executions of a transition system M of length i. The above VC ψ can 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 the logical formula (10) is satisfiable (*i.e.*, returns *true*), then the SMT solver provides a satisfying assignment, from which the values of the digital controller's variables can be extracted to construct a counterexample.

Definition 1 *A counterexample for a property ϕ is a sequence of states $s_0, s_1, \ldots, s_k$ with $s_0 \in S_0$, $s_k \in S$, and $\gamma(s_i, s_{i+1})$ for $0 \leq i < k$.*

If it is unsatisfiable (*i.e.*, returns $false$), then one can conclude there is no error state in k steps or less.

3.2 Digital Controller Verification Methodology

This study describes a novel methodology to verify digital controllers. In particular, this verification methodology is supported by the Digital Systems Verifier (DSVerifier)[1], which is a tool to model check properties related to overflow, limit cycle, time constraints, stability, and minimum phase in the implementation of digital controllers. Using DSVerifier, the control systems engineer verifies that a designed digital controller presents the desired performance, when it is embedded into a given hardware with resource limitations. An overview of the proposed methodology can be seen in Fig. 4.

In step 1, a digital controller is initially designed with typical control system techniques [56, 83]. As this controller will be implemented in a fixed-point processor, one has to define a specific fixed-point representation, in particular, the number of bits and precision (k integer bits and l fractional bits) and a computational realization structure (direct or delta form) as shown in steps 2 and 3 of Fig. 4. Initially, the DSVerifier receives the fixed-point representation, the realization structure, and the hardware specification as well as other verification parameters such as the verification time (*i.e.*, maximum time that the verification process takes) and the category that indicates the property to be verified. Once the configuration has been set in step 4, the verification process is started with a model checking tool (step 5); here we use ESBMC as back-end for DSVerifier.

DSVerifier then checks whether the properties, mentioned above, hold in the digital controller implementation. In step 6, DSVerifier returns verification successful if there is no property violation in the proposed implementation; otherwise, it returns verification failed and shows a counterexample with inputs and states that lead the system to the property violation. With this counterexample, another implementation (*i.e.*, realization and representation) can be chosen to avoid that failure. This process is repeated until the digital controller implementation does not present any failure as shown in Fig. 4.

In order to explain the DSVerifier workflow, the following second-order controller, which can be found in a set of benchmarks available online[2], will be used:

$$H(z) = \frac{1.5610z - 1.485}{z - 0.9}. \tag{11}$$

This digital controller was designed for an induction motor plant, extracted from an example available in Ogata [83], with a sampling period of $0.5s$. The transfer function definition corresponds to the first step of the methodology, which is shown in Fig. 4. The second step is to choose the FWL representation. For this particular example, the $\langle 3, 5\rangle$ fixed-point format with dynamical range between -3 and 3 is employed. A scale factor of 10 is used for the numerator coefficients. These fixed-point parameters are computed according to the classical method described by Carletta *et al.* [11], in order to avoid overflows.

For the DSVerifier usage, users must provide the specification in a ANSI-C file, as shown in Fig. 5, for the digital controller represented by Eq. (11). This file contains the digital-system specification (ds), with numerator (ds.b = {1.561, −1.485 }) and denominator (ds.a = {1.0, −0.9}), and the implementation specification

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

[2] http://www.dsverifier.org/benchmarks

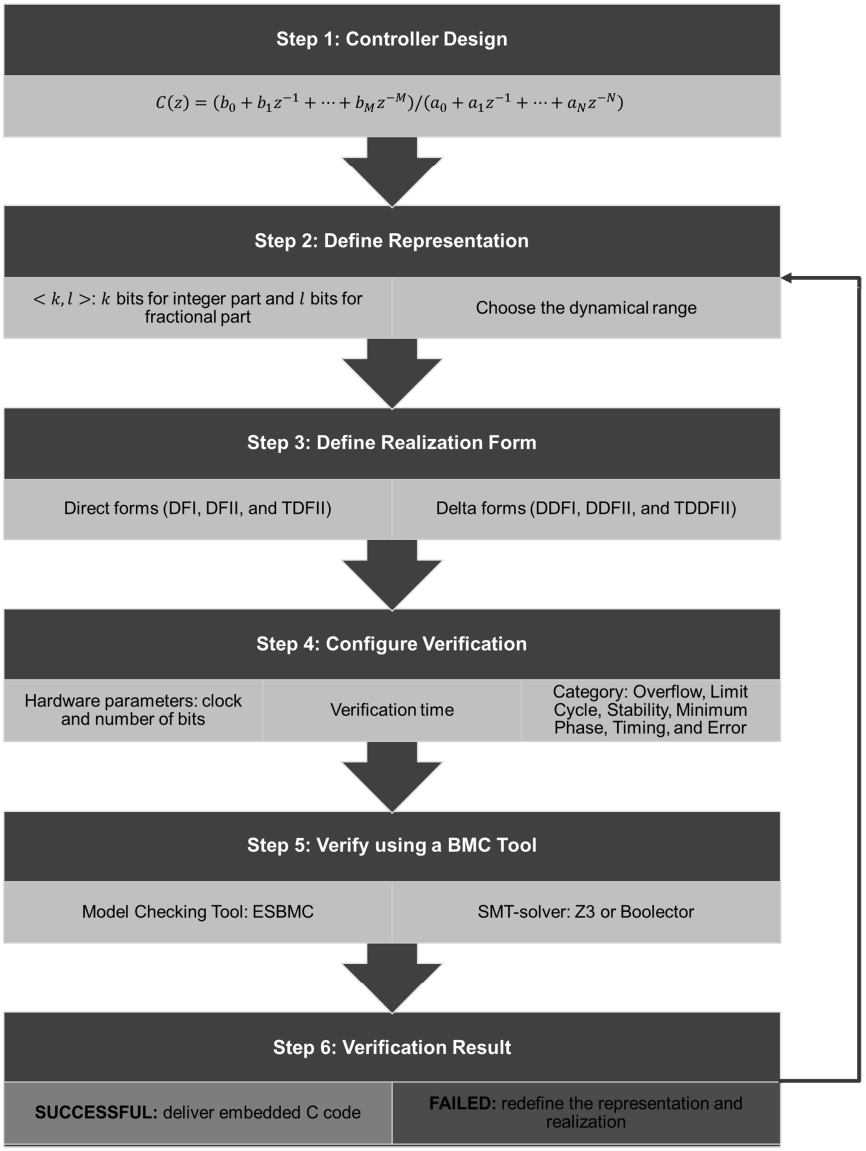

Fig. 4: Proposed methodology for digital controller implementation verification.

itself (impl), which contains the number of bits in the integer (impl.int_bits = 3) and precision (impl.frac_bits = 5) parts, the input range (impl.min = -3 and impl.max = 3), and the scaling factor (impl.scale = 10). The third step is to choose the realization form; for this example, Direct Form I realization (DFI) is chosen.

```
#include<dsverifier.h>

digital_system ds = {
    .a = { 1.0, -0.9 },        /* denominator          */
    .a_size = 2,               /* denominator length   */
    .b = { 1.561, -1.485 },    /* numerator            */
    .b_size = 2                /* numerator length     */
};

implementation impl = {
    .int_bits = 3,             /*  integer bits        */
    .frac_bits = 5,            /*  precision bits      */
    .min = -3.0,               /*  minimum input       */
    .max = 3.0,                /*  maximum input       */
    .scale = 10
};
```

Fig. 5: A digital-system verification input file for DSVerifier.

In Step 4, the configuration must be set. As example, the limit cycle occurrence is verified, with a timeout of 1 hour and a bound of 10 cycles. These parameters must be provided when the DSVerifier is called, as follows:

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

Note that DSVerifier is called using Boolector (Step 5). After a few seconds, DSVerifier concludes the verification process, indicating a failure together with the counterexample (Step 6). As a consequence, the designer has to go back to Step 2 to avoid the limit cycle reported by DSVerifier. In this particular example, a simple change of realization, *e.g.*, adopting the Direct Form II (DFII) in Step 3, can rectify the controller implementation. For this realization, the verification in Step 6 is successfully completed.

The DSVerifier might also be used via Graphical User Interface (GUI), which is available in the DSVerifier website [3]. More details about the GUI and command-line usage of DSVerifier can be found in Ismail *et al.* [87].

3.3 Arithmetic Overflow Verification

When dealing with fixed-point arithmetic processors, one needs to take care about overflows, which is difficult to detected without computational tools, as they usually occur at run-time during the quantization process. In the present study, assertions are coded in the quantizer block, and the verification engine is configured to use nondeterministic inputs in the specified range to detect overflows in a digital controller, for a given fixed-point word-length. For any addition or multiplication result, during controller operation, if there exists a value that exceeds the range representable by the fixed-point, an assert statement detects that as an under or overflow violation. As a consequence, a literal $l_{signed\_overflow}$ is generated, with the goal of representing the validity of each addition and multiplication operation,

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

n	1	2	3	4	5	6	7	8
$x(n)$	0.9375	0.1250	0.6875	0.7500	0.9375	0.5000	1.0000	-1.0000
$y(n)$	0.9375	-0.6875	0.4375	-0.1250	-0.3125	-1.3125	-1.0000	-4.0625

Table 1: An example of overflow for the digital controller represented by (13).

according to the constraint

$$l_{signed\_overflow} \Leftrightarrow (FP \geq MIN) \wedge (FP \leq MAX), \tag{12}$$

where FP is the fixed-point approximation, for the result of adders and multipliers, and MIN and MAX are the minimum and maximum values that are representable for the given fixed-point bit format, respectively. Therefore, in the overflow verification, it can never happen that an expression of a fixed-point type is evaluated to a value that is not in the range of the fixed-point bit format.

An example of arithmetic overflow failure can be seen in the following digital controller

$$C(z) = \frac{1.0 - 2.819z^{-1} + 2.637z^{-2} - 0.8187z^{-3}}{1.0 - 1.97z^{-1} + 1.033z^{-2} - 0.06068z^{-3}}. \tag{13}$$

If this controller is implemented in direct form 1, with 7 bits $< 3, 4 >$ (*i.e.*, 3 bits for the integer part and 4 bits for the fractional part) and with the input range limited to $[-1, 1]$, the controller will present an overflow output of -4.0625 if a input sequence shown in Table 1 is given. However, the number of bits for this digital controller is chosen based on a conservative method, which is the impulse response sum, described by Carletta [11]. The impulse response sum $\sum_{k=-\infty}^{\infty} |h_k|$ for this digital controller is given by

$$\sum_{k=-\infty}^{\infty} |h_k| = 2.30172. \tag{14}$$

Since this controller is stable, Carletta *et al.* [11] show that the maximum value for the output is given by

$$||y||_\infty \leq x_{max} * \sum_{k=-\infty}^{\infty} |h_k| = 1 \times 2.30172 = 2.30172, \tag{15}$$

Note that the maximum value chosen for this controller is greater than that obtained by means of Eqs. (14) and (15). However, the instantaneous change in the input leads to an overshooting that exceeds the maximum representable value, without turning the system into unstable. One can easily see that the application of BMC techniques to digital controllers can detect overflow failures that are not detected by conservative methods.

3.4 Limit Cycle Verification

To verify the presence of limit cycles, in a particular fixed-point controller realization, the quantizer block routine is configured by setting a flag variable on it to enable the wrap around on overflows, which means that the verification engine is not expected to detect overflow failures, as in the previous case. Additionally, the controller is configured to use a zero input signal and a nondeterministic initial state, for previous outputs. The controller 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.

The zero input limit cycle occurrence is represented by a literal l_{LCO}, with the goal of determining whether a set of previous outputs states is found, according to the constraint

$$\begin{aligned} l_{LCO} \Longleftrightarrow \exists n, k \in \mathbb{N} | x_m = 0 \Longrightarrow \exists y_{k+i} = y_{k+n+i}, \\ \forall i \in \{0, 1, 2, ..., n\}, m \in \{k, k+1, k+2, ..., k+2n\}. \end{aligned} \tag{16}$$

The limit cycle absence is then verified by checking $\neg l_{LCO}$, that is, there exists no execution of the digital controller on which a set of previous outputs states is found.

An example of limit cycle failure is shown in Fig. 6. This is a digital controller (see the test case 11)[1] in DFI realization, with output range of $[4, 4]$, and with fixed-point representation $< 2, 13 >$. The verification engine checks for the limit cycle occurrence (l_{LCO}) and then gives the following counterexample: if the system receives a zero-sequence, following a $\{2, 2, 2, 2\}$ sequence of past outputs, the limit cycle will occur, as shown in Fig. 6. In this figure, a simulation with 2 seconds of duration is shown, reproducing the counterexample provided by the verification engine.

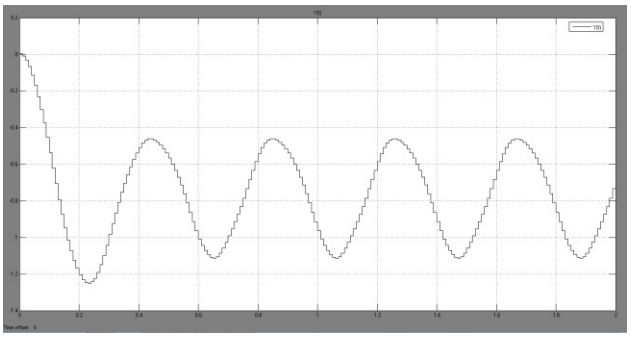

Fig. 6: Limit Cycle in a Digital Controller.

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

3.5 Time Constrains Verification

The sample time is a very important parameter to be chosen in a digital control system. In particular, all the system's dynamic is changed with a modification in the sample time. A precise selection of sample period is thus essential for a computer-controlled system. On the one hand, too short sample times require a higher performance and consequently processors with a high clock frequency; this can impose technical limitation in the design of the digital controller. On the other hand, too long sample times do not permit the reconstruction of continuous signals [83].

In principle, the sample time choice depends on the physical plant, where the control system is applied. The right choice of the computational implementation of a controller may thus reduce the number of arithmetic operations and consequently the computational costs. As control systems are typically real-time systems, they cannot take more time to process tasks than a sample period. In practical applications, the controller is designed with a reasonable sample period to produce good simulations results. Thereafter, it is implemented in a computer-based system, where samples are scheduled at every sample period; this is the maximum time that the processor takes to perform all control tasks and corresponding operations. If an operation cannot terminate on time, then the results might not be correct and the control system might not work as expected.

For this particular reason, a time constraint verification for a particular processor, *i.e.*, central processing unit (CPU) time verification, becomes a very useful controller design tool, which may indicate if the chosen sample period and the computational realization are compatible, before the physical implementation, thus avoiding serious malfunctions of the system. As a result, the needed time to execute a specific code can be roughly estimated (discounting I/O time, other jobs' shares), once each instruction can be broken into a set of assembly instructions; in particular, every processor has a table of clock cycles spent on each assembly instruction. Currently, DSVerifier supports digital controllers realizations implemented and compiled to run on a MSP430G2231, which is an ultra-low-power 16-bit RISC-CPU based microcontroller [84].

Since the loops of the controller implementation are bounded, the instruction count for a particular (controller) realization form is computed by means of BMC technique [85]; additionally, given that the cycles per instruction and clock cycle time are known parameters, it is assumed here that the timing behavior is repeatable and predictable [86]. Actually, DSVerifier has been designed to support different processors based on the Instruction Set Architecture [87].

So, with the assembly file generated from the compilation, this can be compared with the source code, with the goal of identifying instructions for each segment of the program. Then, a worst case execution time (WCET) analysis can be performed in the controller function, considering the number of cycles required to execute instructions and iterations. Kim *et al.* [88] describes a method using static analysis and the model-checking technique to check for program-segment timing, similarly to what is done here.

As an example, the code fragment shown in Figure 7(a) is used to perform multiplications involving coefficients b_k and previous entries, in Eq. (1). Figure 7(b) shows the code in Figure 7(a) converted into some assembly instructions, using the compiler CCS v4 [89].

```
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. 7: (a) C code fragment of the digital controller. (b) Assembly instructions of the code fragment shown in (a).

One can then realize that each instruction takes a different number of clock cycles to execute; 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 62.5 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 estimated, based on the sample rate of the system; for instance, if it operates using a sample rate of 48 KHz, then it means that after each 20.8μs window, new data are obtained from the system input, and the controller function has to process output samples within this time. Formally, a literal l_{CPU_time} is generated to represent the validity of the CPU execution time, with a constraint

$$l_{CPU\_time} \Leftrightarrow ((N \times T) \leq D), \tag{17}$$

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

It is worthwhile to mention that once the instruction count for a particular controller implementation is provided, the time constrain verification is automatically performed by DSVerifier. In particular, the procedure illustrated in Figure 7 is carried out for every code line of the controller implementation, *i.e.*, C code expressions are translated into assembly instructions by the compiler just once; after that, the total number of instructions is automatically computed by means of BMC. Note further that the CPU execution time spent by the controller is also automatically computed in DSVerifier, by multiplying the number of cycles by the cycle time, as described in Eq. (17). The execution time computed by DSVerifier is actually an estimation that considers only CPU execution time, and discounts time losses due to cache, jitter, and scheduling; further improvements could be considered for a more realistic verification.

row	z^n	z^{n-1}	...	z^{n-k}	...	z^1	z^0
1	a_n	a_{n-1}	...	a_{n-k}	...	a_1	a_0
2	a_0	a_1	...	a_k	...	a_{n-1}	a_n
3	b_0	b_1	...	b_{n-k}	...	b_{n-1}	0
4	b_{n-1}	b_{n-2}	...	b_k	...	b_0	0
5	c_0	c_1	...	...	c_{n-2}	0	0
6	c_{n-2}	c_{n-3}	...	...	c_0	0	0
...	...	...	...	...	...	...	...
$2n-1$	r_0	0	0	0	0	0	0

Table 2: Digital controller denominator coefficients distributed in a Jury's table.

3.6 Stability Verification

The stability is a basic requirement, but is very important during designing the digital controller. In particular, digital controllers are updated every sampling period and one has to ensure that the system will be stable during its execution. A discrete system is stable if all its poles are in the interior region of the unitary circle of z-plane (*i.e.*, the poles must have the module less than one) [32].

In previous studies [26, 79], stability verification using Schur Decomposition is used and implemented in the ESBMC tool using the Eigen Library [90]. Such a method, however, involves many matrix operations that make it computationally expensive. In this study, we choose another method to check for stability. We use the Jury's Stability criteria [91], as it is computationally less expensive than the Schur Decomposition and does not require the use of an external library. The advantage of the Jury's algorithm can easily be observed via its complexity, which is $O(n^2)$, whereas the complexity of the previous stability verification, based on the Schur decomposition, is $O(n^3)$ [90].

The Jury's algorithm can be used for a given polynomial of the form

$$F(z) = a_n z^n + a_{n-1} z^{n-1} + ...a_1 z + a_0 = 0, a_n > 0, \tag{18}$$

where a_n until a_0 represent the digital controller denominator coefficients. In particular, these coefficients are distributed in a Jury's table using the format shown in Table 2.

Considering the Jury's table as a matrix m with dimensions $[2n-1][n]$ where n is the number of coefficients. We have some considerations for the algorithm:

1. The first line of matrix m has the digital controller denominator coefficients.
2. The even number lines have the inverse order of their previous lines (*i.e.*, despising final zeros).
3. The b_0 is in line 3 of column 1 and its value is $b_0 = m[3-2][1] - (m[3-2][p]/m[3-2][1]) * m[3-1][1]$.
   Generalizing b_j and c_j leads to $m[i][j] = m[i-2][j] - (m[i-2][p]/m[i-2][1]) * m[i-1][j]$, where p is the last nonzero column number for line $i-2$.
4. The line $2n-1$ has only one element nonzero in the first column.

With the Jury's table filled in, it is necessary to check the stability using the following two definitions:

row	z^2	z^1	z^0
1	1.0	1.068	0.1239
2	0.1239	1.068	1.0
3	0.984649	0.935675	0
4	0.935675	0.984649	0
5	0.095512	0	0

Table 3: Jury's table of the digital controller represented by Eq. (19) using denominator coefficients.

Definition 2 *If the element* $m[1][1]$ *is positive, then F(z) will be stable iff all the first elements in odd lines (i.e.,* $m[3][1]$*,* $m[5][1]$*, ...,* $m[2n-1][1]$*) are positive too.*

Definition 3 *If the element* $m[1][1]$ *is negative, then F(z) will be stable iff the first elements in odd lines (i.e.,* $m[3][1]$*,* $m[5][1]$*, ...,* $m[2n-1][1]$*) are negative too.*

As a running example, we check for the stability of the following digital controller, which is extracted from our benchmarks:

$$H(z) = \frac{2.813z^2 - 0.0163z - 1.872}{z^2 + 1.068z + 0.1239} \tag{19}$$

This digital controller has the Jury's table shown in Table 3. Analyzing Table 3 and considering Definition 2, one can easily conclude that the digital controller represented by Eq. (19) is stable, once $m[1][1]$, $m[3][1]$, and $m[5][1]$ are positive numbers.

The stability verification condition, according to the Jury's criteria, is represented by a literal $l_{stability}$ with the following constraint

$$\begin{aligned} l_{stability} \Longleftrightarrow ((sgn(m[i][1]) \neq sgn(m[i+2][1])) \vee \\ (sgn(m[i][1]) = sgn(m[i+2][1]))), \forall i, \end{aligned} \tag{20}$$

where $sgn(\bullet)$ is a function that indicates the operand signal.

3.7 Minimum Phase Verification

A minimum phase system is defined by a stable system with all the zeros stable. Conceptually, a minimum-phase system has all poles and zeros inside the unitary circle [91]. Minimum-phase is a desirable property in digital controllers because in a closed-loop system, a feedback-controlled system shows the controllers zeros as the general system's poles (*i.e.*, a digital control system with a nonminimum-phase controller is potentially unstable). The verification engine for this particular property is similar to the stability verification and also uses the Jury's stability test, but instead of using digital filter denominator coefficients, we use digital filter numerator coefficients to check for the minimal phase.

As a running example, we check whether the digital controller represented by Eq. (19) has minimum phase. The Jury's table using numerator coefficients, for this controller, is shown in Table 4:

row	z^2	z^1	z^0
1	2.813	−0.0163	−1.872
2	−1.872	−0.0163	2.813
3	1.567218	−0.027147	0
4	−0.027147	1.567218	0
5	1.566748	0	0

Table 4: Jury's table of the digital controller represented by Eq. (19) using numerator coefficients.

Analyzing the Jury's table above and considering Definition 2, one can conclude that the digital controller represented by Eq. (19) has minimum phase, once $m[1][1]$, $m[3][1]$, and $m[5][1]$ are positive numbers. Note that the minimum phase is checked similar to the stability verification.

3.8 Stability and minimum phase verification for delta-form digital controllers

The Jury criteria are easy and is a useful method for verifying convergence in discrete time polynomials in z-domain. However, these criteria do not provide any guarantee for polynomials in δ-domain. For that particular reason, a different stability criteria has to be established for the stability verification of controllers implemented in delta forms. These criteria are described by Definitions 4 and Lemma 3:

Definition 4 *A linear dynamical system represented by Eq. (4), of the N-th order, is asymptotically stable iff its poles $(\lambda_1, \lambda_2, ..., \lambda_N)$ is a convergent sequence that satisfies the property (the proof is provided by Feue and Goodwin [92])*

$$lim_{k\to\infty} \sum_{i=1}^{N} c_i (1 + \Delta\lambda_i)^k = 0, \tag{21}$$

where c_i is a complex non-zero coefficient.

Analyzing Eq. (21), one can conclude that the sum is convergent only if the exponential part presents convergence. Thus, the stability can be verified as

Lemma 3 *A digital system in delta form modeled by Eq. (4) is asymptotically stable iff all the eigenvalues $\lambda_i, i = 1, 2, ...N$ (either multiple or single) satisfy*

$$|1 + \Delta\lambda_i| < 1. \tag{22}$$

The same idea can be applied for minimum phase verification, but must be applied to the numerator polynomial.

4 Experimental Evaluation

The experimental evaluation of our study consists of three parts. Section 4.1 describes the mathematical models of the plants and summarizes the characteristics of the digital controllers for the respective plants. Section 4.2 describes the experiment's configuration, whereas Section 4.3 analyzes and summarizes the experimental results.

4.1 Digital Controllers' Design

We designed 33 digital controllers for the verification of three different plants, where 16 of these controllers are designed for a commercial ball and beam plant, which has the following mathematical model

$$G_1(z) = \frac{1.0067 \times 10^{-8}(z + 9.256)(z + 0.9324)}{(z - 1)^3(z - 0.7041)}. \tag{23}$$

where the sample time is $0.01s$. Other 8 controllers are designed for an A/C motor plant, extracted from Ogata [83] and described by

$$G_2(s) = \frac{1}{s(s + 1)}. \tag{24}$$

This plant is discretized by different sample rates. Finally, 9 controllers are designed for another plant, also extracted from Ogata [83],

$$G_3(s) = \frac{1}{s(s + 0.4)} \tag{25}$$

discretized by different sample times too. The FWL format of the controllers is chosen via the methodology presented by Carletta [11], which is based on impulse response sum ($\sum h(k)$).

Other digital controllers provided by Harnefors [10] are also considered in verifications. These are resonant controllers, with applications for voltage converters, designed in the delta form. Resonant controllers are specially sensitive to the FWL effects, because they present poles next to the stability limit.

4.2 Experimental Setup

For the following verifications, the hardware used for the controller is a 16-bits micro-controller with a clock rate of 16 MHz. This study employs ESBMC v1.24[1], with the SMT solver Boolector v2.0.1[2]. To prevent overflows, scaling factors are considered during the implementation. All controllers are checked against five types of properties, as described in Section 3. In this study, the performance of the delta-form realizations using different delta values is compared to the direct-form realizations. As a result, all test cases are verified in nine different realizations, which include: Direct Form I (DFI), Direct Form II (DFII), Transposed Direct Form II

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

[2] Available at http://fmv.jku.at/boolector/

(TDFII), Delta Direct Form I (DDFI), Delta Direct Form II (DDFII), and Transposed Delta Direct Form II (TDDFII), where each delta form was implemented using delta values 1/4 and 1/8, respectively.

DSVerifier is called for each benchmark, as follows:

`dsverifier <file> --realization <i> --property <j> --x-size <k>`

where $< file >$ is the digital-system specification file, $< i >$ is the chosen realization form, $< j >$ is the property to be verified, and $< k >$ is the verification bound, *i.e.*, the number of times the digital system will be unwound.; the test cases are not determined successful or failed if a timeout occurs. The experiments are executed in a computer with the following hardware configurations: Intel Core i7 − 2600 3.40 GHz processor, 32 GB of RAM, Fedora 17 Beefy Miracle 64-bits OS.

4.3 Experimental Results

Approximately 1100 verifications are performed with the digital controllers designed in the previous subsection. Here, the experiments are split into five verification categories: overflow, limit cycle, stability, minimum phase, and timing. Three different implementations are considered: direct forms (DFI, DFII, and TDFII) and the corresponding delta forms with two Δ values (delta 1/8, where $\Delta = 0.125$, and delta 1/4, where $\Delta = 0.25$).

Figure 8 shows the results for overflow and limit cycle verification using direct and delta realizations, where the horizontal axis represents properties divided into three groups of realizations, the first one is the direct realizations, whereas the second and third ones are the delta realizations using 1/4 ($\Delta = 0.25$) and 1/8 ($\Delta = 0.125$), respectively. The vertical axis represents the property violations percentage (%) for DFI, DFII, and TDFII, which is found by DSVerifier, for each realization group. In particular, the horizontal axis shows groups of realizations (*i.e.*, direct, Delta 1/4, and Delta 1/8) and the vertical axis shows the failure percentage for each realization form (*i.e.*, DFI, DFII, and TDFII) in three different groups of realizations. As a result, the combination of horizontal and vertical axis show nine different implementations (*e.g.*, Direct DFI, Delta 1/4 DFI, and Delta 1/8 DFI) to be checked for overflow and limit cycle.

For each property, we executed 99 tests (*i.e.*, 33 for direct, 33 for delta 1/4, for 33 in delta 1/8). According to Fig. 8, DSVerifier can detect several errors in direct and delta forms.

For overflow verification, DSVerifier found 80% of violations in direct form. Using $\Delta = 1/4$, this number decreases to 56%, a failed reduction of 24%. However, if we consider $\Delta = 1/8$, the violation percentage reduces to 52% (*i.e.*, 28% less violations than in direct form). With respect to realizations, the TDFII forms (both delta 1/4 and delta 1/8) presented the best results (only 13% of overflows failures), where the number of failures decreased to 54% if compared to the representation with the worst results (*i.e.*, DFI with 28% of overflow failures). For limit cycle verification, DSVerifier found 71% of violations in direct form. For $\Delta = 1/4$ and $\Delta = 1/8$, this number decreases to 67% and 65%, respectively.

Figure 9 shows the stability and minimum phase verification results. The horizontal axis represents the stability and minimum phase properties, which is grouped by the realization of the controllers. Vertical axis shows the success and

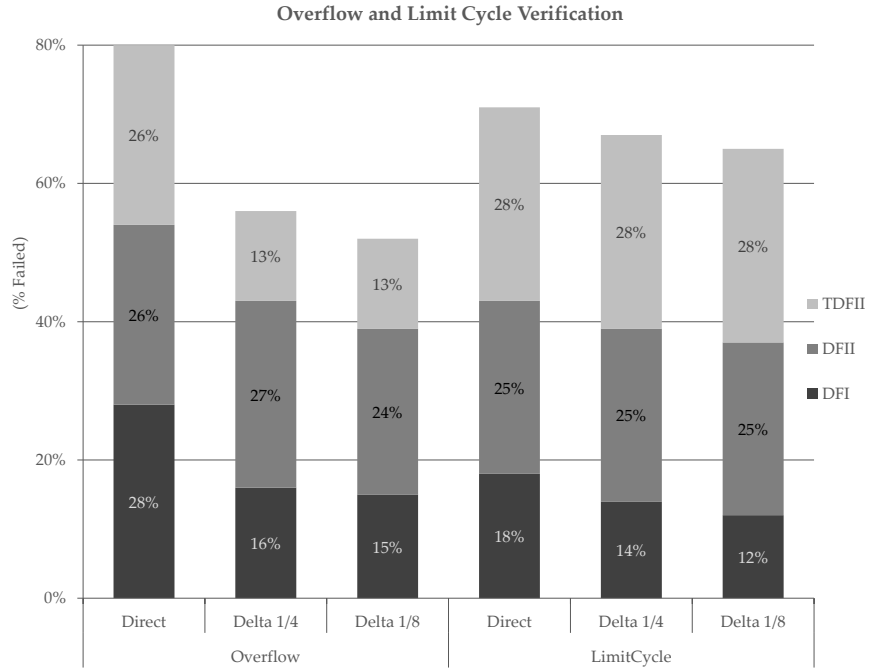

Fig. 8: Verification Results for Overflow and Limit Cycle using Direct and Delta Realizations.

failures percentage. Differently from Fig. 8, this graph does not produce results by each form because the results are based on the controller coefficients only. According to the verification results, 15% of the benchmarks had stability problems in direct form. After delta transformation, using $\Delta = 1/4$, this number decreases to 9% of the benchmarks (a reduction of 6%), but changing the delta value to 1/8, this number increases to 12%. For the minimum phase property, the direct form presented 52% violations. Applying delta to 1/4 and 1/8, this number decreased to 36% in both cases.

The time constraints property verification is not shown in any graph, as all tests returned success for direct form, delta 1/4, and 1/8 realizations. In particular, time constrains verifications do not present failures, because the order of the controllers is relatively low and the sample time is reasonably high.

Figure 10 shows a comparison of the verification time between SMT-Solvers Z3 v4.0 and Boolector v2.0.1. Note that the overall verification time for Z3 is 391352 seconds, whereas for Boolector, it is 175585 seconds. This difference represents a reduction of 55% of the total verification time, which significantly reduces the number of timeouts (*i.e.*, DSVerifier cannot provide a conclusive response within the time limit).

The verifications results show that various factors can influence the performance of the digital controllers: number of bits, sample time, implementation form and domain, and others that are not within the scope of this study. The optimal and failures-free design, taking into account all these factors, is a very

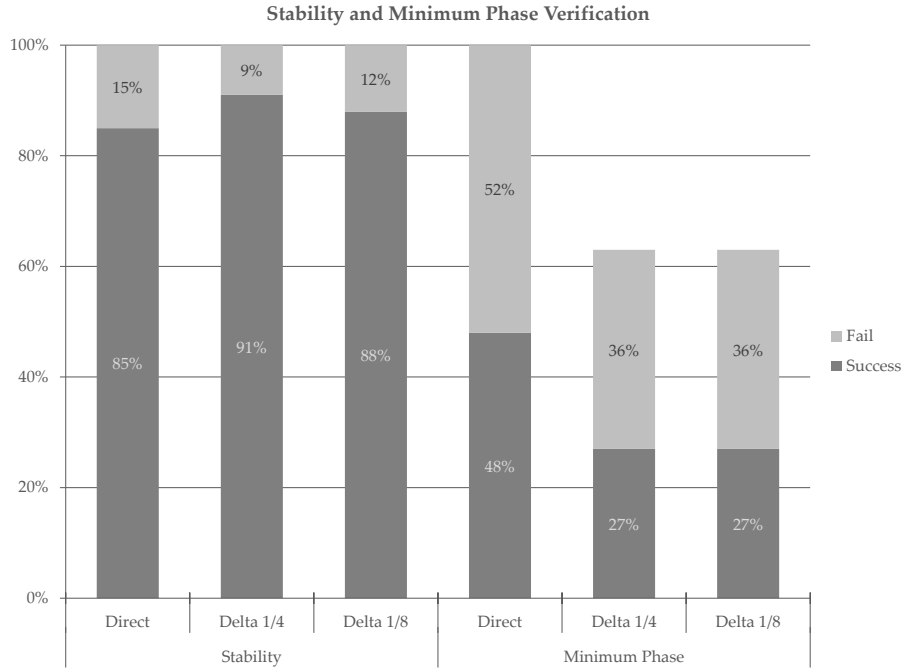

Fig. 9: Verification Results of Stability and Minimum Phase in Direct and Delta Realizations.

challenging design task, even using a consolidated control theory for digital control systems design. The proposed methodology can give a strong support for the control designer with the aid of a reliable and efficient verification tool, which allows reaching a safe and reliable implementation of digital controllers by means of a verification process.

Note that changes to the parameters can impact in the system performance (*e.g.*, the Δ value in delta-form realizations). The results show that lowers values of Δ can provide better round-off performance, resulting in less fragile controllers (*i.e.*, with less implementation failures). All experimental results show better performance of delta controllers when compared to direct realizations, and better performance of $\Delta = 1/8$ when compared to $\Delta = 1/4$. However, excessive reduction of the Δ demands more bits for the fractional part representation of controllers' coefficients; that will present lower numbers with greater orders of the controllers.

There is clearly a trade-off between improved performance offered by the delta representations and the number of bits demanded. Figs. 12 and 13 show the effect of the Δ in the coefficients of the numerator and the denominator of a digital controller. It shows the number of underflows in these coefficients represented by double precision (*i.e.*, the truncation to zero of these values). With a value of $1/64$ for Δ (less than a sample period) almost 60% of the digital controllers present underflows in coefficients, and this amount would grow to three-quarters of the controller if the double precision is not applied. Most processors are provided with double-precision registers, which can be used for saving coefficients of the

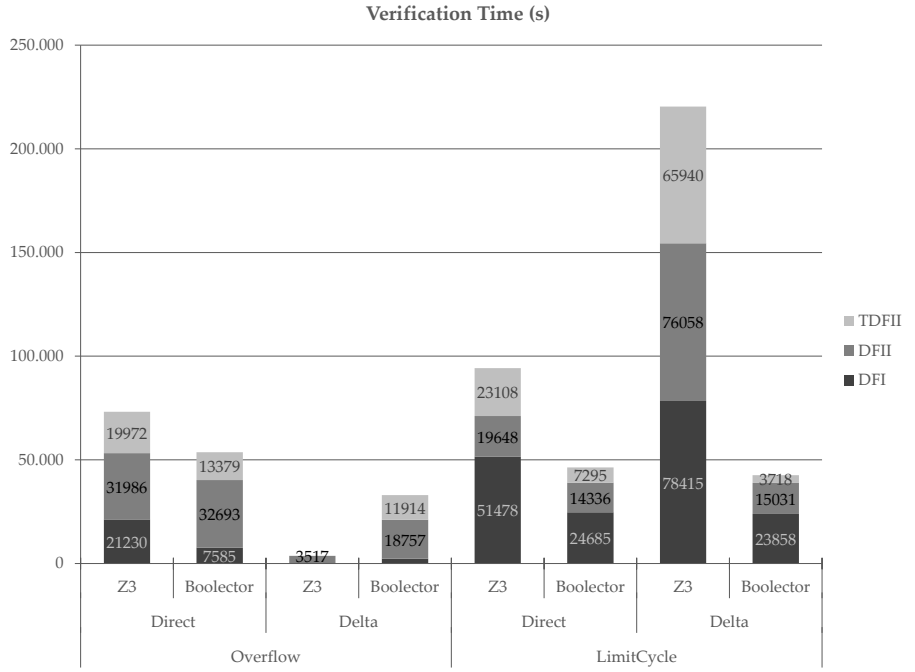

Fig. 10: Verification Time of all benchmarks.

digital controllers, thus improving their precision, but double-precision registers are useless for controllers with order greater than three, and for this reason the graphics in Figs. 12 and 13 shows that elevated number of underflows.

The experimental results also show that DSVerifier is a useful design tool to determine the most optimized fixed-point structure realization for digital controllers. For instance, the results for test case 9, a digital controller for the ball and beam Quanser plant, show that a control engineer can easily conclude that the controller should be implemented in the TDFII or delta forms to avoid LCOs, once this controller failed in the LC verification for DFI and DFII, but it passes in TDFII or delta forms. Note that some failures that appear in the counterexamples are difficult to be detected by simulation tools, similar to limit cycles, which only occur for some few input combinations.

As example, one can analyze in SIMULINK the stability of a closed-loop control system using the following digital controller represented by Eq. (26) and conclude that the closed-loop system is stable. This controller is designed by emulating and mapping analog poles and zeros with the following zero-poles-gain representation

$$C(z) = \frac{50[(z-1)^2(z-0.81873)]}{(z-0.9704)(z-0.9329)(z-0.067032)} \tag{26}$$

Two zeros on 1 can be observed in this controller to cancel two poles in 1 of the ball and beam plant, which then stabilizes the closed-loop system. When this closed-loop system is simulated, the poles and zeroes cancellation occurs and the systems response is acceptable (the step response is shown in Fig. 14). However,

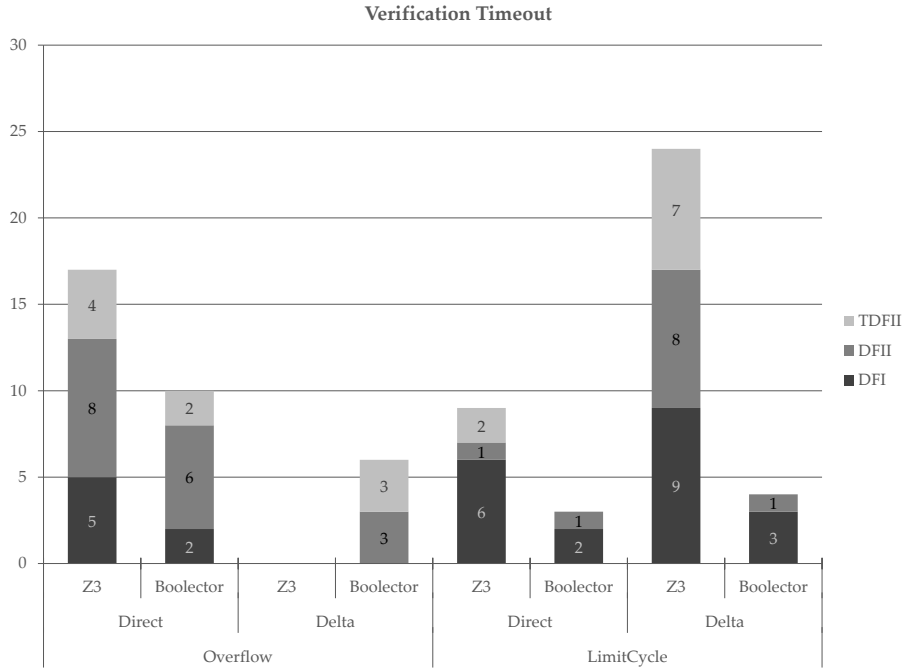

Fig. 11: Number of timeouts for Overflow and Limit Cycle properties.

if the transfer function with quantized coefficients is simulated, then the response is totally different (see Fig. 15).

When the closed-loop system model is formally verified, the stability test fails because of the noncancellation of unstable poles on 1; note that the cancellation does not occur because of errors caused by the FWL effects. The standard SIMULINK/Matlab operations do not consider the quantization effect, and the design tools in the Control System Toolbox do not consider the controllers fragility, and these harming effects of quantization cannot be detected easily using these tools. After the quantization effects, the proposed methodology and tool can ensure the correctness after the quantization effects with a formally verified implementation of a designed digital controller.

5 Related Studies

The FWL effects in digital controllers are well-known in the control systems literature. Most control systems researchers tried to prevent these problems with an additional effort in the design phase, as pointed out in Section 2. These efforts involve the implementation of nonfragile controllers and the pursuit for the optimal FWL format. An overview about these techniques can be seen in Istepanian and Whidborne [1]. Indeed, only a few researchers developed tools to verify the occurrence of FWL effects; however, most initiatives have focused their effort on how to prevent them. There are particular examples of the application of formal

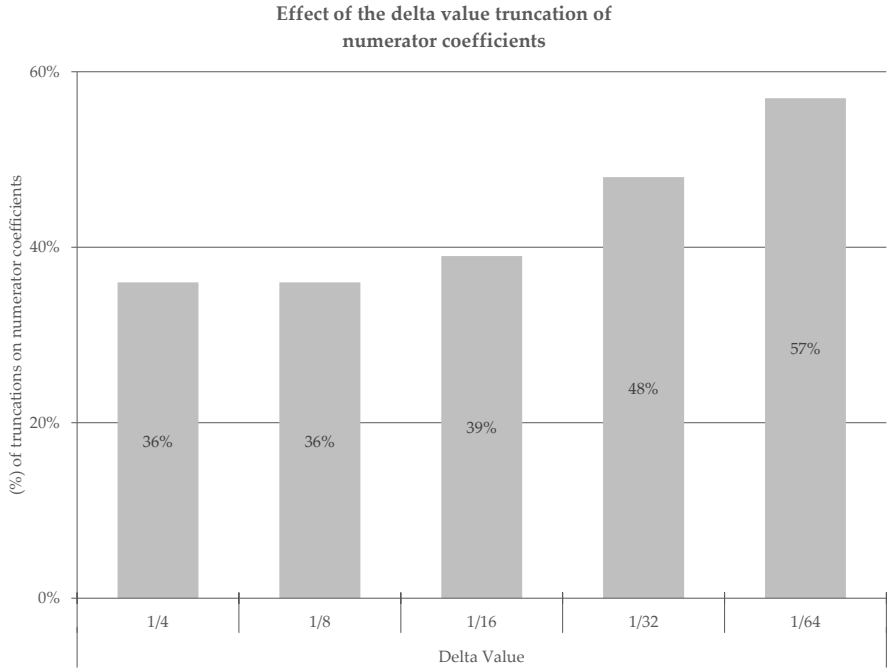

Fig. 12: Results of truncation after delta realization for numerator coefficients.

methods for monitoring and diagnostics of control systems, but they are usually applied to high-level properties related to real-time specifications. In fact, tests and simulations are the most used techniques in control systems, but they are difficult to ensure correctness, as only some system behaviors are explored (coverage problem).

An example of simulation tool is proposed by Sung and Kum who developed a tool to determine the minimum bound of the word-length fixed-point representation via simulation techniques [14]. Similar to any other simulation tool, it only offers partial validation, as not all scenarios are explored; as a result, some FWL-related problems might be ignored. Another interesting study is presented by Anta *et al.* [93], where a tool called Costan finds errors in the implementation of a mathematical model and verifies whether the error is tolerated, considering the quantization effects and the fixed-point implementation. The authors focus their verification on the stability of the system. Most recently, Luengo *et al.* present a simulation tool, which is enhanced with the Monte Carlo algorithm to detect and predict limit cycles in digital filters [23]. This study is a very interesting alternative in terms of efficiency and time, but probabilistic methods might also present errors.

Additionally, there are several related studies about the development of tools to evaluate the performance of digital controllers and filters, estimate their round-off noise, and pursue optimal implementation with respect to FWL effects [94–99].

Alur *et al.* [100, 101] introduced the earliest application of model checking for digital systems, represented by timed automata. Those influential studies inspired

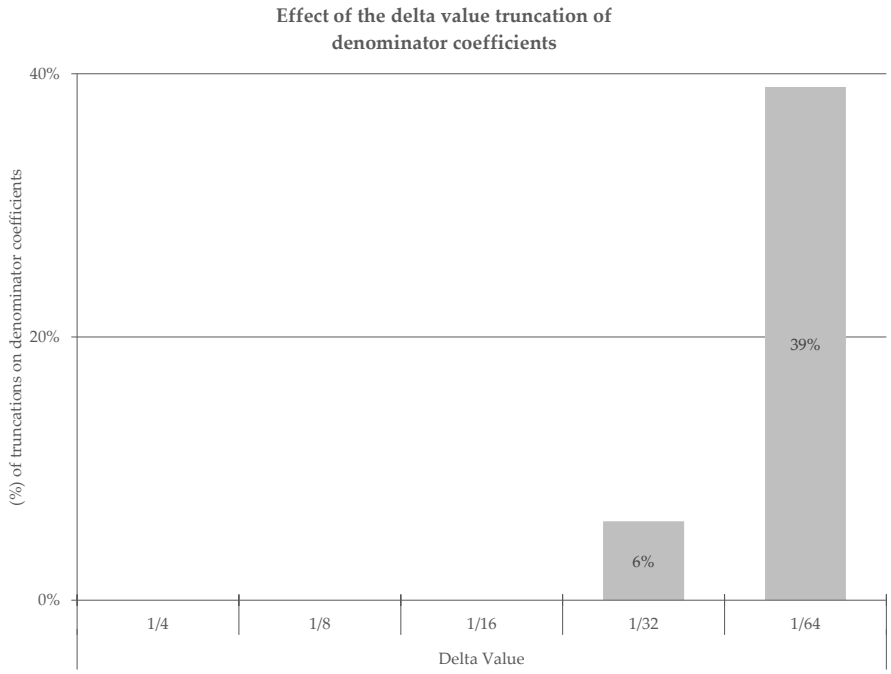

Fig. 13: Results of truncation after delta realization for denominator coefficients.

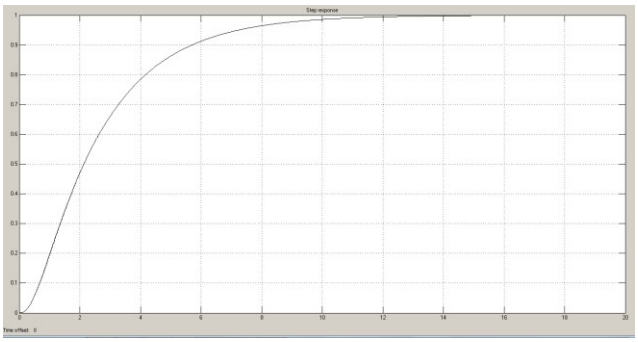

Fig. 14: Step response without quantization effects.

the development of various model checking tools for verifying hybrid automatas and real-time systems. An example is the UPPAAL [17], which is a model checker based on the theory of timed automata and it is designed to verify systems that are modeled via a timed automata network; this tool has a large and successful application in communication protocols verification. Another similar tool is the Open-Kronos [18], which is able to check for the reachability of timed automata and the emptiness of timed Buechi automata. The CPN tools [19] are also used to verify systems modeled via a colored (timed and untimed) Petri Net. However, such approaches are usually employed for high-level verification and have

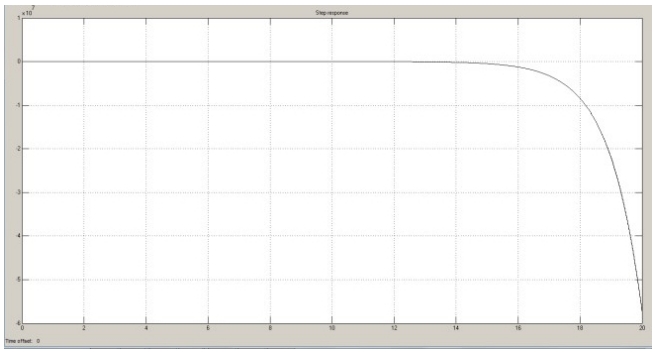

Fig. 15: Step response with quantization effects.

not been used for verifying resilience, i.e., system robustness related to implementation aspects [87]. On the one hand, state-of-the-art BMC tools are able to directly handle the C implementation of digital controllers; they are thus suitable for verifying low-level properties (*e.g.*, overflow and limit cycle), which depend on the hardware architecture. On the other hand, verification tools such as UPPAAL would demand an additional effort to model the implementation aspects of the controllers by means of (timed) automata. Thus, one may notice there is still a gap, regarding verification tools and methodologies to check for implementation aspects of embedded systems.

Some particular examples of the use of BMC techniques to control systems may be cited. Dutertre *et al.* show the advantages of formal methods over traditional debugging tools [102]. The authors use SMT-based tools to diagnose and monitor aircrafts systems. Simko and Jackson demonstrate that digital controllers can be formally verified with a combination of SMT solving (to verify the control software) and Taylor models (to predict the continuous plant dynamic) [103]. In Prabhu and Dasgupta [104], model-checking is used in discrete controllers that only react to specific events and that can be represented via a finite-state machine. The authors use a combination of SMT solvers and existing industrial model-checking tools.

Some recent studies use SMT-based BMC techniques to verify properties of digital filters and controllers. Cox *et al.* [24, 25] show that simulation tools are useful, but insufficient to detect design errors. The authors propose the use of SMT-based bounded and unbounded tools to verify digital filters. The BMC tool used in this study, called ESBMC, has been recently used to verify discrete-time systems. In Abreu *et al.* [79], properties of digital filters are verified using ESBMC, where overflows, limit cycles, time constrains, stability, and frequency response are checked.

Recently, Bessa *et al.* [26] apply SMT-based BMC to verify overflows, limit cycles, and time constrains in digital controllers. Here, an extension of Bessa *et al.* [26, 27] is presented, which applies SMT-based BMC techniques to verify FWL effects on a wide variety of digital controllers properties and realization forms. To the best of our knowledge, there is no other verification tool that can be directly used to check for the different digital controllers properties and realization forms, described in this study, considering implementation aspects. In fact, this is the first work to describe and evaluate a comprehensive SMT-based BMC procedure

to support the checking of various digital controllers implementations, taking into account the platform in which the (embedded) system software operates.

6 Conclusion

The contributions of this study can be split into three main categories: the introduction of BMC techniques in digital control systems validation and verification; the analysis and evaluation of implementation of digital controllers by taking into account variables and parameters that may affect their performance; and the proposal of a novel verification methodology for a safe design of digital controllers.

Firstly, this article shows the potential of an SMT-based BMC approach to verify and validate digital controllers. An extensive verification of different properties and realization forms of digital controllers is carried out. The ESBMC tool, used in this study, is able to provide conclusive results about the absence of failures in the system, with reliable results for the successful verification and with error traces for the failures, in nearly 98% of the verifications (a timeout reduction higher than 50% when compared to our previous studies [26, 27]). Thus, the use of ESBMC represents an automated and reliable alternative if compared to traditional simulation tools.

Secondly, an extensive number of benchmarks allowed a comparative analysis between direct-form and delta-form realizations for implementation of digital controllers using an SMT-based BMC approach to verify properties that are difficult to be checked with simulation tools. The experimental results show that delta-form realizations present a superior performance when compared to the direct-form realizations, reducing the occurrence of FWL-related errors and preserving the stability and minimum-phase properties, which are important indicators of a system's nonfragility. However, the better performance of delta-form realizations demands more fractional bits to avoid underflows in coefficients representations.

Finally, this study provides a simple and iterative methodology for design verification of digital controllers, supported by the DSVerifier. With the proposed methodology, a controller design can verify, in the design phase, if the proposed digital controller will present an expected behavior when it is embedded in a known hardware architecture, with resource limitations. Note that this approach does not remove all design errors once the verification is performed unrolling the system for a limited number of samples, but it decreases them substantially. The methodology and respective tool presented here were used only to verify digital controllers, but their usage should be expanded for any digital system in future study, including closed-loops systems and design requirements related to the digital control system.

Acknowledgements Part of the results presented in this paper were obtained with the project for research and human resources qualification, for under- and post-graduate levels, in the areas of industrial automation, mobile devices software, and Digital TV, sponsored by Samsung Eletrônica da Amazônia Ltda, under the terms of Brazilian Federal Law number 8.387/91. This research was also supported by CNPq 475647/2013-0 and FAPEAM 062.01722/2014 grants.

7 Conflict of Interest

The authors declare they have no conflict of interest.

References

1. Istepanian R, Whidborne J (2001) Digital Controller Implementation and Fragility: A Modern Perspective. Advances in Industrial Control, Springer, London
2. Yang G, Guo X, Che W, Guan W (2013) Linear Systems: Non-Fragile Control and Filtering. CRC Press, New York
3. Masten M, Panahi I (1997) Digital signal processors for modern control systems. Control Engineering Practice 5(4):449 – 458. DOI doi:10.1016/S0967-0661(97)00024-5
4. Monmasson E, Cirstea M (2007) FPGA Design Methodology for Industrial Control Systems 2014;A Review. IEEE T IND ELECTRON 54(4):1824–1842. DOI 10.1109/TIE.2007.898281
5. Li G (1997) On pole and zero sensitivity of linear systems. IEEE T CIRCUITS-I 44(7):583–590. DOI 10.1109/81.596939
6. Li G (1998) On the structure of digital controllers with finite word length consideration. IEEE T AUTOMAT CONTR 43(5):689–693. DOI 10.1109/9.668838
7. Mantey P (1968) Eigenvalue sensitivity and state-variable selection. IEEE T AUTOMAT CONTR 13(3):263–269. DOI 10.1109/TAC.1968.1098902
8. Middleton R, Goodwin G (1986) Improved finite word length characteristics in digital control using delta operators. IEEE T AUTOMAT CONTR 31(11):1015–1021. DOI 10.1109/TAC.1986.1104162
9. Wu J, Chen S, Chu J (2004) Computing a FWL stability measure for second order digital systems. In: Proceedings of Control, Automation, Robotics and Vision Conference, pp. 1593–1598. DOI 10.1109/ICARCV.2004.1469297
10. Harnefors L (2009) Implementation of Resonant Controllers and Filters in Fixed-Point Arithmetic. IEEE T IND ELECTRON 56(4):1273–1281. DOI 10.1109/TIE.2008.2008341
11. Carletta J, Veillette R, Krach F, Fang Z (2003) Determining appropriate precisions for signals in fixed-point IIR filters. In: Proceedings of Design Automation Conference, pp. 656–661. DOI 10.1109/DAC.2003.1219100
12. Istepanian R, Whidborne J (1999) Multi-objective design of finite word-length controller structures. In: Proceedings of Congress on Evolutionary Computation, pp. 61–68. DOI 10.1109/CEC.1999.781908
13. Mohta V (1998) Finite Worldlength Effects in Fixed-point Implementations of Linear Systems. Dissertation, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science
14. Sung W, Kum KI (1995) Simulation-based word-length optimization method for fixed-point digital signal processing systems. IEEE T SIGNAL PROCES 43(12):3087–3090. DOI 10.1109/78.476465
15. Istepanian RH, Chen S (1999) Stability issues of finite-precision controller structures for sampled-data systems. International Journal of Control 72(15):1331–1342. DOI 10.1080/002071799220146
16. Wo S, Zou Y, Chen Q, Xu S (2009) Non-fragile controller design for discrete descriptor systems. Journal of the Franklin Institute 346(9):914 – 922. DOI 10.1016/j.jfranklin.2009.07.008
17. Behrmann G, David A, Larsen KG (2004) A tutorial on UPPAAL. In: Proceedings of Formal Methods for the Design of Real-Time Systems, LNCS 3185, pp. 200–236. DOI 10.1007/978-3-540-30080-9_7
18. Tripakis S, Yovine S, Bouajjani A (2005) Checking timed buechi automata emptiness efficiently. Formal Methods in System Design 26(3):267 – 292. DOI 10.1007/s10703-005-1632-8
19. Jensen K, Kristensen LM (2009) Coloured Petri Nets - Modelling and Validation of Concurrent Systems. Springer. DOI 10.1007/b95112
20. Magellan (2014) Hybrid RTL formal verification. `http://www.synopsys.com/tools/verification/functionalverification/pages/magellan.aspx`, Accessed 12 September 2014

21. Davis TA, Sigmon K (2005) MATLAB primer (7th ed.). CRC Press
22. Johnson GW (1997) LabVIEW Graphical Programming: Practical Applications in Instrumentation and Control, (2nd ed.) McGraw-Hill School Education Group
23. Luengo D, Oses D, Martino L (2014) Monte carlo limit cycle characterization. In: Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing, pp. 8043–8047. DOI 10.1109/ICASSP.2014.6855167
24. Cox A, Sankaranarayanan S, Chang BYE (2012) A Bit Too Precise? Bounded Verification of Quantized Digital Filters. In: Proceedings of Tools and Algorithms for the Construction and Analysis of Systems, LNCS 7214, pp 33–47. DOI 10.1007/978-3-642-28756-5_4
25. Cox A, Sankaranarayanan S, Chang BYE (2014) A bit too precise? Verification of quantized digital filters. In: International Journal on Software Tools for Technology Transfer 16(2):175–190. DOI 10.1007/s10009-013-0279-9
26. Bessa I, Abreu R, Cordeiro L, Filho JE (2014) SMT-Based Bounded Model Checking of Fixed-Point Digital Controllers). In: Proceedings of 40th Annual Conference of the Industrial Electronics Society, pp. 295–301. DOI 10.1109/IECON.2014.7048514
27. Bessa I, Ismail H, Cordeiro L, Filho JE (2014), "Verification of Delta Form Realization in Fixed-Point Digital Controllers Using Bounded Model Checking," In: Proceedings of Brazilian Symposium on Computing Systems Engineering, pp. 49–54. DOI 10.1109/SBESC.2014.14
28. Brummayer R, Biere A (2009) Boolector: An Efficient SMT Solver for Bit-Vectors and Arrays. In: Proceedings of Tools and Algorithms for the Construction and Analysis of Systems, LNCS 5505, pp. 174–177. DOI 10.1007/978-3-642-00768-2_16
29. De Moura L, Bjørner N (2008) Z3: An Efficient SMT Solver. In: Proceedings of Tools and Algorithms for the Construction and Analysis of Systems, LNCS 4963, pp. 337–340. DOI 10.1007/978-3-540-78800-3_24
30. Oppenheim A, Willsky A, Nawab S (1997) Signals and Systems. Prentice-Hall signal processing series, Prentice-Hall International
31. Proakis J, Manolakis D (1996) Digital signal processing: principles, algorithms, and applications. Prentice-Hall International editions, Prentice Hall
32. Åström K, Wittenmark B (1997) Computer-controlled systems: theory and design. Prentice Hall information and system sciences series, Prentice Hall
33. Hilaire T, Chevrel P, Whidborne JF (2010) Finite wordlength controller realisations using the specialised implicit form. International Journal of Control 83(2):330–346. DOI 10.1080/00207170903160747
34. Peretz M, Ben-Yaakov S (2010) Digital control of resonant converters: Resolution effects on limit cycles. IEEE T POWER ELECTR, 25(6):1652–1661. DOI 10.1109/TPEL.2009.2038159
35. Granas A, Dugundji J (2003) Fixed Point Theory. Monographs in Mathematics, Springer
36. Diniz P, da Silva E, Netto S (2010) Digital Signal Processing: System Analysis and Design. E-Libro, Cambridge University Press
37. Jackson L (1996) Digital Filters and Signal Processing. Springer
38. Middleton RH, Goodwin GC (1990) Digital Control and Estimation: A Unified Approach. Prentice Hall Professional Technical Reference
39. Gevers M, Li G (1993) Parametrizations in control, estimation, and filtering problems: accuracy aspects. Communications and control engineering series, Springer
40. Keel L, Bhattacharyya S (1997) Robust, fragile, or optimal?. IEEE T AUTOMAT CONTR, 42(8):1098–1105. DOI 10.1109/9.618239
41. Jackson L, Kaiser J, McDonald H (1968) An approach to the implementation of digital filters. IEEE T ACOUST SPEECH, 16(3):413–421. DOI 10.1109/TAU.1968.1162002
42. Li G, Anderson B, Gevers M, Perkins J (1992) Optimal fwl design of state-space digital systems with weighted sensitivity minimization and sparseness consideration. IEEE T CIRCUITS-I, 39(5):365–377. DOI 10.1109/81.139287
43. Lopez B, Hilaire T, Didier LS (2014) Formatting bits to better implement signal processing algorithms. In: Proceedings of 4th international Conference on Pervasive and Embedded Computing and Communication Systems, pp. 104–111. DOI 10.5220/0004711201040111
44. Menard D, et al (2012) Design of fixed-point embedded systems (defis) french anr project. In: Proceedings of Conference on Design and Architectures for Signal and Image Processing, pp. 1–2
45. Menard D, Rocher R, Sentieys O (2008) Analytical fixed-point accuracy evaluation in linear time-invariant systems. IEEE T CIRCUITS-I 55(10):3197–3208. DOI 10.1109/TCSI.2008.923279

46. Parashar K, Menard D, Sentieys O (2013) A polynomial time algorithm for solving the word-length optimization problem. In: Proceedings of IEEE/ACM International Conference on Computer-Aided Design, pp. 638–645. DOI 10.1109/ICCAD.2013.6691183
47. Sarbishei O, Radecka K, Zilic Z (2012) Analytical optimization of bit-widths in fixed-point lti systems. IEEE T COMPUT AID D, 31(3):343–355. DOI 10.1109/TCAD.2011.2170988
48. Skaf J, Boyd S (2008) Filter design with low complexity coefficients. IEEE T SIGNAL PROCES, 56(7):3162–3169. DOI 10.1109/TSP.2008.919386
49. Viassolo D (2003) Realizations for fixed-point implementation of controllers and filters with peak-to-peak scaling. In: Proceedings of 2003 American Control Conference, pp. 83–88. DOI 10.1109/ACC.2003.1238918
50. Balakrishnan V, Boyd S (1992) On computing the worst-case peak gain of linear systems. In: Proceedings of 31st IEEE Conference on Decision and Control, pp. 2191–2192. DOI 10.1109/CDC.1992.371407
51. Hilaire T, Chevrel P (2011) Sensitivity-based pole and input-output errors of linear filters as indicators of the implementation deterioration in fixed-point context. EURASIP J ADV SIG PR. DOI 10.1155/2011/893760
52. Li T, Zheng WX (2012) New stability criterion for fixed-point state-space digital filters with generalized overflow arithmetic. IEEE T CIRCUITS-I, 59(7):443–447. DOI 10.1109/TCSII.2012.2198983
53. Thiele L (1986) On the sensitivity of linear state-space systems. IEEE T CIRCUITS SYST 33(5):502–510. DOI 10.1109/TCS.1986.1085951
54. Feng Y, Chevrel P, Hilaire T (2011) Generalised modal realisation as a practical and efficient tool for fwl implementation. International Journal of Control 84(1):66–77
55. Hilaire T, Lopez B (2013) Reliable implementation of linear filters with fixed-point arithmetic. In: IEEE Workshop on Signal Processing Systems, pp. 401–406. DOI 10.1109/SiPS.2013.6674540
56. Golnaraghi F, Kuo B (2010) Automatic Control Systems. John Wiley
57. Peterchev A, Sanders S (2003) Quantization resolution and limit cycling in digitally controlled PWM converters. IEEE T POWER ELECTR, 18(1):301–308. DOI 10.1109/TPEL.2002.807092
58. Vaidyanathan P, Liu V (1987) An improved sufficient condition for absence of limit cycles in digital filters. IEEE T CIRCUITS-I, 34(3):319–322. DOI 10.1109/TCS.1987.1086118
59. Jury E, Lee B (1964) On the absolute stability of nonlinear sample-data systems. IEEE T AUTOMAT CONTR, 9(4):551–554. DOI 10.1109/TAC.1964.1105734
60. Singh V (1986) An extension of jury - lee's criterion for the stability analysis of fixed-point digital filters designed with two's complement arithmetic. IEEE T CIRCUITS-I, 33(3):355–355. DOI 10.1109/TCS.1986.1085907
61. Erickson KT, Michel A (1985) Stability analysis of fixed- point digital filters using computer generated lyapunov functions- part i: Direct form and coupled form filters. IEEE T CIRCUITS-I, 32(2):113–132. DOI 10.1109/TCS.1985.1085676
62. Erickson KT, Michel A (1985) Stability analysis of fixed- point digital filters using computer generated lyapunov functions- part ii: Wave digital filters and lattice digital filters. IEEE T CIRCUITS-I, 32(2):132–142. DOI 10.1109/TCS.1985.1085677
63. Li G, Wan C, He X (2008) Digital filter realizations absent of self-sustained oscillations. In: Proceedings of IEEE International Symposium on Circuits and Systems, pp. 2669–2672. DOI 10.1109/ISCAS.2008.4542006
64. Mills W, Mullis C, Roberts R (1978) Digital filter realizations without overflow oscillations. In: Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing, pp. 71–74. DOI 10.1109/ICASSP.1978.1170524
65. Bauer P, Leclerc LJ (1991) A computer-aided test for the absence of limit cycles in fixed-point digital filters. IEEE T SIGNAL PROCES, 39(11):2400–2410. DOI 10.1109/78.97995
66. Djebbari A, Belbachir M, Rouvaen J (1999) A fast exhaustive search algorithm for checking limit cycles in fixed-point digital filters. Signal Processing 69(2):199–205
67. Premaratne K, Kulasekere E, Bauer P, Leclerc LJ (1996) An exhaustive search algorithm for checking limit cycle behavior of digital filters. IEEE T SIGNAL PROCES, 44(10):2405–2412. DOI 10.1109/78.539026
68. Utrilla-Manso M, Lopez-Ferreras F, Oses-del Campo D, Martin-Martin P (2001) A computer-aided test for the characterization of parasitic oscillations in iir digital filters. In: Proceedings of 2nd International Symposium on Image and Signal Processing and Analysis, pp. 475–478. DOI 10.1109/ISPA.2001.938676

69. Green B, Turner L (1988) New limit cycle bounds for digital filters. IEEE T CIRCUITS-I, 35(4):365–374. DOI 10.1109/31.1751
70. Campo J, Cruz-Roldan F, Utrilla-Manso M (2006) Tighter limit cycle bounds for digital filters. IEEE Signal Processing Letters 13(3):149–152. DOI 10.1109/LSP.2005.862606
71. Lee E (2008) Cyber physical systems: Design challenges. In: Proceedings of 11th IEEE International Symposium on Object Oriented Real-Time Distributed Computing, pp. 363–369. DOI 10.1109/ISORC.2008.25
72. Beyer D (2014) Status report on software verification - (competition summary SV-COMP). In: Proceedings of Tools and Algorithms for the Construction and Analysis of Systems, LNCS 8413, pp. 373–388. DOI 10.1007/978-3-642-54862-8_25
73. Biere A (2009) Bounded model checking. In: Handbook of Satisfiability, IOS Press, pp. 457–481
74. Barrett CW, Sebastiani R, Seshia SA, Tinelli C (2009) Satisfiability modulo theories. In: Handbook of Satisfiability, IOS Press, pp. 825–885
75. Cordeiro L, Fischer B (2011) Verifying multi-threaded software using smt-based context-bounded model checking. In: Proceedings of 33rd International Conference on Software Engineering, pp. 331–340. DOI 10.1145/1985793.1985839
76. Cordeiro L, Fischer B, Marques-Silva J (2012) SMT-Based Bounded Model Checking for Embedded ANSI-C Software. IEEE T SOFTWARE ENG 38(4):957–974. DOI 10.1109/TSE.2011.59
77. Falke S, Merz F, Sinz C (2013) LLBMC: Improved bounded model checking of C programs using LLVM: (competition contribution). In: Proceedings of Tools and Algorithms for the Construction and Analysis of Systems, LNCS 7795, pp. 623–626. DOI 10.1007/978-3-642-36742-7_48
78. Kroening D, Tautschnig M (2014) CBMC - C bounded model checker (competition contribution). In: Proceedings of Tools and Algorithms for the Construction and Analysis of Systems, LNCS 8413, pp. 389–391. DOI 10.1007/978-3-642-54862-8_26
79. Abreu RB, Cordeiro LC, Filho EBL (2013) Verifying Fixed-Point Digital Filters using SMT-Based Bounded Model Checking. In: XXXI Brazilian Symposium on Telecommunications, DOI 10.14209/sbrt.2013.57
80. Cordeiro L, Morse J, Nicole D, Fischer B (2012) Context-Bounded Model Checking with ESBMC 1.17 - (Competition Contribution). In: Proceedings of Tools and Algorithms for the Construction and Analysis of Systems, LNCS 7214, pp. 534–537. DOI 10.1007/978-3-642-28756-5_42
81. Morse J, Cordeiro L, Nicole D, Fischer B (2013) Handling Unbounded Loops with ESBMC 1.20 - (Competition Contribution). In: Proceedings of Tools and Algorithms for the Construction and Analysis of Systems, LNCS 7795, pp. 619–622. DOI 10.1007/978-3-642-36742-7_47
82. Morse J, Ramalho M, Cordeiro L, Nicole D, Fischer B (2014) ESBMC 1.22 - (Competition Contribution). In: Proceedings of Tools and Algorithms for the Construction and Analysis of Systems, LNCS 8413, pp. 405–407. DOI 10.1007/978-3-642-54862-8_31
83. Ogata K (1995) Discrete-Time Control Systems. Prentice Hall International editions, Prentice-Hall International
84. MSP430G2231 Mixed Signal Controller, Texas Instrument™, `http://www.ti.com/lit/ds/symlink/msp430g2231-ep.pdf`, Jul. 2015.
85. Barreto R, Cordeiro L, Fischer B (2011) Verifying Embedded C Software with Timing Constraints Using an Untimed Bounded Model Checker. In Brazilian Symposium on Computing Systems Engineering, pp. 46–52. DOI http://dx.doi.org/10.1109/SBESC.2011.19
86. Patterson D, Hennessy J (2012) Computer Organization and Design - The Hardware / Software Interface (Revised 4th Edition). The Morgan Kaufmann Series in Computer Architecture and Design, Academic Press 2012, ISBN 978-0-12-374750-1.
87. Ismail H, Bessa I, Cordeiro L, Lima Filho E, Chaves Filho J (2015) DSVerifier: A Bounded Model Checking Tool for Digital Systems. In: Proceedings of 22nd International SPIN Symposium on Model Checking of Software, LNCS 9232, pp. 126–131. DOI 10.1007/978-3-319-23404-5_9
88. S. Kim, H. D. Patel, and S. A. Edwards, "Using a Model Checker to Determine Worst-case Execution Time", *tech. rep., Computer Science Department Columbia University*, 2009.
89. Code Composer Studio™Integrated Development Environment for MSP430, Texas Instrument™, `http://www.ti.com/tool/ccstudio-msp430`, Jul. 2015.
90. Guennebaud G (2011) Eigen: a C++ Linear Algebra Lirary, URL `http://eigen.tuxfamily.org`, Accessed 22 December 2014

91. Fadali S, Visioli A (2009) Digital Control Engineering: Analysis and Design. Electronics & Electrical, Elsevier/Academic Press
92. Feuer A, Goodwin G (1996) Sampling in Digital Signal Processing and Control . Birkhauser
93. Anta A, Majumdar R, Saha I, Tabuada P (2010) Automatic Verification of Control System Implementations. In: Proceedings of tenth ACM international Conference on Embedded software, pp. 9–18. DOI 10.1145/1879021.1879024
94. Brleanu A, Bitoiu V, Stan A (2011) Digital filter optimization for C language. Advances in Electrical and Computer Engineering 11(3):111–114
95. Caffarena G, Carreras C, Lopez J, Fernandez A (2010) Fast fixed-point optimization of DSP algorithms. In: Proceedings of 18th IEEE/IFIP VLSI System on Chip Conference, pp. 195–200. DOI 10.1109/VLSISOC.2010.5642659
96. Hilaire T, Menard D, Sentieys O (2008) Bit accurate roundoff noise analysis of fixed-point linear controllers. In: Proceedings of IEEE International Conference on Computer-Aided Control Systems, pp. 607–612. DOI 10.1109/CACSD.2008.4627366
97. Lopez J, Caffarena G, Carreras C, Nieto-Taladriz O (2008) Fast and accurate computation of the roundoff noise of linear time-invariant systems. IET Circuits, Devices Systems 2(4):393–408. DOI 10.1049/iet-cds:20070198
98. Naud JC, Meunier Q, Ménard D, Sentieys O (2011) Fixed-point Accuracy Evaluation in the Context of Conditional Structures. In: Proceedings of 19th European Signal Processing Conference. URL `https://hal.inria.fr/hal-00747610`
99. Patra A, Mukhopadhyay S (1996) A software tool for performance evaluation of digital control algorithms on finite wordlength processors. Computers & Electrical Engineering 22(6):403 – 419. DOI 10.1016/S0045-7906(96)00008-0
100. Alur R, Courcoubetis C, Dill D (1990), "Model-checking for real-time systems," In: Proceedings of Fifth Annual IEEE Symposium on Logic in Computer Science, pp. 414–425. DOI 10.1109/LICS.1990.113766
101. Alur R, Courcoubetis C, Dill D (1993), "Model-Checking in Dense Real-Time," In: Information and Computation 104 (1), pp. 2–34. DOI 10.1006/inco.1993.1024
102. Dutertre B, Rushby J, Tiwari A, Munoz C, Siminiceanu R (2008) Formal Verification and Automated Testing for Diagnostic and Monitoring Systems. In: Proceedings of AIAA Guidance, Navigation and Control Conference and Exhibit. DOI 10.2514/6.2008-6802
103. Simko G, Jackson EK (2014) A Bounded Model Checking Tool for Periodic Sample-hold Systems. In: Proceedings of 17th international Conference on Hybrid systems: computation and control, pp. 157–162. DOI 10.1145/2562059.2562134
104. Prabhu S, Dasgupta P (2013) Model Checking Controllers with Predicate Inputs. In: Proceedings of International Conference on VLSI Design, pp. 332–337. DOI 10.1109/VLSID.2013.210