Verifying Digital Systems with MATLAB

Lennon Chaves, Iury Bessa
Federal University of Amazonas
Manaus, Brazil
lennonchaves@ufam.edu.br,
iurybessa@ufam.edu.br

Lucas Cordeiro,
Daniel Kroening
University of Oxford
Oxford, United Kingdom
lucas.cordeiro@cs.ox.ac.uk,
kroening@cs.ox.ac.uk

Eddie Lima
Samsung Electronics
Manaus, Brazil
eddie_batista@yahoo.com.br

ABSTRACT

A MATLAB toolbox is presented, with the goal of checking occurrences of design errors typically found in fixed-point digital systems, considering finite word-length effects. In particular, the present toolbox works as a front-end to a recently introduced verification tool, known as Digital-System Verifier (DSVerifier), and checks overflow, limit cycle, quantization, stability, and minimum phase errors in digital systems represented by transfer-function and state-space equations. It provides a command-line version with simplified access to specific functionality and a graphical-user interface, which was developed as a MATLAB application. The resulting toolbox enables application of verification to real-world systems by control engineers.

CCS CONCEPTS

• **Computer systems organization → Real-time systems**; *Embedded systems*; • **Software and its engineering → Model checking**; Formal methods; • **Theory of computation** → *Verification by model checking*;

KEYWORDS

Embedded Digital Systems; MATLAB Toolbox; Software Model Checking; Formal Verification.

ACM Reference format:
Lennon Chaves, Iury Bessa, Lucas Cordeiro, Daniel Kroening, and Eddie Lima. 2017. Verifying Digital Systems with MATLAB. In *Proceedings of 26th ACM SIGSOFT International Symposium on Software Testing and Analysis , Santa Barbara, CA, USA, July 2017 (ISSTA'17-DEMOS),* 4 pages.
https://doi.org/10.1145/3092703.3098228

1 INTRODUCTION

Digital systems (*e.g.*, filters and controllers) are used in a wide variety of applications, owing to advantages over their analog counterparts, such as reliability, flexibility and cost. Nonetheless, there are disadvantages: since they are normally implemented in microprocessors, errors might be introduced, due to quantization and related round-off effects [6].

Hardware choices, structure representations (*e.g.*, direct forms) and implementation features (*e.g.*, fixed-point arithmetic) can influence the precision and performance of a given digital system [2]. Additionally, such implementations are particularly susceptible to finite word-length (FWL) effects, *e.g.*, overflows, limit cycles and poles/zeros sensitivity, which have the potential to reduce reliability and efficiency. Previous studies have shown that FWL effects might affect the efficiency, performance and stability of control systems [8, 11, 12]. Thus, it is important to detect such errors and ideally prove safety of the implementations w.r.t. FWL effects.

The Digital-System Verifier (DSVerifier) [7] is a model checker based on Boolean Satisfiability (SAT) and Satisfiability Modulo Theories (SMT) and detects such errors in digital systems. DSVerifier checks specific properties related to overflow, limit cycle, stability and minimum-phase in digital-system implementations and also supports the verification of robust stability, considering parametric uncertainties for closed-loop systems represented by transfer functions [3]. Recently, DSVerifier was extended to support state-space systems to verify violations in stability, controllability, observability and quantization-error properties [10]. Although those contributions are important advances in formal verification of digital systems, they do not integrate with tools such as MATLAB [13], which are usually employed in the design of digital filters and controllers.

Currently, there are several toolboxes in MATLAB that facilitate digital system design [13]. For instance, the fixed-point designer toolbox provides data-types and tools for developing fixed-point digital systems. There are also other modules with different objectives, such as optimization, control systems and digital signal processing. In particular, users could employ formal verification methods to identify errors and generate test vectors for reproducing failures. In that sense, Simulink Design Verifier [13] employs formal methods to identify hidden design errors, without extensive simulation runs; it detects blocks that result in integer overflow, dead logic, array access, division by zero and requirement violations. Additionally, it is possible to use tools for detecting errors in C/C++ code, through Polyspace Bug Finder [13]. Nonetheless, both tools are unable to automatically detect specific errors related to digital system design (*e.g.*, limit cycle, stability and minimum-phase), unless an engineer provides additional assertions. Finally, the mentioned tools do not consider FWL effects during verification and also, there is no MATLAB toolbox for verifying digital systems using symbolic model checking based on SAT and SMT solvers.

The present paper addresses this problem and describes a MATLAB toolbox for DSVerifier,[1] known as DSVerifier Toolbox, which applies SAT- and SMT-based model checking to digital systems [7], in MATLAB's environment. The main advantage regarding the use of a MATLAB toolbox lies in designing digital systems in MATLAB

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

ISSTA'17-DEMOS, July 2017, Santa Barbara, CA, USA

ACM ISBN 978-1-4503-5076-1/17/07…$15.00
https://doi.org/10.1145/3092703.3098228

and then promptly verifying their desired properties. Additionally, when using DSVerifier Toolbox, an engineer is able to design a digital system with MATLAB, through transfer-function or state-space representations, consider low-level systems parameters (numerical format), define realization forms (delta and direct forms) and evaluate different overflow modes (wrap-around or saturate mode). Finally, if a verification procedure fails, DSVerifier Toolbox returns a counterexample in a ".MAT" file, which exploits a given violation, considering inputs, initial states and outputs, in order to reproduce a particular counterexample.

2 VERIFYING DIGITAL SYSTEMS WITH DSVERIFIER TOOLBOX

2.1 DSVerifier Toolbox Architecture

The proposed verification methodology is based on DSVerifier [7] and can be split into four main steps, as illustrated in Fig. 1.

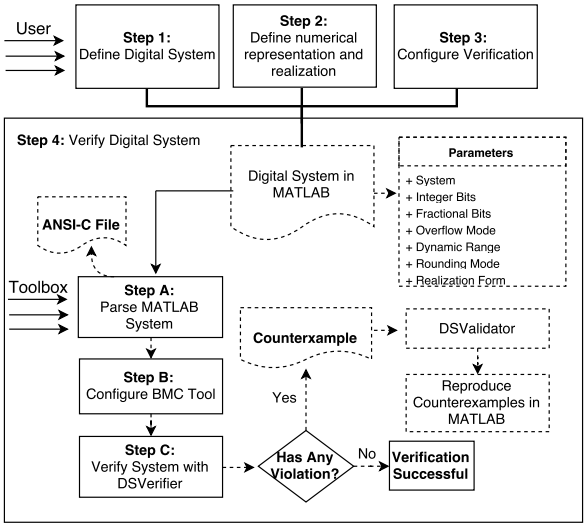


Figure 1: DSVerifier Toolbox's Verification Methodology.

In step 1, a digital system is designed (as open- or closed-loop), with any design technique or tool. Implementation features are defined in step 2, *i.e.*, the FWL format that includes the number of bits in the integer and fractional parts, dynamic range and realization form (direct or delta). Here, DSVerifier formulates a FWL function $\mathcal{FWL}[\cdot] : \mathcal{P}^n \rightarrow \mathcal{P}^n$, where $\mathcal{P}^n$ is a space of polynomials of n-th order, to reproduce the effects of the chosen FWL format over the coefficients of a digital system. For instance, $\mathcal{FWL}[B(z)]$ and $\mathcal{FWL}[A(z)]$ represent the denominator and numerator polynomials $B(z)$ and $A(z)$ of a transfer function with FWL effects used to compute round-off effects in digital systems. Those definitions are then passed to DSVerifier, along with hardware specifications, verification parameters and properties to be checked.

In particular, with respect to open-loop systems in transfer-function representation, DSVerifier supports verification of overflow (we check whether a sum or product exceeds the number representation), stability and minimum-phase (we check whether the system poles and zeros are inside the unitary circle), limit cycle (we check whether there are persistent oscillations in the output of a system with constant input or zero input) and quantization error (we check whether the output error in digital systems implementation will be within admissible range), while it provides verification for stability, controllability (we check whether the system states can be changed by changing the system input), observability (we check whether the value of the initial state can be determined from the system output) and quantization properties, in state-space representation. Regarding closed-loop systems represented by a controller and a plant, in transfer function form, DSVerifier is able to verify stability, quantization error and limit-cycle, while it checks stability, controllability, observability and quantization error when state-space equations are employed.

Once the configuration has been set up in step 3, the verification process is then started in step 4, with the chosen back-end model checker: CBMC [9] or ESBMC [5] can be used to verify the generated C code. DSVerifier then checks the desired properties and returns "successful" if there is no property violation in the proposed implementation, or "failed" together with a counterexample, which contains inputs and states that lead the system under evaluation to a given property violation. In the latter case, the provided counterexample supports the fixes to the implementation parameters and the design. Realization, representation and FWL format can be re-chosen to avoid further errors. This process is repeated until a digital controller implementation does not present any failure.

DSVerifier Toolbox uses bounded model checking (BMC) as verification engine [5, 9]. The basic idea of BMC is to check the negation of a given property ϕ at a given depth k. Thus, overflow, limit cycle and quantization errors, in transfer-function representation, and quantization error verification, when employing a state-space representation, must be unrolled k times to find violations; the verification result is sound for executions of up to k steps. By contrast, properties such as stability and minimum-phase, in transfer-function representation, and controllability, stability and observability, in state-space representation, do not depend on the system inputs and thus do not require a bound k (verification is complete and sound) [2, 3, 10].

Note that the verification methodology is split into two main stages: manual (user) and automated (toolbox) procedures. In the former, the user manually performs steps 1 to 3, which are the same tasks performed by DSVerifier. Note further that all those specifications are provided as parameters and translated into a specific format in the automated procedures performed by the toolbox, as illustrated in Fig. 1. In particular, the DSVerifier Toolbox's automated engine (steps A to C) receives a digital system specification (as parameters) and verifies the desired property ϕ. In step A, an intermediate ANSI-C code for the desired implementation is generated, based on parameters that are then translated into a specific format (in MATLAB) and parsed, while the respective BMC tool is set and all requirements are configured in step B. Finally, in step C, the resulting ANSI-C code is passed to the verifier. If any violation is found, then DSVerifier reports a counterexample, which contains system inputs that lead to a failure; otherwise, it returns a successful verification. In particular, in case of a failure, the proposed DSVerifier Toolbox receives a counterexample and generates a corresponding ".MAT' file.

The main development challenge related to DSVerifier Toolbox is the represention of the control system models. In particular, our implementation consists of 2000 lines of code related to

MATLAB scripts to soundly represent the control system models, considering FWL effects. In addition, there are 250 lines of code to parse MATLAB functions and there are approximately 1000 benchmarks to ensure the DSVerifier Toolbox's regression. All those artifacts together with documentation and videos are available in a public github repository, which is located at https://github.com/ssvlab/dsverifier/tree/master/toolbox-dsverifier.

2.2 DSVerifier Toolbox Features

DSVerifier Toolbox's features can be described as follows:

(1) **Digital-system representation:** DSVerifier Toolbox handles digital systems represented by transfer-function and state-space representations (cf. step 1 of Fig. 1).
(2) **Realization:** DSVerifier Toolbox performs the verification of direct-form I (DFI), direct-form II (DFII) and transposed direct-form II (TDFII), and also delta direct-form I (DDFI), delta direct-form II (DDFII) and delta transposed direct-form II (TDDFII) (cf. step 2 of Fig. 1).
(3) **Open-loop systems:** DSVerifier Toolbox verifies, for transfer-function representation, stability, overflow, minimum phase, limit-cycle and quantization error, while in state-space representation, it verifies stability, quantization error, observability and controllability properties (cf. step 3 of Fig. 1).
(4) **Closed-loop systems:** DSVerifier Toolbox verifies stability, limit-cycle and quantization error in transfer-function representation, while for state-space systems, all properties mentioned for open-loop systems are checked, via state feedback matrix (cf. step 3 of Fig. 1).
(5) **BMC tools:** DSVerifier Toolbox handles the verification for digital-systems using CBMC [9] or ESBMC [5] as back-end, to perform the symbolic verification (cf. step 3 of Fig. 1).

2.3 DSVerifier Toolbox Usage

In order to explain the DSVerifier Toolbox's workflow, the following second-order controller represented by a transfer-function $H(z) = \frac{B(z)}{A(Z)}$ (cf. section 2) for an A/C motor plant is used

$$H(z) = \frac{z^3 - 2.819z^2 + 2.6370z - 0.8187}{z^3 - 1.97z^2 + 1.033z - 0.06068}. \quad (1)$$

2.3.1 Command Line Version. Users must provide a digital system described as a MATLAB system using a `tf` (for transfer-function) or an `ss` (for state-space) command (cf. step 1 of Fig. 1). DSVerifier Toolbox is then invoked to check the digital system representation and the desired property (cf. steps 2 and 3 of Fig. 1). Table 1 summarizes the DSVerifier Toolbox's commands that perform the proposed automatic verification and the required parameters for each property (cf. step 4 of Fig. 1). In Table 1, *system* represents the digital system in transfer-function or state-space format, *intbits* is the integer part, *fracbits* is the fractional part, *max* and *min* are the maximum and minimum dynamic range, respectively, *bound* is the k bound to be employed during verification, *cmode* is the connection mode, for closed-loop systems in transfer-function (series or feedback), and *error* is the maximum possible value in the quantization error check. Additionally, optional parameters can be included, such as overflow mode, rounding mode, BMC tool, solver, quantization error mode and delta coefficient (for delta realization).[2] All available functions w.r.t. DSVerifier Toolbox have been exhaustively tested and experimental results are available online.[3]

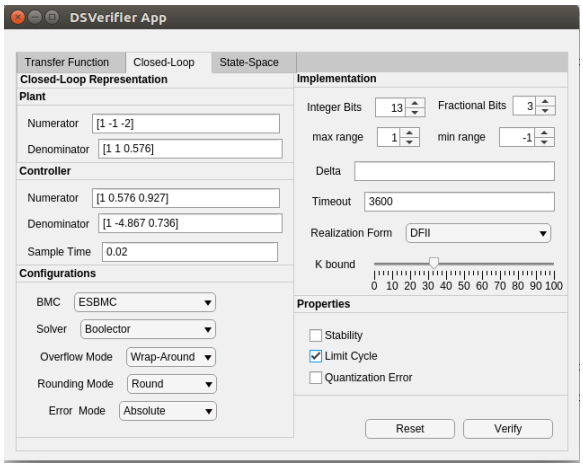


Figure 2: GUI Application for Transfer-Function Verification, in Closed-Loop Format.

2.3.2 GUI Application Version. A graphical user interface application was developed (Fig. 2), in order to favor digital-system verification in MATLAB, besides improving usability and, consequently, attracting more digital-system engineers. Users can provide all required parameters for digital-system verification: specification, target implementation and properties to be checked.

2.4 Illustrative Example

To illustrate the DSVerifier Toolbox's usage, Fig. 3 shows the stability verification for the digital system specified in Eq. 1, where "num" and "den" represent the numerator $A(z)$ and denominator $B(z)$, resp., using a dynamic range $[-1, 1]$ and a fixed-point format $\langle 2, 13\rangle$, *i.e.*, 2 bits for the integer part and 13 for the fractional one.

```
>> num = [1.0000 -2.8190 2.6370 -0.8187];
>> den = [1.0000 -1.9700 1.0330 -0.0607];
>> system = tf(num,den,0.001);
>> verifyStability(system,2,13,1,-1);
>> VERIFICATION SUCCESSFUL
```

Figure 3: Verifying stability for Eq. 1 in MATLAB, with a fixed-point format $\langle 2, 13\rangle$.

If the fixed-point format is changed to $\langle 12, 3\rangle$, for the same system described in Eq. 1, the verification indicates a failure, *i.e.*, a digital system is unstable, as shown in Fig. 4, which indicates that DSVerifier Toolbox can correctly verify digital systems with different implementations.

After verifying that the adopted digital system is unstable with format $\langle 12, 3\rangle$, the respective failed verification result can be confirmed by reproducing the counterexample generated by DSVerifier

[2] All functions implemented in DSVerifier Toolbox are detailed in the Toolbox's Documentation.
[3] http://www.dsverifier.org/benchmarks

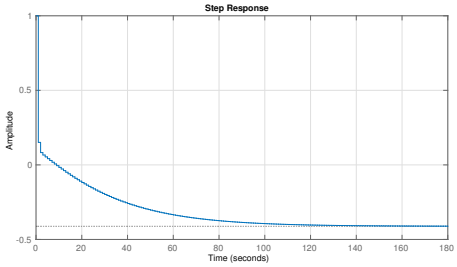
Table 1: DSVerifier Toolbox's commands and parameters used during verification procedures.

Verification Command	system	intbits	fracbits	max	min	bound	cmode	error
verifyStability	x	x	x	x	x			
verifyOverflow	x	x	x	x	x	x		
verifyError	x	x	x	x	x	x		x
verifyMinimumPhase	x	x	x	x	x			
verifyLimitCycle	x	x	x	x	x	x		
verifyClosedStability	x	x	x	x	x		x	
verifyClosedQuantizationError	x	x	x	x	x	x	x	x
verifyClosedLimitCycle	x	x	x	x	x	x	x	
verifyStateSpaceStabiltiy	x	x	x					
verifyStateSpaceControllability	x	x	x					
verifyStateSpaceObservability	x	x	x					
verifyStateSpaceQuantizationError	x	x	x			x		x

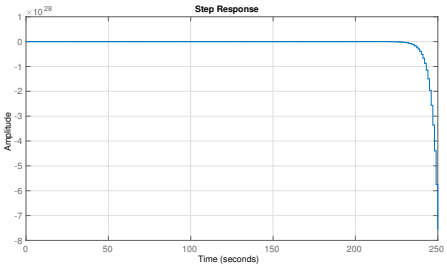
```
>> verifyStability(system,12,3,1,-1);
>> VERIFICATION FAILED
```

Figure 4: Verifying stability for Eq. 1 in MATLAB, with a fixed-point format $\langle 12, 3\rangle$.

Toolbox. In particular, we compute the roots of $\mathcal{FWL}[A(z)]$, in order to check stability. If any root has modulus equal or greater than one, then the system is unstable; otherwise, it is stable. When applying $\mathcal{FWL}[H(z)]$, with the first case (*i.e.*, $\langle 2, 13\rangle$) and computing the denominator roots of $\mathcal{FWL}[H(z)]$, we obtain the following set of poles: $I = \{0.9629, 0.9400, 0.0672\}$, from which we can conclude that all poles are inside the unit circle. This means that the mentioned system is stable, if the numeric representation $\langle 2, 13\rangle$ is used; however, when applying $\mathcal{FWL}[H(z)]$ to the second case (*i.e.*, $\langle 12, 3\rangle$) and then computing the denominator roots of $\mathcal{FWL}[H(z)]$, the following set of poles is obtained: $J = \{1.3090, 0.5000, 0.1910\}$, where set J has one root with modulus greater than one, which confirms that using $\langle 12, 3\rangle$ format the verified system becomes unstable. The stability for the digital systems described above could be indeed observed through the associated step response for both cases, as shown in Fig. 5. In subfigure 5(a), the step response shows that the digital system is stable, while in 5(b) it is unstable.



(a) Successful verification using the fixed-point format $\langle 2, 13\rangle$.



(b) Failed verification using the fixed-point format $\langle 12, 3\rangle$.

Figure 5: Step Response for Eq. (1).

3 CONCLUSION

DSVerifier Toolbox is able to verify dynamic digital-systems (controllers or filters) designed in MATLAB given as transfer-function or state-space representations in open- or closed-loop form. We have shown that using different numerical representations for a digital controller can yield different verification results. In particular, we demonstrated that the choice of representation has the potential to cause instability and then compromises the entire system. Given the current literature in formal verification, there is no other MATLAB toolbox for verifying specific properties of digital systems while taking into account implementation aspects. As future work, DSVerifier Toolbox will perform verification for robust stability and be combined with DSValidator [4].

REFERENCES

[1] K.J. Åström and B. Wittenmark. 1997. *Computer-controlled systems: theory and design.* Prentice Hall, Upper Saddle River, New Jersey.
[2] I.V. Bessa, H.I. Ismail, L.C. Cordeiro, and J.E.C. Filho. 2016. Verification of fixed-point digital controllers using direct and delta forms realizations. *Design Autom. for Emb. Sys.* 20, 2 (2016), 95–126.
[3] Iury Bessa, Hussama Ismail, Reinaldo Palhares, Lucas Cordeiro, and João Chaves Filho. 2017. Formal Non-Fragile Stability Verification of Digital Control Systems with Uncertainty. *IEEE Trans. on Computers* 66, 3 (2017), 545–552.
[4] L. Chaves, I. Bessa, and L. Cordeiro. 2016. *DSValidator: An Automated Counterexample Reproducibility Tool for Digital Systems.* Technical Report. arXiv.
[5] L. Cordeiro, B. Fischer, and J. Silva. 2012. SMT-Based Bounded Model Checking for Embedded ANSI-C Software. *IEEE Trans. Soft. Eng.* 38, 4 (2012), 957–974.
[6] Paulo Diniz, Sergio Netto, and Eduardo Da Silva. 2002. *Digital Signal Processing: System Analysis and Design.* Cambridge University Press.
[7] Hussama Ismail, Iury Bessa Lucas Cordeiro, Eddie B. Lima Filho, and Jo ao Edgar Chaves Filho. 2015. DSVerifier: A Bounded Model Checking Tool for Digital Systems. In *SPIN (LNCS)*, Vol. 9232. Springer, 126–131.
[8] L.H. Keel and S.P. Bhattacharyya. 1997. Robust, fragile, or optimal? *IEEE Trans. on Automatic Control* 42, 8 (1997), 1098–1105.
[9] D. Kroening and M. Tautschnig. 2014. CBMC – C Bounded Model Checker (Competition contribution), In TACAS. *LNCS* 8413, Springer, 389–391.
[10] Felipe R. Monteiro. 2016. Bounded model checking of state-space digital systems: the impact of finite word-length effects on the implementation of fixed-point digital controllers based on state-space modeling. In *FSE*. ACM, 1151–1153.
[11] M.M. Peretz and S. Ben-Yaakov. 2010. Digital Control of Resonant Converters: Resolution Effects on Limit Cycles. *IEEE Trans. on Power Elect.* 25, 6, 1652–1661.
[12] A.V. Peterchev and S.R. Sanders. 2003. Quantization resolution and limit cycling in digitally controlled PWM converters. *IEEE Trans. on Power Elect.* 18, 1, 301–308.
[13] The MathWorks, Inc. 2017. MATLAB Toolbox. https://www.mathworks.com/products/. (2017). Accessed: 02-05-2017.