DSVerifier: A Bounded Model Checking Tool for Digital Systems

Hussama I. Ismail, Iury V. Bessa, Lucas C. Cordeiro,
Eddie B. de Lima Filho and João E. Chaves Filho

Electronic and Information Research Center
Federal University of Amazonas, Brazil

Abstract. This work presents the Digital-Systems Verifier (DSVerifier), which is a verification tool developed for digital systems. In particular, DSVerifier employs the bounded model checking technique based on satisfiability modulo theories (SMT) solvers, which allows engineers to verify the occurrence of design errors, due to the finite word-length approach employed in fixed-point digital filters and controllers. This tool consists in an additional module for the efficient SMT-based context-bounded model checker and presents command-line and graphical user interface (GUI) versions. Indeed, the GUI version is essential for reporting property violations, together with associated counterexamples. DSVerifier is implemented in C/C++ and uses JavaFX for providing GUI support.

1 Introduction

Digital filters and controllers are currently used in a wide variety of applications, due to some advantages over their analog counterparts, such as improved reliability, sensitivity, flexibility, and cost. However, errors may be introduced during the quantization process, given that such systems are typically implemented in microcomputers, microprocessors, digital signal processors, and field-programmable gate arrays. Thus, hardware choice, computational representation (e.g., direct and delta forms), and other implementation features (e.g., number of bits, fixed- or floating-point arithmetic, and sample rate) have a strong influence on precision and performance figures.

Implementations of digital systems are especially susceptible to finite word-length (FWL) effects (e.g., overflows, limit cycles, and poles and zeros sensitivity), which thus reduce their reliability and efficiency. For instance, the presence of limit cycles, in digital systems, reduces semiconductor lifespans and increases energy consumption. Besides, pole-zero positions also affect the system dynamics and fundamental requisites, such as stability.

In order to avoid performance degradation, engineers usually invest a great deal of time and effort during the design phase, aiming to solve problems caused by FWL effects. Although one finds a myriad of design tools, there is a clear lack of initiatives for validating digital systems, w.r.t. implementation aspects. In particular, software engineering techniques typically disregard the platform in which the (embedded) system software operates and restrict itself to verify software in isolation [1].

Alur et al. [2,3] introduced the earliest application of model checking for digital systems, represented by timed automata. Those influential studies inspired the development of various model checking tools for hybrid automatas and cyber-physical systems, e.g., UPPAAL [4], Open-Kronos [5], and Maellan [6]. However, such approaches are usually employed for high-level verification and have not been used for verifying resilience, i.e., system robustness related to implementation aspects. One may notice there is still a gap, regarding verification tools and methodologies to check for implementation aspects of embedded systems.

The present paper addresses this problem with the Digital-Systems Verifier (DSVerifier)[1], which is a bounded model checking (BMC) tool based on satisfiability modulo theories (SMT). DSVerifier is a powerful tool for supporting the design and verification steps of digital systems, which is more reliable and less laborious than traditional simulation tools (e.g., Matlab [7]), since it offers formal guarantees and is completely automatic.

In previous studies [8,9,10], an SMT-based BMC approach related to overflow, limit cycle, time constraints, stability, and minimum phase, in digital filters and controllers, was already discussed, and a novel methodology for verifying digital systems was presented. In contrast, this paper tackles implementation and usage aspects, related to a tool that provides support for the same methodology.

2 The Digital-Systems Verifier (DSVerifier)

DSVerifier is an internal module for the efficient SMT-based context-bounded model checker (ESBMC) [11], with the goal to add support for digital-system verification. The complete verification tool includes four components from ESBMC, together with DSVerifier, which are represented as dashed white boxes in Fig. 1: C parser, GOTO Program, GOTO Symex, and SMT solver.

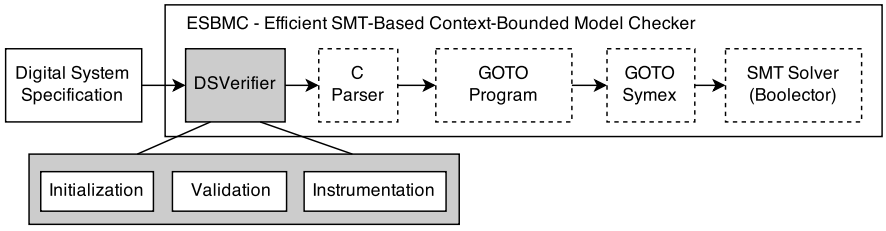

Fig. 1. An overview of the verification architecture.

The DSVerifier module is included before the C parser (gray box), as seen in Fig. 1. This module provides functions, which are related to quantization in fixed-point arithmetic and different digital-system realizations, and makes use of ESBMC as a verification engine, in order to check for properties related to overflow, limit cycle, time constraints, stability, and minimum phase.

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

In summary, DSVerifier performs three main procedures: initialization, validation, and instrumentation. When DSVerifier receives the digital-system specification, the first step is to initialize its internal parameters for quantization, that is, it computes the maximum and minimum representable numbers for the chosen FWL format. Then, during validation, DSVerifier checks whether all required parameters, for the verification procedure, were correctly provided. In the last step, DSVerifier adds explicit calls to the verification engine (for the evaluated properties), using functions available in ESBMC (e.g., *__ESBMC_assume* and *__ESBMC_assert*), in order to check for property violations.

Once the mentioned procedures are finished, an ANSI-C code file is generated, which can be verified by any C model-checker that supports bit-vector reasoning. This file is directly sent to the C parser module (see Fig. 1) and follows the normal ESBMC verification flow. In the present work, ESBMC is used, since it is the most efficient tool for reasoning about bit-vector programs, according to the last edition of the software verification competition [12]. If the verification framework finds a property violation, it produces a counterexample; otherwise, the evaluated design can be embedded into a computer-based system.

2.1 DSVerifier Features

The current version of DSVerifier supports five verification properties, regarding three direct- and delta-form implementations of digital systems, which include the cascade form. The following verifications are supported:

- **Overflow.** If a sum or product exceeds the number representation, then the resulting value will not be correctly stored. DSVerifier ensures the absence of overflows, by formally verifying every sum and product;
- **Limit Cycle.** There can be persistent oscillations in the output of a system with constant input. DSVerifier is able to check for zero-input limit cycles, for any initial condition;
- **Stability.** DSVerifier may be used for verifying digital-system stability, considering FWL effects on pole locations, i.e., on the system dynamics;
- **Minimum phase.** DSVerifier may perform a similar analysis for system zeros, in order to verify minimum phase for digital controllers;
- **Time constraints.** DSVerifier is able to investigate whether a specific computational realization respects time constraints.

2.2 DSVerifier-Aided Design Methodology

Using DSVerifier, a development engineer may verify if a digital-controller (or filter) design will present the desired performance, when it is embedded into a given hardware, considering the chosen implementation characteristics. An overview of the proposed methodology can be seen in Fig. 2. In step 1, a digital system is initially designed, with any available design technique or tool. Later, the necessary implementation characteristics have to be defined, as shown in steps 2 and 3: FWL format (number of bits in the integer and fractional parts), dynamic range, and realization form (direct or delta). The mentioned definitions are then fed to the DSVerifier engine, along with hardware specifications and other verification parameters, such as verification time (i.e., maximum time that

the verification process takes) and properties to be checked. Once the configuration has been set up, in step 4, the verification process is then started, in step 5, with the chosen model checking tool (ESBMC is used as back-end).

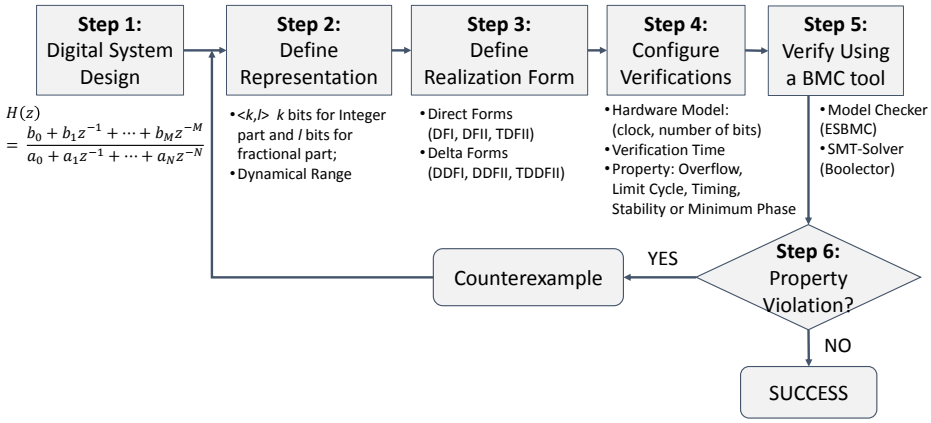

Fig. 2. Proposed methodology for digital-system verification.

DSVerifier then checks the desired properties and, in step 6, returns the verification result, which is 'successful' if there is no property violation in the proposed implementation; otherwise, it returns that the verification 'failed' and shows a counterexample, which contains inputs and states that led the system to the found property violation. With this counterexample, other implementation options (i.e., realization and representation) can be chosen, in order to avoid that failure. This process is repeated until the digital controller implementation does not present any failures, as shown in Fig. 2.

Note that this methodology has been applied to open-loop systems, where the design under verification is unwound k times, together with a property, in order to form an SMT formula, which is then passed to the SMT solver. The verification of stability and minimum-phase is complete and sound, since it does not depend on system outputs and inputs. However, the verification of other property types is typically unsound, unless some induction technique is used.

2.3 DSVerifier Usage

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:

$$H(z) = \frac{2.813z^2 - 0.0163z^1 - 1.872}{z^2 + 1.068z^1 + 0.1239}. \quad (1)$$

It was designed for an induction motor plant (extracted from an example available in Ogata [13]), with a sampling period of $0.5s$.

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

Command-line Version In this version, users must provide a description ANSI-C file, as shown in Fig. 3 for the digital controller represented by Eq. (1). This file contains the digital-system specification (ds), with numerator (ds.b = {2.813, −0.0163, −1.872}) and denominator (ds.a = {1.0, 1.068, 0.1239}), and the implementation specification itself (impl), which contains the number of bits in the integer (impl.int_bits = 4) and precision (impl.frac_bits = 10) parts and the input range (impl.min = -5 and impl.max = 5).

```
#include<dsverifier.h>
digital_system ds = {
    .a = { 1.0, 1.068, 0.1239 },     /* denominator */
    .a_size = 3,                    /* denominator length */
    .b = { 2.813, -0.0163, -1.872 }, /* numerator */
    .b_size = 3                     /* numerator length */
};
implementation impl = {
    .int_bits = 4,          /*  integer bits    */
    .frac_bits = 10,        /*  precision bits  */
    .min = -5.0,            /*  minimum input   */
    .max = 5.0              /*  maximum input   */
};
```

Fig. 3. A digital-system verification input file for DSVerifier.

In the command-line version, DSVerifier is invoked as:

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

where $< file >$ is the digital-system specification file, $< i >$ is the chosen realization, $< j >$ is the property to be verified, and $< k >$ is the verification bound, i.e., the number of times the digital system will be unwound. Currently, 12 realizations are supported: direct form I (DFI), direct form II (DFII), transposed direct form II (TDFII), delta direct form I (DDFI), delta direct form II (DDFII), transposed delta direct form II (TDDFII), cascade direct form I (CDFI), cascade direct form II (CDFII), cascade transposed direct form II (CTDFII), cascade delta direct form I (CDDFI), cascade delta direct form II (CDDFII), and cascade transposed direct form II (CTDDFII). Furthermore, 5 different properties can be chosen: overflow, limit cycle, stability, minimum phase, and timing. Most verifications consider only FWL effects, based on the number of bits specified by the user; however, time-constraint verifications also consider hardware parameters such as processor clock, instruction count, and cycles per instruction.

Graphical User Interface (GUI) In order to facilitate the digital-system verification, a GUI was developed for DSVerifier, aiming to improve usability and, consequently, attract more digital-system engineers. The user can provide all required parameters for digital-system verifications: digital-system specification, information about the target processor, and the desired properties to be checked.

Another interesting feature is the parallel execution of verification tasks, which has the potential to decrease the total verification time spent by DSVerifier. The GUI also provides graphical verification of results and counterexamples with error trace, which help adjust the verified design. It is worth noticing that users can even access documentation, benchmarks, and publications about the tool, which are also available on the DSVerifier website. In terms of software package installation, it is necessary to have at least the Java RunTime Environment Version 8.0 Update 40 (jre1.8.0_40)[3], due to the JavaFX components.

3 Conclusion

DSVerifier was presented as an SMT-based BMC tool for verifying and validating digital systems, which supports extensive verification of different properties and realization forms. With this tool, a development engineer can verify, during the design phase, if the proposed digital system will present an expected behavior, when it is embedded into a given hardware architecture.

DSVerifier can be regarded as an automated and reliable alternative, when compared with traditional simulation tools. It is freely available for download (Linux x86-64 and x86 versions), including documentation, benchmarks, experimental results presented in previous studies, publications, and source code. For future work, other properties, hardware platforms, and BMC tools will be integrated into DSVerifier, in addition to support for closed-loop systems [14].

References

1. Michael Jackson, "The world and the machine," *ICSE*, pp. 283–292, 1995.
2. Alur et al., "Model-checking for real-time systems," *LICS*, pp. 414–425, 1990.
3. Alur et al., "Model-Checking in Dense Real-Time," *IC* vol. 104 (1), pp. 2–34, 1993.
4. Behrmann et al., "A tutorial on UPPAAL," *SFM-RT*, LNCS 3185, pp. 200–236, 2004.
5. Tripakis et al., "Checking Timed Buechi Automata Emptiness Efficiently," *FMSD*, pp. 267–292, 2005.
6. Magellan, "Hybrid RTL formal verification," `http://www.synopsys.com/tools/verification/functionalverification/pages/magellan.aspx`, Accessed 12 September 2014.
7. Davis TA, Sigmon K, "MATLAB primer," (7. ed.). CRC Press, 2005.
8. Abreu et al., "Verifying Fixed-Point Digital Filters using SMT-Based Bounded Model Checking," *SBrT*, DOI http://dx.doi.org/10.14209/sbrt.2013.57, 2013.
9. Bessa et al., "SMT-Based Bounded Model Checking of Fixed-Point Digital Controllers)," *IECON*, pp. 295–301, 2014.
10. Bessa et al., "Verification of Delta Form Realization in Fixed-Point Digital Controllers Using Bounded Model Checking," *SBESC*, pp. 49–54, 2014.
11. Cordeiro et al., "SMT-Based Bounded Model Checking for Embedded ANSI-C Software," *TSE* vol. 38 (4), pp. 957–974, 2012.
12. D. Beyer, "Software Verification and Verifiable Witnesses - (Report on SV-COMP 2015)," *TACAS*, LNCS 9035, pp. 401–416, 2015.
13. Ogata, K., "Discrete-Time Control Systems," Prentice Hall International editions, Prentice-Hall International, 1995.
14. Platzer, A., "Logic and compositional verification of hybrid systems (invited tutorial)," *CAV*, LNCS 6806, pp. 28–43, 2011.

[3] http://www.oracle.com/technetwork/java/javase/8u40-relnotes-2389089.html