DSVerifier: A Bounded Model Checking Tool for Digital Systems

Hussama Ismail, Iury Bessa, Lucas Cordeiro,
Eddie Batista de Lima Filho and João Edgar Chaves Filho

Electronic and Information Research Center
Federal University of Amazonas
Brazil

Abstract. This work presents the Digital-System 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. It is worth noticing that 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. This way, 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.

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 finite word-length (FWL) effects. Besides, although one may find a myriad of design tools, there is a clear lack of options (tools and methodologies) for validating digital systems, regarding implementation aspects.

Consequently, some automated verification-tools have already been proposed, like UPPAAL [1], Open-Kronos [2], and Maellan [3]. However, such approaches are usually employed for high-level verification and have not been used for verifying resilience, that is, system robustness related to implementation aspects.

Thus, one may notice that there is still a gap, regarding formal verification tools and methodologies for embedded systems.

The present paper addresses this problem by introducing the Digital-System Verifier (DSVerifier) [1], which is a bounded model checking (BMC) tool based on satisfiability modulo theories (SMT). DSVerifier is a powerful tool intended for aiding in the design and verification steps of digital systems, which is more reliable and less laborious than traditional simulation tools (e.g., Matlab [4]).

In previous work [5,6,7], verification methodologies related to overflow, limit cycle, time constraints, stability, and minimum phase, in digital filters and controllers, were already discussed. This tool paper, in turn, focuses on implementation aspects of DSVerifier, which aims to check whether a designed digital system meets the desired performance, when it is embedded into a given hardware with resource limitations.

2 The Digital-System Verifier (DSVerifier)

DSVerifier is an internal module for the efficient SMT-based context-bounded model checker (ESBMC) [8], 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.

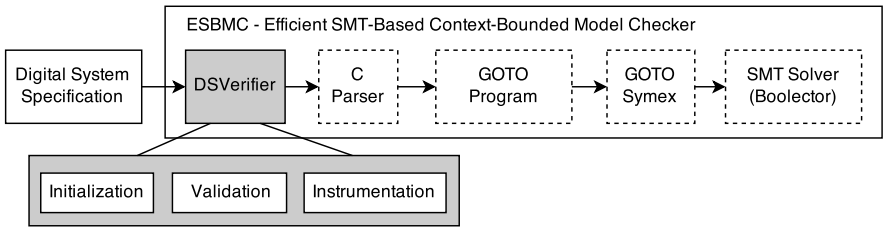

Fig. 1. An overview of the verification architecture.

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.

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 if all required

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

parameters for the verification procedure were correctly provided. In the last step, DSVerifier adds explicit calls to the verification engine (for the evaluated properties), by using functions available in ESBMC (e.g., *__ESBMC_assume* and *__ESBMC_assert*), in order to check for property violations.

As soon as the mentioned procedures of the DSVerifier module are finished, an ANSI-C code file is generated, which can be verified by any C model-checker that supports bit-vectors. 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 programs that make use of bit-vector arithmetic, according to the last edition of the software verification competition [9]. If the verification framework finds a property violation, it produces a counterexample; otherwise, the evaluated design is ready to be embedded into a given computer-based system.

2.1 DSVerifier Features

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

- **Overflow.** When a sum or product exceeds the number representation, 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 zero-input limit cycles, for any initial condition;
- **Stability.** DSVerifier may be used for verifying digital-system stability, considering FWL effects on pole locations;
- **Minimum phase.** DSVerifier may perform the same previous analysis for system zeros, in order to verify minimum phase for digital controllers;
- **Time constraints.** DSVerifier is able to investigate if 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

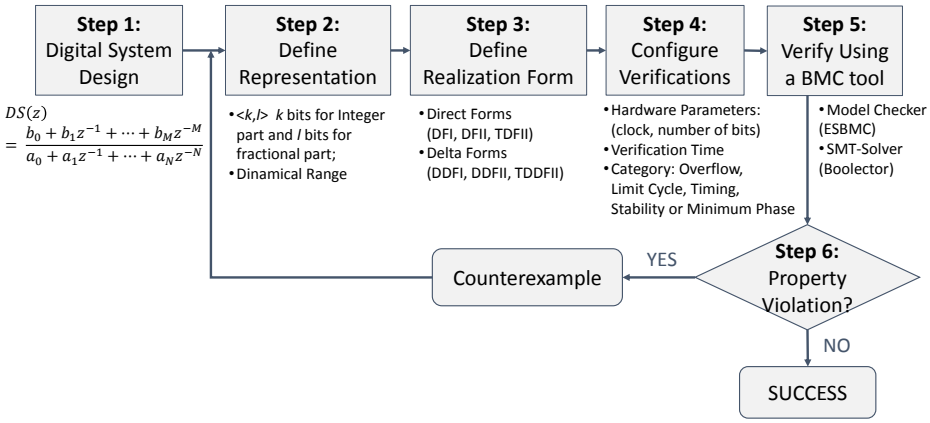

Fig. 2. Proposed methodology for digital-system verification.

then started, in step 5, with the chosen model checking tool (ESBMC is used as back-end for DSVerifier).

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.

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 the book written by Ogata [10]) with a sampling period of $0.5s$.

Command-line Version In this version, the user must provide a description ANSI-C file, as shown in Fig. 3 for the digital controller represented by (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).

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

```
#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:

`esbmc <file> -DSVERIFIER -DREALIZATION=<`i`> -DPROPERTY=<`j`> -DX_SIZE=<`k`> --boolector --no-bounds-check --no-pointer-check --no-div-by-zero-check`,

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, that is, the number of times the digital system will be unfolded. 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). In addition, 5 different properties can be chosen: overflow, limit cycle, timing, stability, and minimum phase.

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

Another interesting feature of the proposed GUI is the parallel execution of verification tasks, which has the potential to decrease the total verification time spent by DSVerifier. The GUI also allows to graphically verify results and to obtain a counterexample with error trace, which helps adjust the verified design. It is worth noticing that the user 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) in `http://www.dsverifier.org/`, including documentation, benchmarks, experimental results presented in previous studies, publications, and source code. In a future work, other verification engines will be integrated into DSVerifier (new properties, etc.), and versions for other operating systems (e.g., Windows and Mac OSX) will be made available.

References

1. Behrmann G, David A, Larsen KG (2004) A tutorial on UPPAAL. In: SFM-RT, LNCS 3185, pp 200–236
2. Tripakis S, Yovine S, Bouajjani A (2005) Checking timed buechi automata emptiness efficiently. In: Formal Methods in System Design, pp 267–292
3. Magellan (2014) Hybrid RTL formal verification. `http://www.synopsys.com/tools/verification/functionalverification/pages/magellan.aspx`, Accessed 12 September 2014
4. Davis TA, Sigmon K (2005) MATLAB primer (7. ed.). CRC Press
5. Abreu RB, Cordeiro LC, Filho EBL (2013) Verifying Fixed-Point Digital Filters using SMT-Based Bounded Model Checking. In: SBrT, DOI http://dx.doi.org/10.14209/sbrt.2013.57
6. Bessa I, Abreu R, Cordeiro L, Filho JE (2014) SMT-Based Bounded Model Checking of Fixed-Point Digital Controllers). In: IECON, pp 295–301, DOI http://dx.doi.org/10.1109/IECON.2014.7048514
7. Bessa I, Ibrahim H, Cordeiro L, Filho JE (2014) Verification of Delta Form Realization in Fixed-Point Digital Controllers Using Bounded Model Checking. In: SBESC, pp 49–54, DOI http://dx.doi.org/10.1109/SBESC.2014.14
8. Cordeiro L, Fischer B, Marques-Silva J (2012) SMT-Based Bounded Model Checking for Embedded ANSI-C Software. IEEE TSE 38(4):957–974, DOI http://dx.doi.org/10.1109/TSE.2011.59
9. D. Beyer (2015), Software Verification and Verifiable Witnesses - (Report on SV-COMP 2015). In: TACAS, LNCS 9035, pp. 401–416, DOI http://dx.doi.org/10.1007/978-3-662-46681-0_31
10. Ogata K (1995) Discrete-Time Control Systems. Prentice Hall International editions, Prentice-Hall International

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

Appendix - Oral Presentation Plan

The oral presentation plan consists of three parts. First, a brief motivation about general digital-system problems, which are related to FWL effects, by emphasizing the effort spent in the design process for preventing these effects. Second, an overview about the DSVerifier implementation and its verification engine. Third, a quick demonstration of DSVerifier, considering both command-line and GUI versions.

1. Introduction and Motivation (20%)

   - Brief introduction about Digital Systems
   - FWL Effects
   - Why to use bounded model checking techniques for digital system verification?

2. DSVerifier Implementation Aspects (20%)

   - Properties supported by DSVerifier
   - DSVerifier implementation and architecture

3. DSVerifier Demonstration (60%)

   In this part, we will demonstrate the DSVerifier usage, based on our set of benchmarks.

   Video demonstration: `https://www.youtube.com/watch?v=BIkm8XUQEdo`

The tools are available for downloading at:

DSVerifier: `http://www.dsverifier.org/`
ESBMC: `http://www.esbmc.org/`