DSValidator: An Automated Counterexample Reproducibility Tool for Digital Systems

Joint work with Lennon Chaves, Iury Bessa, and Daniel Kroening

Lucas Cordeiro
University of Oxford
lucas.cordeiro@cs.ox.ac.uk
Establish Trust in Verification Results

Specification

Implementation

Digital Controller and Filter
Establish Trust in Verification Results

- Specification
- Digital System Verifiers
- Implementation
- Digital Controller and Filter
Establish Trust in Verification Results

- Specification
- Implementation
- Digital Controller and Filter

Verification Successful

Digital System Verifiers

CE Reproducible

CE Irreproducible
Establish Trust in Verification Results

Digital System Verifiers

DSValidator

CE Reproducible

Counter-example

Fix the implementation

Verification Successful

Specification

Implementation

Digital Controller and Filter
Establish Trust in Verification Results

- Specification
- Implementation
- Digital Controller and Filter

Digital System Verifiers

- Verification Successful
- CE Irreproducible
- Incorrect result
- Counter-example
- CE Reproducible

Fix the implementation
Verification & Validation Methodology

Verification Steps

1. **Step 1:** Digital System Design
2. **Step 2:** Define Representation
3. **Step 3:** Define Realization Form
4. **Step 4:** Configure Verification
5. **Step 5:** Verifier/Solver
   - Property Violation?
     - **Counterexample**
     - **DSValidator**
6. **Step 6:** Property Violation?
   - YES
   - Fix the implementation
   - **DSValidator**
   - Verification Result (Exchangeable Format)
   - **SUCCESS
   - NO

Validation Steps
Verification & Validation Methodology

Verification Steps

Step 1: Digital System Design

Step 2: Define Representation

Step 3: Define Realization Form

Step 4: Configure Verification

Step 5: Verifier/Solver

Fix the implementation

Incorrect result

Verification Result (Exchangeable Format)

Validation Steps

DSValidator

Step 6: Property Violation?

YES

Counterexample

NO

SUCCESS
Verification & Validation Methodology

Verification Steps

1. **Step 1:** Digital System Design
2. **Step 2:** Define Representation
3. **Step 3:** Define Realization Form
4. **Step 4:** Configure Verification
5. **Step 5:** Verifier/Solver

Verification Result (Exchangeable Format)

Validation Steps

- **DSValidator**
- **Step 6:** Property Violation?
  - YES: Counterexample
  - NO: SUCCESS

Incorrect result

Fix the implementation
Verification & Validation Methodology

Verification Steps

1. **Step 1:** Digital System Design
2. **Step 2:** Define Representation
3. **Step 3:** Define Realization Form
4. **Step 4:** Configure Verification
5. **Step 5:** Verifier/Solver

Validation Steps

- **DSValidator**
- **Step 6:** Property Violation?
  - YES: Counterexample
  - NO: SUCCESS
- Incorrect result

Fix the implementation

Verification Result (Exchangeable Format)
Verification & Validation Methodology

Verification Steps

Step 1: Digital System Design
Step 2: Define Representation
Step 3: Define Realization Form
Step 4: Configure Verification
Step 5: Verifier/Solver

Validation Steps

DSValidator

Property Violation?

Verification Result (Exchangeable Format)

Correct result

Fix the implementation

YES

Counterexample

NO

SUCCESS

Fix the implementation

Incorrect result
Verification & Validation Methodology

Verification Steps

1. Digital System Design
2. Define Representation
3. Define Realization Form
4. Configure Verification
5. Verifier/Solver

Verification Result (Exchangeable Format)

Validation Steps

DSValidator

YES
NO
SUCCESS

Fix the implementation
Counterexample
Incorrect result
Property Violation?
Verification & Validation Methodology

**Verification Steps**

1. **Step 1:** Digital System Design
2. **Step 2:** Define Representation
3. **Step 3:** Define Realization Form
4. **Step 4:** Configure Verification
5. **Step 5:** Verifier/Solver

**Validation Steps**

- **DSValidator**
  - **Step 6:** Property Violation?
    - YES: Counterexample
    - NO: SUCCESS

- Fix the implementation
- Incorrect result
- Verification Result (Exchangeable Format)
Verification & Validation Methodology

Verification Steps

Step 1: Digital System Design
Step 2: Define Representation
Step 3: Define Realization Form
Step 4: Configure Verification
Step 5: Verifier/Solver

Validation Steps

DSValidator

Verification Result (Exchangeable Format)

Incorrect result

Fix the implementation

YES

Counterexample

Step 6: Property Violation?

NO

SUCCESS
Establish trust in verification results for digital systems
Objectives

Establish trust in verification results for digital systems

- Propose a format to represent the counterexamples that can be used by any verifier
Objectives

Establish trust in verification results for digital systems

- Propose a format to represent the counterexamples that can be used by any verifier
- Reproduce counterexamples that refute properties related to limit cycle, overflow, stability and minimum-phase
Objectives

Establish trust in verification results for digital systems

- Propose a format to represent the counterexamples that can be used by any verifier
- Reproduce counterexamples that refute properties related to limit cycle, overflow, stability and minimum-phase
- Validate a set of intricate counterexamples for digital controllers used in a real quadrotor attitude system
DSVerifier Counterexample Format

- A counterexample is a **trace** that **shows** that a given **property does not hold** in the model represented by a **state transition system**

<table>
<thead>
<tr>
<th>Property</th>
<th>LIMIT_CYCLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Numerator</td>
<td>{ 2002, -4000, 1998 }</td>
</tr>
<tr>
<td>Denominator</td>
<td>{ 1, 0, -1 }</td>
</tr>
<tr>
<td>X_Size</td>
<td>10</td>
</tr>
<tr>
<td>Sample_Time</td>
<td>0.001</td>
</tr>
<tr>
<td>Implementation</td>
<td>(&lt;13,3&gt;)</td>
</tr>
<tr>
<td>Numerator (fixed-point)</td>
<td>{ 2002, -4000, 1998 }</td>
</tr>
<tr>
<td>Denominator (fixed-point)</td>
<td>{ 1, 0, -1 }</td>
</tr>
<tr>
<td>Realization</td>
<td>DFI</td>
</tr>
<tr>
<td>Dynamical_Range</td>
<td>{ -1, 1 }</td>
</tr>
<tr>
<td>Initial_States</td>
<td>{ -0.875, 0, -1 }</td>
</tr>
<tr>
<td>Inputs</td>
<td>{ 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5 }</td>
</tr>
<tr>
<td>Outputs</td>
<td>{ 0, -1, 0, -1, 0, -1, 0, -1, 0, -1 }</td>
</tr>
</tbody>
</table>
DSVerifier Counterexample Format

- A counterexample is a trace that shows that a given property does not hold in the model represented by a state transition system.

<table>
<thead>
<tr>
<th>Property</th>
<th>LIMIT_CYCLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Numerator</td>
<td>{ 2002, -4000, 1998 }</td>
</tr>
<tr>
<td>Denominator</td>
<td>{ 1, 0, -1 }</td>
</tr>
<tr>
<td>X_Size</td>
<td>10</td>
</tr>
<tr>
<td>Sample_Time</td>
<td>0.001</td>
</tr>
<tr>
<td>Implementation</td>
<td>&lt;13,3&gt;</td>
</tr>
<tr>
<td>Numerator (fixed-point)</td>
<td>{ 2002, -4000, 1998 }</td>
</tr>
<tr>
<td>Denominator (fixed-point)</td>
<td>{ 1, 0, -1 }</td>
</tr>
<tr>
<td>Realization</td>
<td>DFI</td>
</tr>
<tr>
<td>Dynamical_Range</td>
<td>{ -1, 1 }</td>
</tr>
<tr>
<td>Initial_States</td>
<td>{ -0.875, 0, -1 }</td>
</tr>
<tr>
<td>Inputs</td>
<td>{ 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5 }</td>
</tr>
<tr>
<td>Outputs</td>
<td>{ 0, -1, 0, -1, 0, -1, 0, -1, 0, -1 }</td>
</tr>
</tbody>
</table>
DSVerifier Counterexample Format

- A counterexample is a trace that shows that a given property does not hold in the model represented by a state transition system.

<table>
<thead>
<tr>
<th>Property</th>
<th>LIMIT_CYCLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Numerator</td>
<td>{ 2002, -4000, 1998 }</td>
</tr>
<tr>
<td>Denominator</td>
<td>{ 1, 0, -1 }</td>
</tr>
<tr>
<td>X_Size</td>
<td>10</td>
</tr>
<tr>
<td>Sample_Time</td>
<td>0.001</td>
</tr>
<tr>
<td>Implementation</td>
<td>&lt;13, 3&gt;</td>
</tr>
<tr>
<td>Numerator (fixed-point)</td>
<td>{ 2002, -4000, 1998 }</td>
</tr>
<tr>
<td>Denominator (fixed-point)</td>
<td>{ 1, 0, -1 }</td>
</tr>
<tr>
<td>Realization</td>
<td>DFI</td>
</tr>
<tr>
<td>Dynamical_Range</td>
<td>{ -1, 1 }</td>
</tr>
<tr>
<td>Initial_States</td>
<td>{ -0.875, 0, -1 }</td>
</tr>
<tr>
<td>Inputs</td>
<td>{ 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5 }</td>
</tr>
<tr>
<td>Outputs</td>
<td>{ 0, -1, 0, -1, 0, -1, 0, -1, 0, -1 }</td>
</tr>
</tbody>
</table>
DSVerifier Counterexample Format

- A counterexample is a trace that shows that a given property does not hold in the model represented by a state transition system

```
Property = LIMIT_CYCLE
Numerator  = { 2002, -4000, 1998 }
Denominator = { 1, 0, -1 }
X_Size = 10
**Sample_Time** = 0.001
Implementation = <13,3>
Numerator (fixed-point) = { 2002, -4000, 1998 }
Denominator (fixed-point) = { 1, 0, -1 }
Realization = DFI
Dynamical_Range = { -1, 1 }
Initial_States = { -0.875, 0, -1 }
Inputs = { 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5 }
Outputs = { 0, -1, 0, -1, 0, -1, 0, -1, 0, -1 }
```
A counterexample is a trace that shows that a given property does not hold in the model represented by a state transition system.

```
Property = LIMIT_CYCLE
Numerator = { 2002, -4000, 1998 }
Denominator = { 1, 0, -1 }
X_Size = 10
Sample_Time = 0.001
Implementation = <13,3>
Numerator (fixed-point) = { 2002, -4000, 1998 }
Denominator (fixed-point) = { 1, 0, -1 }
Realization = DFI
Dynamical_Range = { -1, 1 }
Initial_States = { -0.875, 0, -1 }
Inputs = { 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5 }
Outputs = { 0, -1, 0, -1, 0, -1, 0, -1, 0, -1 }
```
A counterexample is a **trace** that shows that a given **property does not hold** in the model represented by a **state transition system**

```
Property = LIMIT_CYCLE
Numerator  = { 2002, -4000, 1998 }
Denominator  = { 1, 0, -1 }
X_Size = 10
Sample_Time = 0.001
Implementation = <13,3>
Numerator (fixed-point) = { 2002, -4000, 1998 }
Denominator (fixed-point) = { 1, 0, -1 }
Realization = DFI
Dynamical_Range = { -1, 1 }
Initial_States = { -0.875, 0, -1 }
Inputs = { 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5 }
Outputs = { 0, -1, 0, -1, 0, -1, 0, -1, 0, -1 }
```
DSVerifier Counterexample Format

- A counterexample is a trace that shows that a given property does not hold in the model represented by a state transition system.

---

Property = LIMIT_CYCLE
Numerator  = \{ 2002, -4000, 1998 \}
Denominator = \{ 1, 0, -1 \}
X_Size = 10
Sample_Time = 0.001
Implementation = <13,3>
Numerator (fixed-point) = \{ 2002, -4000, 1998 \}
Denominator (fixed-point) = \{ 1, 0, -1 \}
Realization = DFI
Dynamical_Range = \{ -1, 1 \}
Initial_States = \{ -0.875, 0, -1 \}
Inputs = \{ 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5 \}
Outputs = \{ 0, -1, 0, -1, 0, -1, 0, -1, 0, -1 \}
A counterexample is a **trace** that **shows** that a given **property does not hold** in the model represented by a **state transition system**

<table>
<thead>
<tr>
<th>Property = LIMIT_CYCLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Numerator = { 2002, -4000, 1998 }</td>
</tr>
<tr>
<td>Denominator = { 1, 0, -1 }</td>
</tr>
<tr>
<td>X_Size = 10</td>
</tr>
<tr>
<td>Sample_Time = 0.001</td>
</tr>
<tr>
<td>Implementation = &lt;13,3&gt;</td>
</tr>
<tr>
<td>Numerator (fixed-point) = { 2002, -4000, 1998 }</td>
</tr>
<tr>
<td>Denominator (fixed-point) = { 1, 0, -1 }</td>
</tr>
<tr>
<td>Realization = DFI</td>
</tr>
<tr>
<td><strong>Dynamical_Range</strong> = { -1, 1 }</td>
</tr>
<tr>
<td>Initial_States = { -0.875, 0, -1 }</td>
</tr>
<tr>
<td>Inputs = { 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5 }</td>
</tr>
<tr>
<td>Outputs = { 0, -1, 0, -1, 0, -1, 0, -1, 0, -1 }</td>
</tr>
</tbody>
</table>
DSVerifier Counterexample Format

- A counterexample is a **trace** that **shows** that a given **property does not hold** in the model represented by a **state transition system**

```
Property = LIMIT_CYCLE
Numerator = { 2002, -4000, 1998 }
Denominator = { 1, 0, -1 }
X_Size = 10
Sample_Time = 0.001
Implementation = <13,3>
Numerator (fixed-point) = { 2002, -4000, 1998 }
Denominator (fixed-point) = { 1, 0, -1 }
Realization = DFI
Dynamical_Range = { -1, 1 }
Initial_States = { -0.875, 0, -1 }
Inputs = { 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5 }
Outputs = { 0, -1, 0, -1, 0, -1, 0, -1, 0, -1 }
```
A counterexample is a trace that shows that a given property does not hold in the model represented by a state transition system.

Property = OVERFLOW
Numerator = { 2002, -4000, 1998 }
Denominator = { 1, 0, -1 }
X_Size = 10
Sample_Time = 0.02
Implementation = <10,6>
Numerator (fixed-point) = { 2002, -4000, 1998 }
Denominator (fixed-point) = { 1, 0, -1 }
Realization = DFI
Dynamic_Range = {-1, 1}
Inputs = { -1, -0.75, 0.0, -0.5, 0.0, 0.25, 1, -0.5, 0.078125, 0.6875 }
Outputs = { -2002, 2498.5, -1000.0, -1.0, 1000.0, -499.5, 2002, -5001, 6156, -4936.125 }
A counterexample is a trace that shows that a given property does not hold in the model represented by a state transition system.

Property = OVERFLOW
Numerator = \{ 2002, -4000, 1998 \}
Denominator = \{ 1, 0, -1 \}
X_Size = 10
Sample_Time = 0.02
Implementation = <10,6>
Numerator (fixed-point) = \{ 2002, -4000, 1998 \}
Denominator (fixed-point) = \{ 1, 0, -1 \}
Realization = DFI
Dynamic_Range = \{ -1, 1 \}
Inputs = \{ -1, -0.75, 0.0, -0.5, 0.0, 0.25, 1, -0.5, 0.078125, 0.6875 \}
Outputs = \{ -2002, 2498.5, -1000.0, -1.0, 1000.0, -499.5, 2002, -5001, 6156, -4936.125 \}
DSValidator Reproducibility Engine

- Supports digital systems (controller and filter) represented by a transfer function:

\[
H(z) = \frac{B(z)}{A(z)} = \frac{b_0 + b_1z^{-1} + \ldots + b_Mz^{-M}}{a_0 + a_1z^{-1} + \ldots + a_Nz^{-N}}
\]
DSValidator Reproducibility Engine

- Supports digital systems (controller and filter) represented by a transfer function:

\[ H(z) = \frac{B(z)}{A(z)} = \frac{b_0 + b_1 z^{-1} + \ldots + b_M z^{-M}}{a_0 + a_1 z^{-1} + \ldots + a_N z^{-N}} \]

- Computes
  - finite-word lengths effects over the \(a_k\) and \(b_k\) coefficients
  - roots of a polynomial for stability and minimum-phase
DSValidator Reproducibility Engine

- Supports digital systems (controller and filter) represented by a transfer function:

\[ H(z) = \frac{B(z)}{A(z)} = \frac{b_0 + b_1 z^{-1} + \ldots + b_M z^{-M}}{a_0 + a_1 z^{-1} + \ldots + a_N z^{-N}} \]

- Computes
  - finite-word lengths effects over the \( a_k \) and \( b_k \) coefficients
  - roots of a polynomial for stability and minimum-phase

- Unrolls the system for a given realization form
  - overflow, granular LCO, overflow LCO

\[ y(n) = -\sum_{k=1}^{N} a_k y(n-k) + \sum_{k=0}^{M} b_k x(n-k) \]
DSValidator Validation Process

**Step 1: Extraction**
- obtains the counterexample from the verifier

**Step 2: Parser**
- MATLAB Variables

**Step 3: Simulation**
- Outputs Computation

**Step 4: Comparison**
- Verification Output vs Simulation Output

**Step 5: Report**
- Successful
- Failed

Counterexamples

Counterexample .out

Automatic Counterexample Validation Process
DSValidator Validation Process

- **Extraction**
  - obtains the counterexample from the verifier
- **Parser**
  - converts all counterexample attributes into variables
DSValidator Validation Process

**Automatic Counterexample Validation Process**

1. **Step 1: Extraction**
   - Counterexamples
   - Counterexample .out files

2. **Step 2: Parser**
   - MATLAB Variables

3. **Step 3: Simulation**
   - Outputs Computation

4. **Step 4: Comparison**
   - Verification Output vs Simulation Output

5. **Step 5: Report**
   - .MAT file

   - Successful
   - Failed

**• Extraction**
  - obtains the counterexample from the verifier

**• Parser**
  - converts all counterexample attributes into variables

**• Simulation**
  - simulates the counterexample (violation) for the failed property
DSValidator Validation Process

**Automatic Counterexample Validation Process**

- **Step 1: Extraction**
  - obtains the counterexample from the verifier
- **Step 2: Parser**
  - converts all counterexample attributes into variables
- **Step 3: Simulation**
  - simulates the counterexample (violation) for the failed property
- **Step 4: Comparison**
  - checks MATLAB simulation vs verifier output
- **Step 5: Report**

Counterexamples

- .out files

Counterexample .out

MATLAB Variables

Outputs Computation

Verification Output vs Simulation Output

Successful

Failed

.com file
DSValidator Validation Process

• Extraction
  ‣ obtains the counterexample from the verifier

• Parser
  ‣ converts all counterexample attributes into variables

• Simulation
  ‣ simulates the counterexample (violation) for the failed property

• Comparison
  ‣ checks MATLAB simulation vs verifier output

• Report
  ‣ stores the counterexample in a .MAT file and reports its reproducibility
DSValidator Features

• Validation Functions
  ▸ reproduce the validation steps (e.g., extraction, parsing, simulation, comparison and report)
DSValidator Features

• Validation Functions
  ‣ reproduce the validation steps (e.g., extraction, parsing, simulation, comparison and report)

• Properties
  ‣ checks and validates overflow, limit-cycle, stability and minimum-phase
DSValidator Features

• **Validation Functions**
  ‣ reproduce the validation steps (e.g., extraction, parsing, simulation, comparison and report)

• **Properties**
  ‣ checks and validates overflow, limit-cycle, stability and minimum-phase

• **Realization**
  ‣ reproduces realization forms to validate overflow and limit-cycle (for direct and delta forms)
DSValidator Features

• Validation Functions
  ‣ reproduce the validation steps (e.g., extraction, parsing, simulation, comparison and report)

• Properties
  ‣ checks and validates overflow, limit-cycle, stability and minimum-phase

• Realization
  ‣ reproduces realization forms to validate overflow and limit-cycle (for direct and delta forms)

• Numerical Functions
  ‣ performs the quantization process, select rounding and overflow mode, fixed-point operations and delta operator
Graphical Functions

plot_limit_cycle(system)

plot_overflow(system)
DSValidator Usage

- MATLAB Command Line:
  - `validation(path, property, ovmode, rmode, filename)`
  - **path**
    - is the directory with the counterexample
  - **property**
    - “m” for minimum phase
    - “s” for stability
    - “o” for overflow
    - “lc” for limit cycle
  - **ovmode**
    - overflow mode: `wrap` or `saturate`
  - **rmode**
    - rounding mode: `round`, `float` or `ceil`
  - **filename**
    - represents the `.MAT` filename, which is generated after the validation process; by default, the `.MAT` file is named `digital_system`
Case Study: Digital Controllers for UAV

- 11 digital controllers extracted from a quadrotor unmanned aerial vehicle
- Overflow, minimum-phase, stability and limit-cycle
- 8-, 16- and 32-bit
- DFI, DFII and TDFII
Experimental Evaluation

• RQ1 (performance) do the executable test cases take considerably less effort than verification?

• RQ2 (sanity check) are the counterexamples sound and can their reproducibility be confirmed?

<table>
<thead>
<tr>
<th>Property</th>
<th>CE Reproducible</th>
<th>CE Irreproducible</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overflow</td>
<td>24</td>
<td>0</td>
<td>0.190 s</td>
</tr>
<tr>
<td>Limit Cycle</td>
<td>26</td>
<td>1</td>
<td>0.483 s</td>
</tr>
<tr>
<td>Minimum-Phase</td>
<td>54</td>
<td>0</td>
<td>0.012 s</td>
</tr>
<tr>
<td>Stability</td>
<td>54</td>
<td>0</td>
<td>0.188 s</td>
</tr>
</tbody>
</table>

• For the limit cycle property:
  ▪ it did not take into account overflow in intermediate operations to compute the system’s output using the DFII realization form
Github commit to fix the bug

enabling overflow mode saturation for intermediate operations.

lennonchaves committed on Nov 4, 2016

Showing 2 changed files with 9 additions and 2 deletions.

```c
@@ -303,6 +303,7 @@ void fxp_to_double_array(double f[], fxp_t r[], int N) {

    fxp_t fxp_abs(fxp_t a) {
        fxp_t tmp;
        tmp = ((a < 0) ? -(fxp_t)(a) : a);
+       tmp = fxp_quantize(tmp);
    return tmp;

@@ -315,6 +316,7 @@ fxp_t fxp_abs(fxp_t a) {

    fxp_t fxp_add(fxp_t aadd, fxp_t badd) {
        fxp_t tmpadd;
        tmpadd = ((fxp_t)(aadd) + (fxp_t)(badd));
+       tmpadd = fxp_quantize(tmpadd);
    return tmpadd;
```
Conclusions and Future Work

- DSVValidator reproduces counterexamples generated for digital controllers of a quadrotor attitude system
  - implementation aspects
  - stability, minimum-phase, limit-cycle and overflow
Conclusions and Future Work

- DSVValidator reproduce counterexamples generated for digital controllers of a quadrotor attitude system
  - implementation aspects
  - stability, minimum-phase, limit-cycle and overflow
- There is no other automated MATLAB toolbox that can reproduce counterexamples for digital system generated by verifiers
  - identify the reason why the counterexample cannot be reproduced
Conclusions and Future Work

• DSValidator reproduces counterexamples generated for digital controllers of a quadrotor attitude system
  ‣ implementation aspects
  ‣ stability, minimum-phase, limit-cycle and overflow

• There is no other automated MATLAB toolbox that can reproduce counterexamples for digital system generated by verifiers
  ‣ identify the reason why the counterexample cannot be reproduced

• As future work, we expect to contribute to digital system validation by supporting further verifiers (e.g., Polyspace)
  ‣ Simulate the hybrid dynamics over the continuous time

A BOUNDED MODEL CHECKING TOOL FOR DIGITAL SYSTEMS

DSVerify (Digital Systems Verify) is a bounded model checker to aid engineers to check for overflow, limit cycle, error, timing, stability, and minimum phase in digital systems, considering finite word length (FWL) effects.

- Stability
- Overflow
- Limit-Cycle
- Minimum-Phase
- Quantization Error
- Timing Constraints
- Robust Stability