Cyber-Component Verification Using HybridSAL

Ashish Tiwari

SRI International, Menlo Park, CA. ashish.tiwari@sri.com

Abstract. This tutorial describes the process of verifying cyber-components of a complex cyber-physical system. We use the HybridSal verification tool, which can verify hybrid dynamical systems; that is, systems that have dynamics described by a combination of differential equations and discrete state transitions.

1 The Tools

The tools used in the verification of cyber-controllers are the following:

Matlab Simulink Stateflow: We assume that the cyber-components are designed using Matlab's Simulink Stateflow language. Any changes to the controller models are performed using Matlab.
Preferred version: R13b

Matlab extension for specifying LTL properties: VU has developed scripts that extend the graphical user interface of Matlab Simulink/Stateflow with dialog boxes for specifying temporal logic properties using a pattern system. Desired properties of the controllers are attached to the Simulink subsystem or Stateflow chart using this extension.

MDL2MGACyber.exe: VU has developed a tool that translates a Matlab model (exported as a .mdl file) into an XML representation (cyber-composition language). Both the system and the LTL property are included in the XML file.
This tool is part of META release.

cc_hra_verifier.exe: SRI has developed a tool that translates models in the cyber composition language (.xml files) into the HybridSal formal verification language, and then uses the HybridSal relational abstracter, combined with the SAL infinite bounded model checker, to verify the LTL properties specified by the user using the Matlab extension described above.
This tool is currently available as an executable (.exe) file.

The Output Visualizer: SRI has developed scripts that will enable visualization of the output generated by `cc_hra_verifier.exe` in the dashboard.
The visualization and integration scripts are in the Vanderbilt SVN and available for use.

2 A Simple Exercise

We describe a simple verification exercise using `cc_hra_verifier`. It consists of the following steps.

- Building/editing the cyber model and annotating it with LTL properties
- Converting the Matlab model into the cyber composition language
- Creating a HybridSal representation of the model and the properties
- Verifying the HybridSal model and presenting the results.

2.1 Building/editing the cyber model

We start with controllers designed by VU. The Matlab files for these controllers are:

- Torque_Converter_control.mdl
- TorqueReductionSignal.mdl
- SimplifiedShiftController.mdl

These files can be opened in Simulink and edited. For example, Figure 1 shows the `SimplifiedShiftController` in Simulink.

If the LTL-extension for Matlab, developed by Vanderbilt, is installed, then LTL properties can be attached to these models directly by right clicking on the model. Figure 2 shows a screenshot of the process of attaching an LTL property to the top-level Simulink model of the shift controller. Note that the LTL property specifier is template-based – there are predefined LTL templates that the user can pick. In the example in Figure 2, we have picked the template called `Response`, which is useful for specifying properties that state that something should (eventually) happen if something else has occured. Details of the different LTL templates can be found on the website
`http://patterns.projects.cis.ksu.edu/documentation/patterns/ltl.shtml`

The LTL property specification window that opens up when clicking on the `Response` button is shown in Figure 3. Properties can be specified on the input and output port names of the Simulink top-level block. In the example shown in Figure 3, the input variable `driver_gear_select` and the output variable `gear_selected` are used to specify the property. Arithmetic operators such as $==, \leq, \geq, <, >$ can be used to specify the property. We note that LTL operators `always` (written as `G`) and `eventually` (written as `F`) can still be used inside the property specification blocks of the larger LTL template. In the example shown in Figure 3, we have used the `G` operator to say that if the input `driver_gear_select` is consistently set to 1, then eventually the output `gear_selected` will take a value that is at most 1.

Once the text blocks have been filled, the model and the LTL properties can both be saved using the standard Simulink `save as` command. Models (and the properties) are saved as .mdl files. Multiple properties can be attached to the same model.

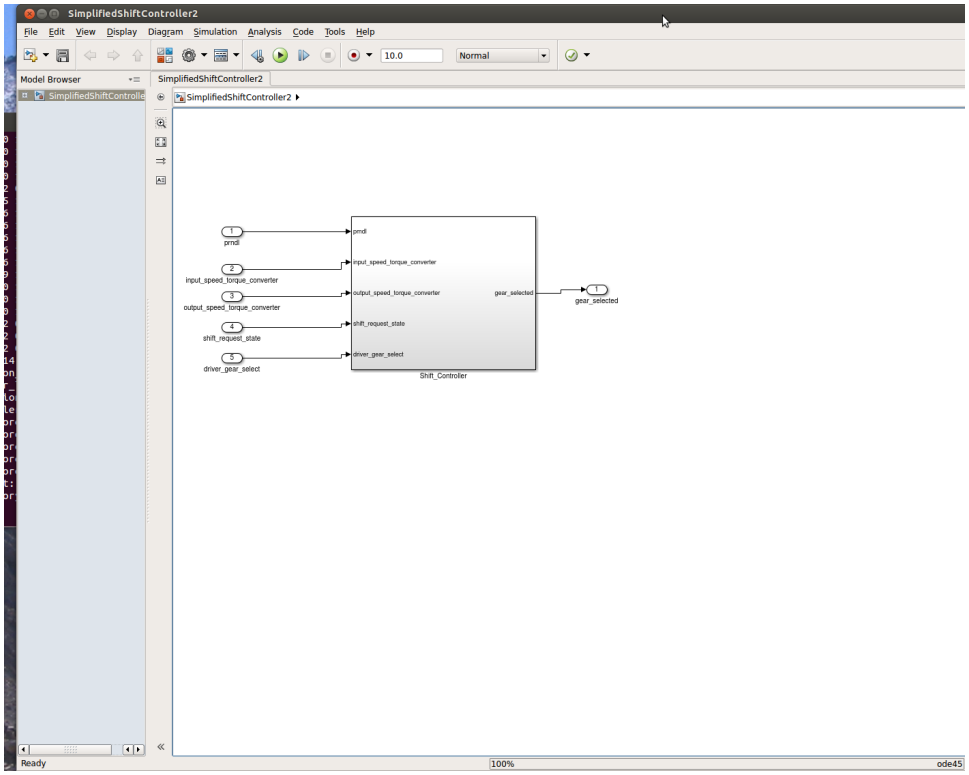

Fig. 1. SimplifiedShiftController in Simulink

2.2 Converting Matlab to Cyber Composition language

Using the `MDL2MGACyber.exe` tool, an XML representation of the models is generated. The details of calling this conversion tool and its interface in the META dashboard is not described in this tutorial. For purposes of verification, all we really need is the XML files generated from the MDL files. Corresponding to the above mdl files, we get the following files, assuming that the models are saved with the same name as before even after adding the LTL properties.

- Torque_Converter_control.xml
- TorqueReductionSignal.xml
- SimplifiedShiftController.xml

The `cc_hra_verifier` distribution also contains an example `SimplifiedControllerCyberWithProp.xml` file that contains all the three components above in a single file. Note that the LTL properties are included in the XML files.

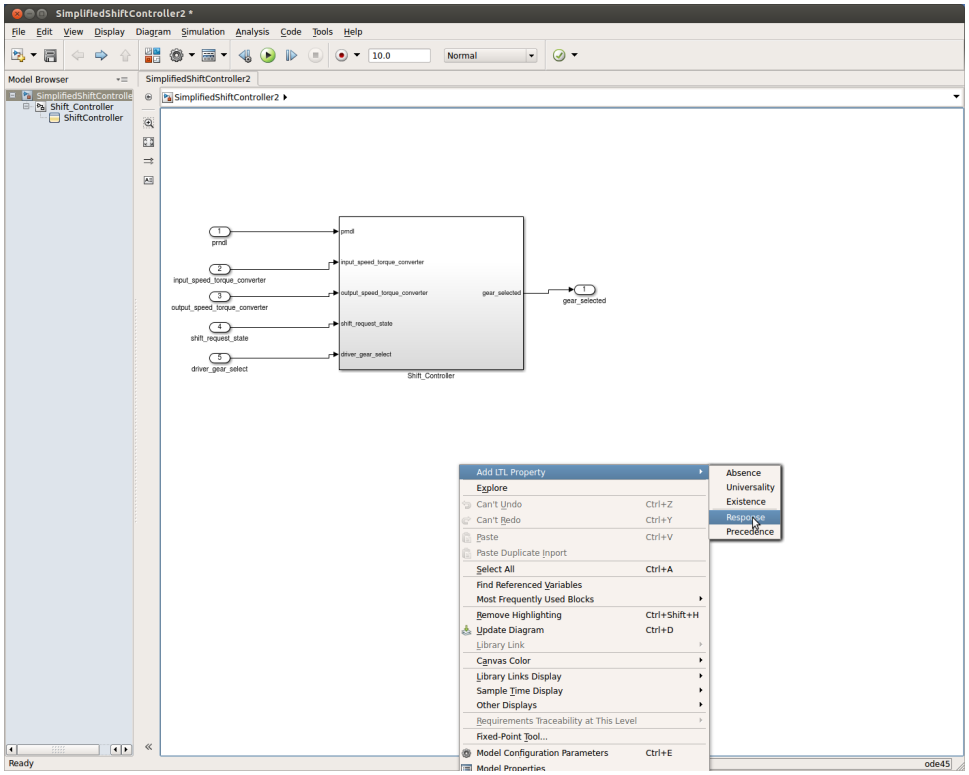

Fig. 2. Inserting LTL property into the SimplifiedShiftController model in Simulink. Vanderbilt's LTL extension must be installed over the basic Matlab Simulink/Stateflow installation.

2.3 Verifying the CyberComposition XML

The tool `cc_hra_verifier` takes as input the XML file generated above and outputs the result of verifying the LTL properties embedded inside the XML file against the model in the same file.

Internally, the `cc_hra_verifier` tool performs the following actions:

1. First, the tool converts the XML into HybridSal. The result of the conversion is a new file called `<filename>.hsal`, where `<filename` is the basename of the XML file. The HybridSal file will contain the model and all the LTL properties that were in the XML file.
   For example, if the `cc_hra_verifier` tool is used on `SimplifiedShiftController.xml`, then it will generate a file called `SimplifiedShiftController.hsal`.
   Interested users can open this file and see the formal representation of the cyber-models and the LTL properties.
2. Second, a relational abstraction of the HybridSal model is constructed. The result is stored in a file with the same filename, but with extension ".sal".

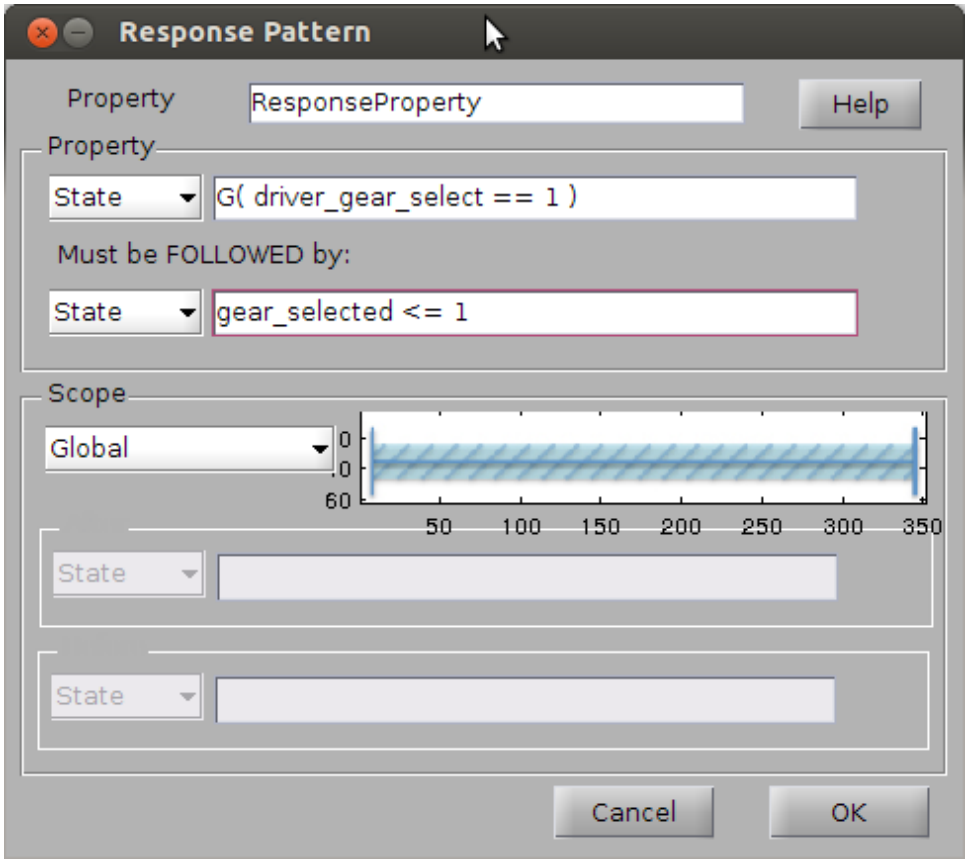

Fig. 3. SimplifiedShiftController in Simulink

In the above example, the tool will create a new file called `SimplifiedShiftController.sal`.

3. Third, the SAL model is model-checked using an infinite-state model checker. The output is either a counter-example for the property, or a statement that no counter-example was found.
   The result is stored in a file called `<filename>Result.txt`. For the example above, `cc_hra_verifier` would generate the results in the file `SimplifiedShiftControllerResult.sal`.

Design/property changes can be made using Matlab on the Simulink/Stateflow models. The verification process can then be repeated. For example, the stateflow chart in the shift controller, which is shown in Figure **??**, can be edited – by adding transitions, removing transitions, or changing the guards or actions, etc. Alternatively, it is also possible to edit the intermediate HybridSal or Sal files directly and the verification tools can be run on these file using command-line invocation – this can save time, but the edits are not carried back to the Matlab models.

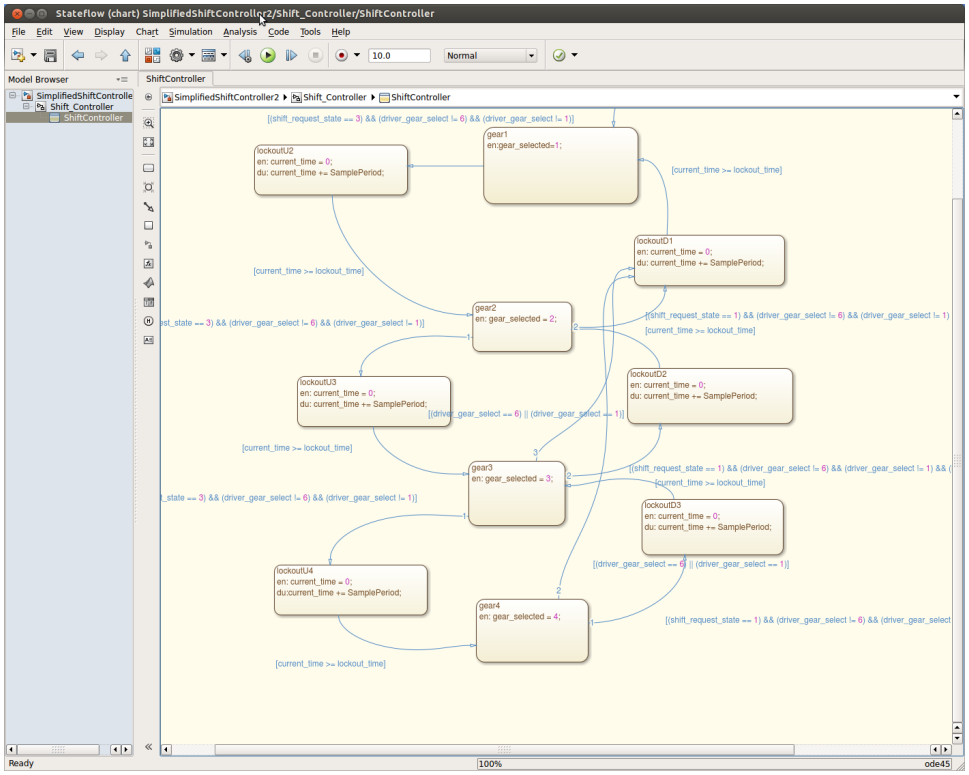

Fig. 4. The stateflow chart inside the toplevel `ShiftController` model. This chart is opened here in Matlab, and it can be edited, saved, and re-verified for the LTL properties attached to the enclosing toplevel Simulink subsystem.

3 Displaying Results in the Dashboard

We have also developed scripts that enable integration of the tool with the dashboard. In particular, we have scripts that can display the output of the verification tool `cc_hra_verifier` in the dashboard.

Figure 5 shows the top-level display for the results. Assuming there were two properties named `p2` and `AbsenceProperty` – the verification results are shown as in Figure 5. The green color indicates that Property `p2` was verified, and the red color indicates that Property `AbsenceProperty` was violated.

Further details about the verification results can be obtained by clicking on the two properties. Clicking on Property `p2` shows the screenshot in Figure 6, while clicking on Property `AbsenceProperty` shows the screenshot in Figure 7.

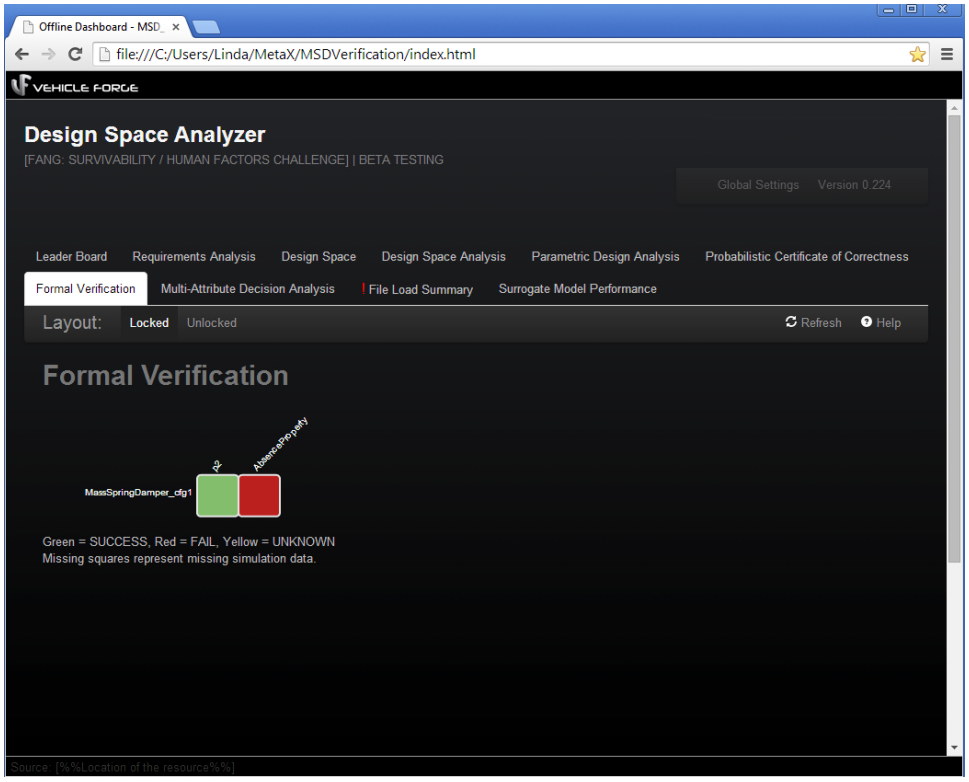

Fig. 5. Top-level display for the verification results in the dashboard. Note that there are two properties, called `p2` and `AbsenceProperty`, whose results are displayed. One property, namely `p2`, was verified, but the other one, namely `AbsenceProperty`, was violated.

4 Remarks

In the above tutorial, only the cyber components were being analyzed. The plant model is completely abstracted. Hence, the analysis can be coarse. But, the analysis can still be useful, especially if care is taken in specifying the LTL properties. In particular, LTL properties should constrain the inputs of the controllers, and then check that the response of the controller is appropriate for that scenario. The predefined LTL templates may not be sufficient for this purpose.

The properties `p1` and `p2` in the HybridSal files were included to illustrate the utility. Property `p1` says that if the input `shift_requested` is consistently set to 1, then after a few steps, the output `gear_selected` of the controller is 1. This property was found to be invalid. Property `p2` says that if the input `shift_requested` is consistently set to 1, then after a few steps, the output `gear_selected` of the controller is at most 2. No counter-examples were found for

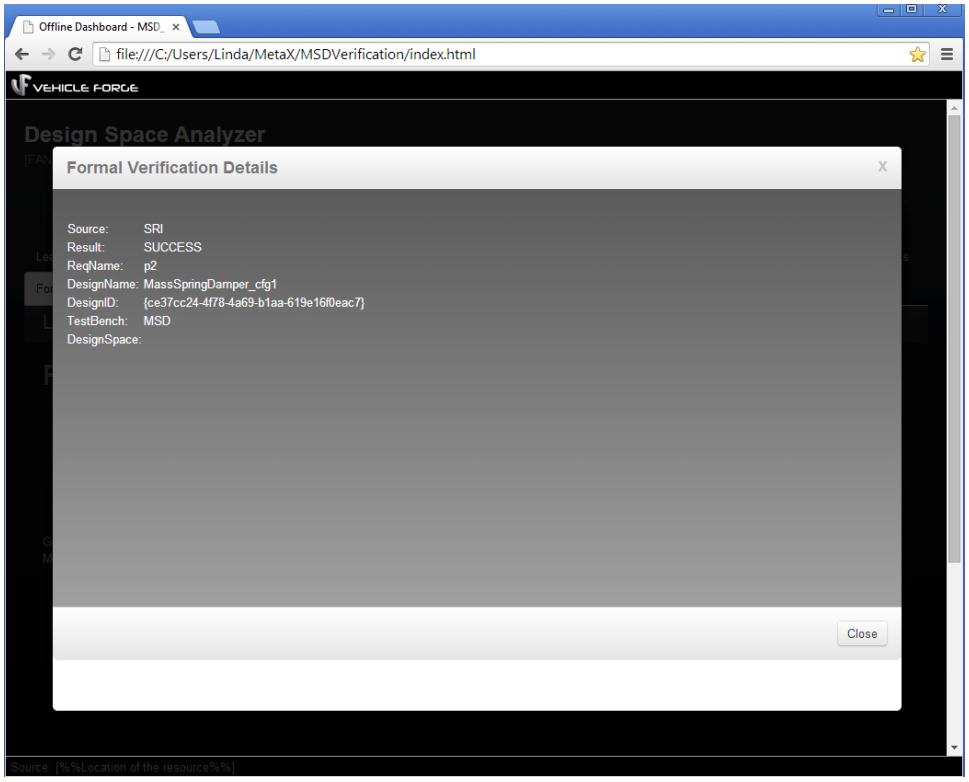

Fig. 6. Details of the verification of Property `p2`.

this property. However, it is possible to fix the controller logic to make property `p1` true. Failure of `p1` was possibly a bug in the designed controller.

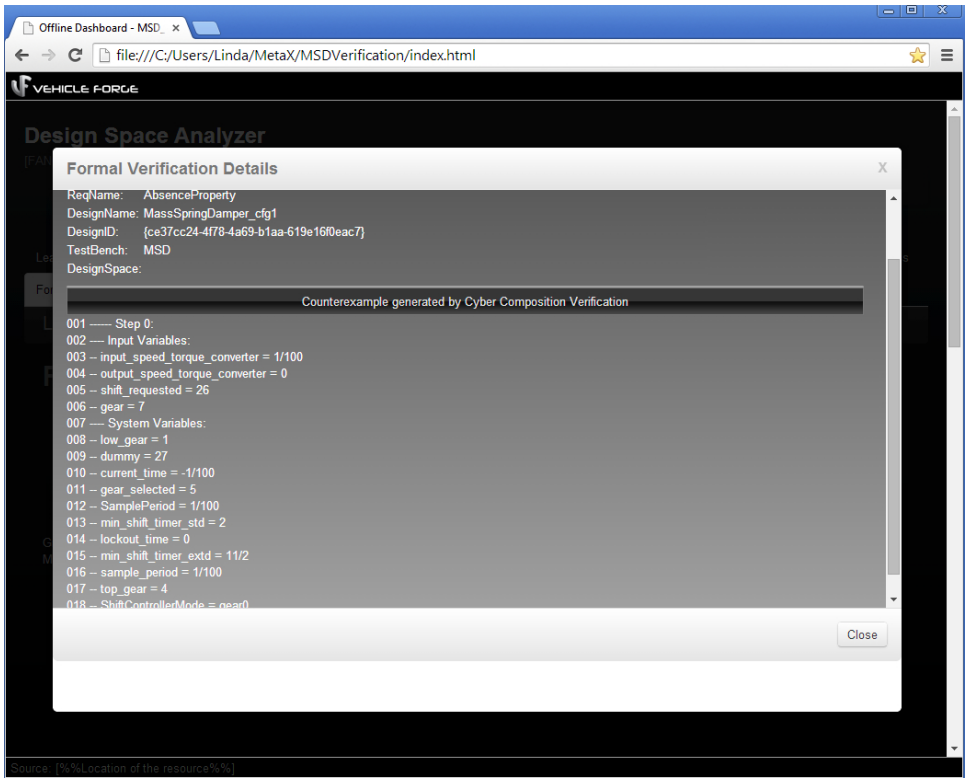

Fig. 7. Details of the verification of Property `AbsenceProperty`.