Testing TargetLink® Models and C Code with Reactis®

Build better embedded software faster. Generate tests from TargetLink models. Detect runtime errors. Execute and debug models. Track coverage. Back-to-back testing of generated code against model.

RSITR 3.6
February 15, 2015

www.reactive-systems.com
1 Introduction

2 Working with TargetLink Models in Reactis
   2.1 Reactis-Friendly TargetLink Settings ........................................ 3
      2.1.1 Activating MIL Mode ...................................................... 3
      2.1.2 Disabling Block Logging .................................................. 4
      2.1.3 Making Ports Virtual (TargetLink 2) .................................... 4
      2.1.4 TargetLink Full-Featured and Stand-alone Installations .......... 4
   2.2 TargetLink-Friendly Reactis Settings ......................................... 5
      2.2.1 Selecting the MATLAB Version in Reactis ............................. 5
      2.2.2 Configuring the Reactis Path ............................................ 6
      2.2.3 Enable Propagate set_param changes ................................ 6
   2.3 Configuring Inport Types ....................................................... 7
   2.4 Pitfalls to Avoid ........................................................................ 7
      2.4.1 Nonstandard Switching Between TargetLink Versions ............ 7
      2.4.2 Incidental TargetLink Dialogs ............................................. 7

3 Working with TargetLink-Generated Code in Reactis for C .................... 8
   3.1 Generating the TargetLink C Code ............................................ 8
   3.2 Comparing Behavior of Generated Code to Model ....................... 8
   3.3 Creating Reactis for C Build files for TargetLink-Generated C Code .... 9
   3.4 Supporting TargetLink fixed-point code generation in Reactis for C ...... 9
   3.5 Reactis support for TargetLink Custom Code Feature ................. 10

4 Conclusions ............................................................................. 10
About Reactive Systems, Inc.

Reactive Systems, founded in 1999, is a privately held company based in Cary, NC. The company’s Reactis product line provides automated testing and validation tools to support the development of embedded control software. Reactis, Reactis for C Plugin, Reactis for EML Plugin, Reactis Model Inspector, and Reactis for C support model-based design with Simulink, Stateflow, Embedded MATLAB, and C code. Reactis Tester automatically generates comprehensive yet compact test suites from a Simulink model or C code. Reactis is used at companies worldwide in the automotive, aerospace, and heavy-equipment industries.

Reactive Systems, Inc.
341 Kilmanyne Dr.
Suite 101
Cary, NC 27511
USA

Tel.: +1 919-324-3507
Fax: +1 919-324-3508

Web: www.reactive-systems.com
E-mail: info@reactive-systems.com
Abstract

The automated testing and validation capability of the Reactis® tool suite significantly enhances the gains realized from a model-based design process. Reactis can successfully test, simulate, and debug models created using the dSPACE TargetLink® blockset. Combined with Reactis for C, Reactis also provides a robust test-generation and debugging capability for TargetLink-generated C code. This document describes the steps necessary to configure Reactis and TargetLink models for test generation, simulation, and debug. The paper also outlines the execution and validation of automatically-generated C code.

Reactis is a registered trademark of Reactive Systems, Inc.
TargetLink is a registered trademark of dSPACE, GmbH

1. Introduction

Reactis provides model-based testing and validation of discrete-time embedded-controller models in the Simulink® / Stateflow® notation developed by MathWorks. The dSPACE autocode tool TargetLink has become popular because:

- its Simulink-compatible blockset enables detailed specification of the C code attributes, and
- the C code generated from TargetLink models is very high quality.

With proper configuration, Reactis can work with not only the TargetLink blockset (of many versions of TargetLink), but also with the C code it generates. This has many obvious advantages:

- Direct comparison between idealized-model behavior and realized C-code behavior.
- Reactis test vectors derived for model-coverage can be used to analyze code coverage in the same environment. This can be extremely useful when looking at boundary or threshold conditions.
- Interactive analysis using the full step-accurate debugging capability of Reactis.

A model-based design environment involving Reactis and TargetLink is depicted in Figure 1. Reactis contains three core components: Tester, which offers automatic test generation from models; Simulator, which enables users to visualize model execution to debug models and track coverage; and Validator, which offers automated checks of models for violations of user-specified requirements. For more information on the basic capabilities of Reactis, please see [RSI10].
2. Working with TargetLink Models in Reactis

As shown in Figure 2, some preparation steps are necessary before working with a TargetLink model in Reactis. After performing these steps, users can manipulate TargetLink models in Reactis in the same way native Simulink models are processed. Namely it is easy to:

- Generate tests from a model with Reactis Tester.
- Simulate and debug models in Reactis Simulator.
- Check that a model meets its requirements with Reactis Validator.

Reactis is compatible with a number of different TargetLink versions including 2.1.6, 2.2.1, 2.3.0, 3.0, 3.1, 3.2, 3.3, 3.4, 3.5, 4.0, 4.1, and 4.2. Most of the model-preparation steps are similar for the different versions. When differences exist, they are described below. The biggest difference is that when using version 3.0 or later of the TargetLink blockset, it is not necessary to switch to the stand-alone blockset as is required to work with previous versions of the TargetLink blocks in Reactis.
2.1. Reactis-Friendly TargetLink Settings

A few TargetLink settings may need adjustment before working with a model in Reactis. We describe those settings in this section.

2.1.1 Activating MIL Mode

In order to work properly with Reactis, TargetLink models should be set to Model-in-the-Loop (MIL) mode. If the simulation mode is set to Software-in-the-Loop (SIL), or Program-in-the-Loop (PIL), it may not function properly in Reactis. The following steps will set your model...
to use MIL mode:

1. Double-click on the TargetLink block at the top of your model to open the TargetLink Main Dialog.

2. In the Code Generation tab, click the button labeled Activate MiL.

2.1.2 Disabling Block Logging

TargetLink provides a useful mechanism for logging and graphing the output signals of blocks selected by the user. During or shortly after a simulation (depending on the version of Targetlink) a window with graphs will appear that displays the selected signals. This feature is not compatible with Reactis, so the user should disable the feature as follows:

1. Double-click on the TargetLink block at the top of your model to open the TargetLink Main Dialog.

2. In the Code Generation tab, select Do not log anything as the Global logging option.

Reactis provides alternative ways to view signal graphs when running a model in Reactis Simulator.

2.1.3 Making Ports Virtual (TargetLink 2)

Older versions of TargetLink include special inports and outports which are used to augment its code generation capability. (These ports are not present in version 3.0 or later.) The ports frequently serve to define various interface boundaries. Most of the functionality is interpreted only by the TargetLink code generator, but in certain situations, the blocks can affect simulation behavior. Certain settings will cause the dynamic insertion of Data Type Conversion blocks that are typically invisible to the user (as shown in Figure 3), but cause problems for Reactis.

To prevent this problem, one should configure the block as a pass-through. This is done as follows:

1. Load the model in Simulink

2. Double-click on the TargetLink Inport (or Outport)

3. Select the Logging & Autoscaling tab

4. Under Simulation Behavior, check the box Virtual Port.

2.1.4 TargetLink Full-Featured and Stand-alone Installations

dSPACE offers two ways to install TargetLink on your computer: full-featured and stand-alone.

The full-featured version is intended to be used in the main TargetLink environment when the objective is code generation or Software-in-the-Loop (SIL) simulation.
The stand-alone installation enables users to simulate TargetLink models in the Simulink environment, but not generate code or access many of the advanced TargetLink features. Use of the stand-alone installation does not require a TargetLink license.

When using a full installation, it is possible to switch a model back and forth between the full and stand-alone versions of the TargetLink blocks. Prior to TargetLink Version 3.0, it was necessary to switch to the stand-alone blockset to work with a model in Reactis. As of TargetLink 3.0, Reactis is compatible with both the full-featured and the stand-alone blockset.

When using the full blockset, prepare a model for use with Reactis as follows:

1. Start MATLAB (the version paired with TargetLink)
2. At the MATLAB command prompt enter: `tl_switch_blockset`
3. Load the model in Simulink
4. Save the model

To switch back to the full-featured blockset, just enter the `tl_switch_blockset` command a second time and save the model. To determine which blockset is currently active, at the MATLAB command prompt enter: `tl_get_blockset_mode`. The blockset setting is persistent over different MATLAB sessions, so this step need not be performed each time a model is opened in Reactis.

2.2. TargetLink-Friendly Reactis Settings

2.2.1 Selecting the MATLAB Version in Reactis

Reactis can be configured to use a specific version of MATLAB/Simulink as follows:

1. Start Reactis.
2. Select File → Global Settings...
3. In the resulting Global Settings dialog, select the MATLAB tab and then select the appropriate MATLAB version using the pulldown menu.
4. Click OK to dismiss the Global Settings dialog.
2.2.2 Configuring the Reactis Path

Like MATLAB, Reactis maintains a path variable consisting of a list of folders to be searched for various files used by a model. These files include libraries, including the TargetLink libraries. Unlike MATLAB, Reactis maintains both a global and a model-specific path. The model-specific path is prepended to the global path to construct the full search path used with a model.

When using Reactis with TargetLink, we recommend updating the model-specific path to include the folders containing the TargetLink libraries. The following steps set the model-specific path:

1. Make sure the model loads and simulates properly in the Simulink environment.
2. Start Reactis and load the model.
3. Select Edit → Search Path...
4. Click the Import button. This will cause Reactis to:
   (a) load the model in Simulink (executing any initialization files that might update the path),
   (b) query Simulink for the resulting path for the model,
   (c) add the returned MATLAB path to the model-specific Reactis path for the model.
5. Select File → Save to save the updated path into the .rsi file maintained by Reactis for the model.

The model-specific path should now include the dSPACE directories required to simulate a TargetLink model.

2.2.3 Enable Propagate set_param changes

Some TargetLink blocks have attributes that are dynamically updated after the model is loaded. Models using these self-modification methods (e.g. ‘set_param’, ‘add_block’, etc.) could previously cause problems because Reactis would not see the changes made by them. To avoid these issues, Reactis now offers a setting that should be turned on when working with TargetLink models.

The setting is named Propagate set_param changes by saving the model to a temporary file and can be enabled as follows:

1. Load your model in Reactis.
2. Select Edit → General...
3. In the resulting dialog, ensure the box labeled Propagate set_param changes by saving the model to a temporary file is checked.

When this setting is enabled, Reactis will:
1. Invoke Simulink/TargetLink to apply the changes to the model,
2. Automatically save changed model to a temporary file,
3. Import the temporary model file, allowing Reactis to see the applied changes.

All of this happens invisibly without any need for interaction from the user.

2.3. Configuring Inport Types

To avoid overflows and improve test coverage, Reactis should be configured to provide inputs within the range expected by the generated code. These ranges can be automatically defined as follows:

1. Make sure the model loads and simulates properly in the Simulink environment.
2. Start Reactis and load the model.
3. Select **Edit → Inport Types...**
4. A dialog may appear asking for permission to retrieve the inport types from the model.
   If this dialog appears, click on the **OK** button. Retrieving the types may take a minute or two.
5. Select **Tools → Synchronize inport ranges with TargetLink data dictionary**
6. The inport types will have range and resolution constraints based on the type that the inport will have in the code generated by TargetLink. These constraints can be adjusted by editing the inport type if desired.

2.4. Pitfalls to Avoid

2.4.1 Nonstandard Switching Between TargetLink Versions

In earlier releases, each TargetLink version mapped to one distinct MATLAB® ¹ installation. To switch mapping, the dSPACE Installation Manager required a system reboot to complete the activation. Some users created backdoor scripts to more quickly switch between installations of TargetLink. Unfortunately, these scripts are not compatible with Reactis, so the user should use the normal TargetLink activation procedure through dSPACE Installation Manager. In later versions of the dSPACE Installation Manager, switching between TargetLink versions is unnecessary in most circumstances, and otherwise very quick.

2.4.2 Incidental TargetLink Dialogs

Because Reactis reads and interprets TargetLink models, occasionally a TargetLink dialog will appear that requires user interaction. Typically, the user will only need to close the dialog or click OK. But it is important the user remain attentive to these dialogs as they may suspend further processing of the model.

¹MATLAB is a registered trademark of MathWorks.
3. Working with TargetLink-Generated Code in Reactis for C

dSPACE TargetLink’s main purpose is generating native C code from models. Reactis can analyze those models to create comprehensive test suites. Reactis for C (a separate product) can take the test suite generated for the model and run it directly on the final C code, as part of a comprehensive back-to-back testing process.

3.1. Generating the TargetLink C Code

To generate suitable C code for testing using Reactis for C, do the following:

1. Make sure the TargetLink model runs successfully in Model-In-the-Loop (MIL) mode in TargetLink.

2. Make sure the TargetLink model runs successfully in Software-In-the-Loop (SIL) mode. This confirms that the code can be generated from the model, and that any supporting C files have been located, and that a linkable construct has been built.

3. Select the option “Do not log anything” in the Simulation Frame Options window of the TargetLink main dialog.

4. Select the option “Clean Code” in the Code and Logging Section of the TargetLink main dialog.

3.2. Comparing Behavior of Generated Code to Model

Since Reactis generates a test suite from the model that is the same format as used by Reactis for C, it is very easy to compare the C code behavior against the original model from which it was generated. In describing the comparison method, assume the following:

- A.mdl is a TargetLink model
- Reactis Tester is used to generate test suite TS.rst from A.mdl
The following steps let you run the test suite TS.rst (generated from A.mdl) on the generated C code B using Reactis for C. Any output differences will be flagged.

1. Create a Reactis Build file (.rsm file) that lists the C source files used by B. This process is described in Section 3.3.

2. Create a new harness, stored in a harness library (.rsh file), that specifies the entry function of the C code along with the inputs and outputs of the unit under test. For more information, please consult the Reactis for C documentation.


4. Load and run TS.rst. Any differences in behavior will be flagged. The executing C code can be easily examined in the Simulator environment.

### 3.3. Creating Reactis for C Build files for TargetLink-Generated C Code

Reactis for C offers white-box testing of the C code generated from the TargetLink model. This section describes how to create a Reactis Build file (RSM file) for TargetLink-generated C code. Doing so enables the white-box analysis of TargetLink-generated code in Reactis for C.

An RSM file includes the following information:

- The C source files generated from the model
- The location of RSM files for any additional libraries referenced from the C code.
- The search path for finding header files during preprocessing
- Any macro definitions required to compile the C code

The RSM file for code generated from one of the TargetLink fuelsys demo models is shown in Figure 5.

No libraries are used by this model. If you require libraries you would need to define RSM files for those libraries and list them in the "Libraries" tab of the RSM file.

Some other information to be aware of when creating RSM files includes the following:

- Depending on the version of TargetLink used, the generated code is usually present in the “TLProj” directory.

### 3.4. Supporting TargetLink fixed-point code generation in Reactis for C

Reactis for C also supports TargetLink code generation for fixed-point applications. Reactis uses the native C files that implement the fixed point math functions, and builds them into a separate library, as described below.

Note: To make the compilation of the dSPACE fixed-point function files efficient, perform the following: In Reactis under Edit → General, make sure the 'Create and use cache files for C code (files with extension MWI)' option is checked.
The dsfxp.rsm library is built from C files located in directory:
C:\dSPACE\TL300R2008a\MATLAB\tl\SrcFiles\Generic\DSFxp

In your rsm file dialog is a tab titled 'Libraries'; select it. Click the 'Add' button in that section to create your library rsm file. For simplicity, you may want to create the dsfxp.rsm in the same directory as the source C files, so navigate to that directory.

Select ALL the C files from this directory, then add them to the “Source Files” section. Add the appropriate paths to the “Include Search Path” section and create the entries in the “Defines” section as seen in Figure 6. Because the source files named in dsfxp are part of a standard library, there is no need to track code coverage within it. To do so, change the “Coverage Tracking” selection in the General tab of the rsm dialog to 'Off'.

3.5. Reactis support for TargetLink Custom Code Feature

TargetLink supports the insertion of user-supplied C code into the Simulink simulation environment. This may be useful if:

- Blocks that are not supported by TargetLink have to be implemented.
- Well-tested custom functions or algorithms need to be called from within the Simulink environment.

With the TargetLink “Custom Code Block”, you can insert parts of your own code directly, and have these sections ultimately appear in the TargetLink-generated code. This block is one of the TargetLink simulation blocks found in the “tlib” library.

Because these blocks implement S-Function-wrapped C code, they can be executed in Reactis, and even debugged using the Reactis for C Plugin. An example block from the model, “Custom Blocks” (provided in the TargetLink demo directory ) is shown in Figure 7.

The file listed in the File name dialog, “table_code.c” is not actually a C file but a template file that TargetLink uses to provide a location for user insertion of both fixed- and floating-point code snippets. In the dialog of the “table_code.c” block, note the “Use production code for floating-point simulation” checkbox. This box should be checked if the user desires to build and execute fixed-point code. Otherwise it should be un-checked.

Because of the many ways the custom code block can be configured, it is the responsibility of the user to know which defines should be added, and which source files should be included. Please see the TargetLink documentation for more information on editing, compiling, and building the C code associated with this feature. For this example, an rsm file in shown in Figure 8.

4. Conclusions

In this paper we have discussed how to use Reactis to test and validate TargetLink models. We have also covered the steps necessary to bring TargetLink-generated C code into the Reactis for C environment for detailed testing, validation and analysis of the C code.

TargetLink is available now from dSPACE GmbH. Please see www.dspaceinc.com for details.
Reactis and Reactis for C are available now from Reactive Systems, Inc. Please see the Company’s web site at www.reactive-systems.com for ordering information and for instructions on how to download a free 30-day evaluation copy of the software.

References

Figure 5: RSM file tabs for C code generated in TargetLink 4.0.
Figure 7: Example model using the “Custom Code” block.
Figure 8: RSM file tabs for custom code library