Signature Tool

September 2019

McMaster Centre for Software Certification (McSCert)

1 Introduction

A Simulink subsystem has Inports (explicit links to the subsystem), and Outports (explicit links from the subsystem), which are the explicit interface of the subsystem. However, there are usually hidden (implicit) data dependencies across Simulink subsystems. Hidden dependencies originate from two Simulink mechanisms: 1) Data Store Memory; and 2) Goto/From blocks with a Goto Tag Visibility[1].

The Signature tool extracts the *signature* of a Simulink subsystem. A signature represents the interface of a Simulink subsystem, making all data flow into and out of the subsystem explicit. The tool identifies two types of signatures for a subsystem: *strong* signature and *weak* signature. The strong signature identifies the data mechanisms that *are accessed* by the subsystem or any of its children. The weak signature identifies the data mechanisms that a subsystem *can access* but is not necessarily using (i.e., those which are declared higher up in the hierarchy, and are thus available at lower levels).

The Signature tool can be used to either include the signature in the model itself, or export the signature into a text (`.txt`), TEX (`.tex`), or Microsoft Word (`.doc`) file. The tool can also be used to improve test harnessing with commercial automatic test generation tools.

More Information

For more about the theoretical background of signatures and how they can be used, an interested reader is referred to:

Marc Bender, Karen Laurin, Mark Lawford, Jeff Ong, Steven Postma, Vera Pantelic, "Signature Required - Making Simulink Dataflow and Interfaces Explicit," *Proceedings of 2nd International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2014)*, SCITEPRESS, 2014, 119-131. (Nominated for Best Paper Award)

For more information on the tool and how it can be used in a model-based development with Simulink, please refer to the following papers:

Vera Pantelic, Steven Postma, Mark Lawford, Monika Jaskolka, Bennett Mackenzie, Alexandre Korobkine, Marc Bender, Jeff Ong, Gordon Marks, Alan Wassyng, "Software engineering practices and Simulink: bridging the gap," *International Journal on Software Tools for Technology Transfer (STTT)*, 2017, 1-23.

Vera Pantelic, Steven Postma, Mark Lawford, Alexandre Korobkine, Bennett Mackenzie, Jeff Ong, and Marc Bender, "A Toolset for Simulink: Improving Software Engineering Practices in Development with Simulink," *Proceedings of 3rd International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2015)*, SCITEPRESS, 2015, 50–61.

[1] Called *scoped* Goto/From blocks

2 How to Use the Tool

This section describes what must be done to setup the tool, as well as how to use the tool.

2.1 Prerequisites and Installation

1. Use MATLAB/Simulink 2011b or newer.
2. To install the tool,
   (a) from a `.zip` file — unzip the contents into your desired location. Ensure the unzipped folder and subfolders are present in your MATLAB search path, or add them if they are not present. Run sl_refresh_customizations to refresh the Context Menu.
   (b) from a `.mltbx` file — simply open MATLAB and double-click on the file. Your MATLAB search path should be automatically configured.
   (c) from the files only — add the folders and subfolders to your MATLAB search path. Run sl_refresh_customizations to refresh the Context Menu.
   - *Note:* If running the command "`which StrongSignature`" indicates that the script is not found, then the tool needs to be added to the MATLAB search path. For information on adding files to the MATLAB search path, please see the MathWorks documentation.
3. Ensure your model is open (or loaded, for command line use) and unlocked.

2.2 Getting Started

The tool can be used via the Simulink Context Menu, which can be viewed by right-clicking in a model. The following options are available in the Signature menu, as shown in Figure 1:

- *Extract Signature*
- *Augment for Test Harness*

2.3 Functionality

This section describes the tool functionality when it is used from the Simulink Context Menu (Figure 1).

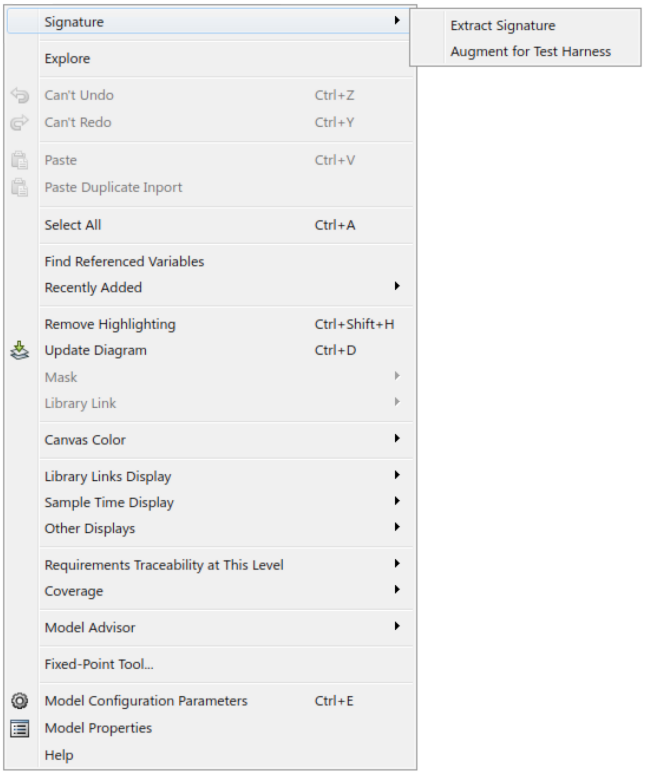

Figure 1: Simulink Context Menu with tool options visible.

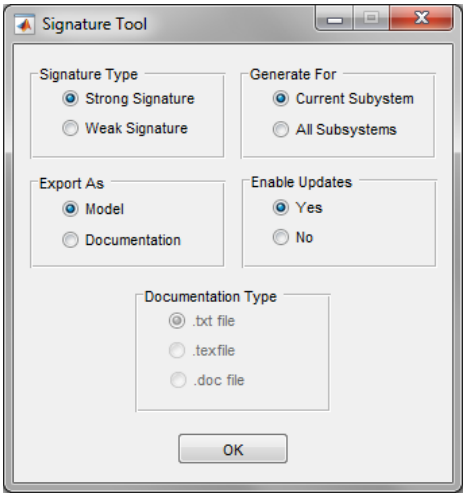

Figure 2: Signature extraction interface.

Extract Signature

Right-clicking anywhere in the model and then selecting `Extract Signature` from the Context Menu will display the user interface shown in Figure 2.

The user has the following options when it comes to generating a signature:

- Signature Type — Choose whether to extract the strong signature or weak signature.
- Generator For — Choose whether to get the signature for the current subsystem (i.e., one level of the hierarchy) or all subsystems.
- Export As — Choose to represent the signature in the model, or as textual documentation.
- Enable Updates[2] — Choose whether updates are to be included in the signature, or included separately as Reads and Writes. *Note:* Do not enable when using with `Augment for Test Harness`.
- Documentation Type — If exporting as documentation, choose to output as text files, TEX files, or Microsoft Word files.

For signatures generated in the model, a copy of the original model will be created, where the signature will be generated. This new model will have the same filename as the original, with `_StrongSig` or `_WeakSig` appended. This is done to ensure no changes are made to the original model.

Augment for Test Harness

Prerequisite: The strong signature, without updates, must be generated in the model. Otherwise, this operation will have no effect on the model.

Right-clicking anywhere in the model and then selecting `Augment for Test Harness` from the Context Menu, will augment a strong signature for use with test harnessing. This operation is most useful when the subsystem of interest is extracted from the model, with implicit data flow to/from higher levels of the model. Some testing tools, such as Reactis by Reactive Systems, support subsystem extraction in order to test a single subsystem of the model. In this case, implicit data flow information higher in the model may be lost. Moreover, testing tools commonly do not account for implicit data flow into the subsystem, thereby not generating input data, and ultimately not exercising the model fully. In this situation, depending on the tool and signal type, the value of Froms/Data Store Reads will constantly default to 0 for all tests.

To allow test inputs to be fed into Froms/Data Store Reads, or test output data to be recorded from Gotos/Data Store Writes, Inport and Outport blocks are added to the strong signature. As a result of these additional Inports/Outports,

[2]Updates refers to Data Store Memory blocks and scoped Goto/From blocks that are both written to and read from in a subsystem.

testing coverage can increase, while testing efforts (e.g., number of test cases) decrease.

Note: This operation will have no effect at the root system level, or if the subsystem does not make use of implicit data higher in the hierarchy.

2.4 Configuration Parameters

The configuration file `config.txt` is included in `Signature\src`. The following configuration parameters are utilized by the tool, and can be modified by the user in order to tailor tool functionality:

- `gotofrom_bgcolor` – The color of the Goto/From blocks in the signature, and their corresponding Goto/From blocks connecting the signature to the original model.
- `heading_size` – The font size for the signature section heading annotations (i.e., "Inputs", "Outputs", "Test Harness").

Please see the configuration file for more details regarding parameter usage and accepted values. These parameters can be modified with MATLAB open, and do not require that MATLAB be restarted for the changes to take effect.

2.5 Errors and Warnings

Any errors or warnings during tool use will be visible in the MATLAB Command Window. Typically, errors will be shown when the model is locked or function parameters are incorrect.

3 Example

Use the command `SignatureDemo` in the Simulink command window to open the example model, shown in Figure 3. This example has implicit data flow passing to other levels of the model via Data Stores, as well as a scoped Goto/From blocks. In particular, we will focus on Subsystem1, which is shown in Figure 4. The following steps will demonstrate the various functions of the tool.

1. **Generate Strong Signature** Let us extract the strong signature of Subsystem1. To do this, right-click anywhere in the model and select the `Extract Signature` option. The Signature GUI will pop-up (shown in Figure 2). Change the `Export As` option to `Model`. The resulting model is shown in Figure 5 and is named `SignatureDemo_StrongSig.mdl`.

   Recall that the strong signature shows all the data that is actually used in the system. Straightforwardly, the Inports and Outports of the subsystem are part of the signature, as these are the typical inputs and outputs of the subsystem. Likewise, the subsystem's use of the Data Store Read block dsB, scoped From block scopedZ, and Data Store Write block dsA

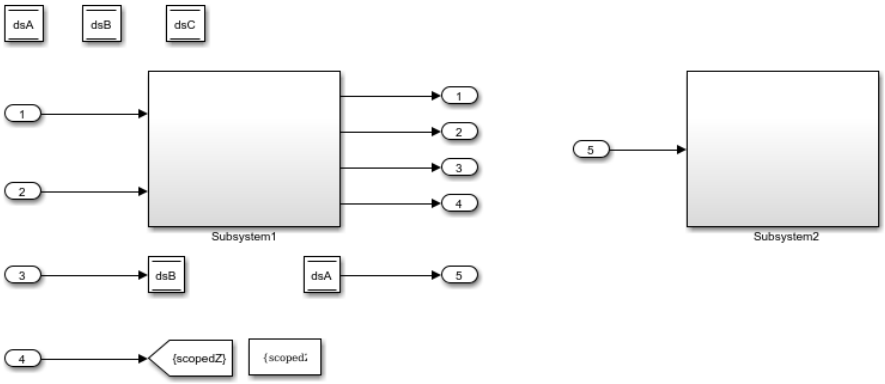

Figure 3: Signature demo: Root level of `SignatureDemo`.

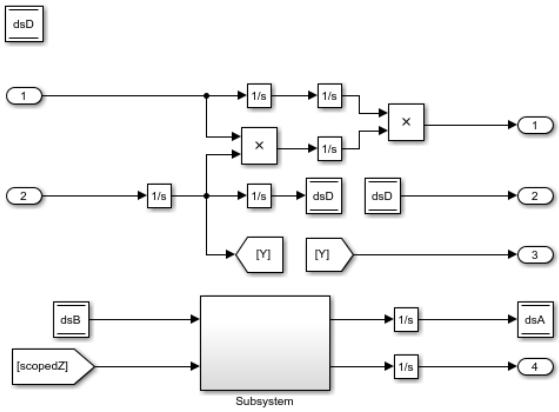

Figure 4: Signature demo: Subsystem1 from Figure 3.

are reflected in the signature. Lastly, the Data Store Memory block dsD is included on the signature, as it is read/written to inside Subsystem.

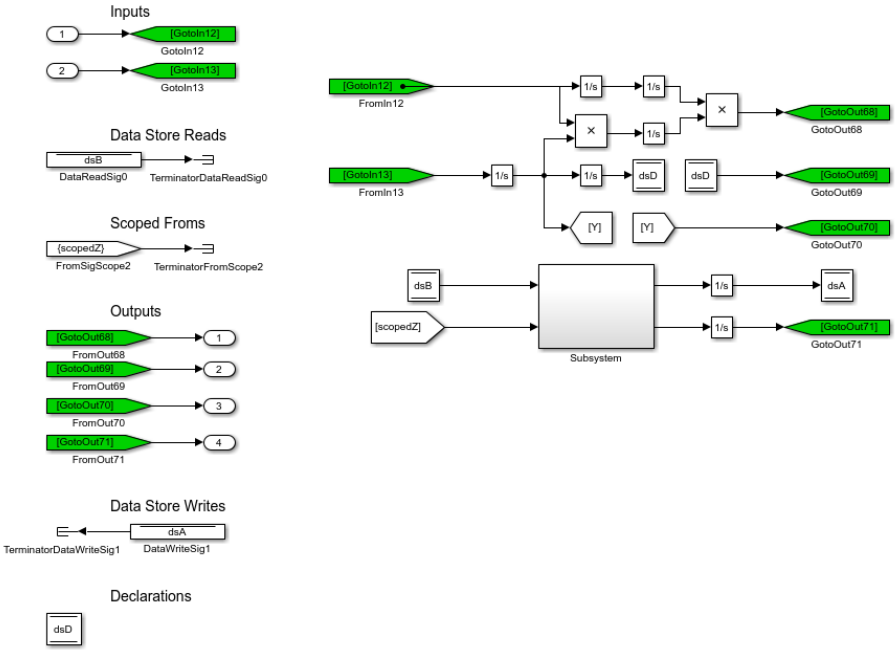

Figure 5: Strong signature (shown in the model) for Figure 4.

2. **Generate Weak Signature** Let us now extract the weak signature of Subsystem1. To do this, right-click anywhere in the original model and select the `Extract Signature` option. The Signature GUI will pop-up (shown in Figure 2). Change the `Signature Type` to `Weak Signature` and change the `Export As` option to `Model`. For demonstration purposes, also select `No` for the `Enable Updates` option (this will be explained in Step 3). The resulting model is shown in Figure 6 and is named `SignatureDemo_WeakSig.mdl`.

   In contrast to the strong signature, the weak signature shows all the data that the system *can access*, regardless of if it is actually used. All the information that was in the strong signature is also in the weak signature. For this reason, we again see the Inports and Outports of the subsystem, as well as the Data Store Read block dsB, scoped From block scopedZ, Data Store Write block dsA, and Data Store Memory block dsD are present, as they are all read/written to in Subsystem1.

   Due to the presence of Data Store Memory blocks dsA and dsC at the root level, this subsystem has the ability to read or write to this memory

(although it is currently not doing so). For this reason, dsA and dsC are included in the weak signature under the Data Store Read and Data Store Write headings. Moreover, although Subsystem1 is already reading from Data Store Memory dsB, it has the potential to also write to it. Therefore, dsB is now also now found under the Data Store Writes heading.

If at any point in development the use of implicit data items found in the weak signature (and not in the strong signature) is needed, a developer can simply replace the terminator block connected to the element with a Goto or From block (depending on the scenario), and easily use the element in their subsystem. The benefit of the weak signature is that it notifies the developer of data that is already available to them.

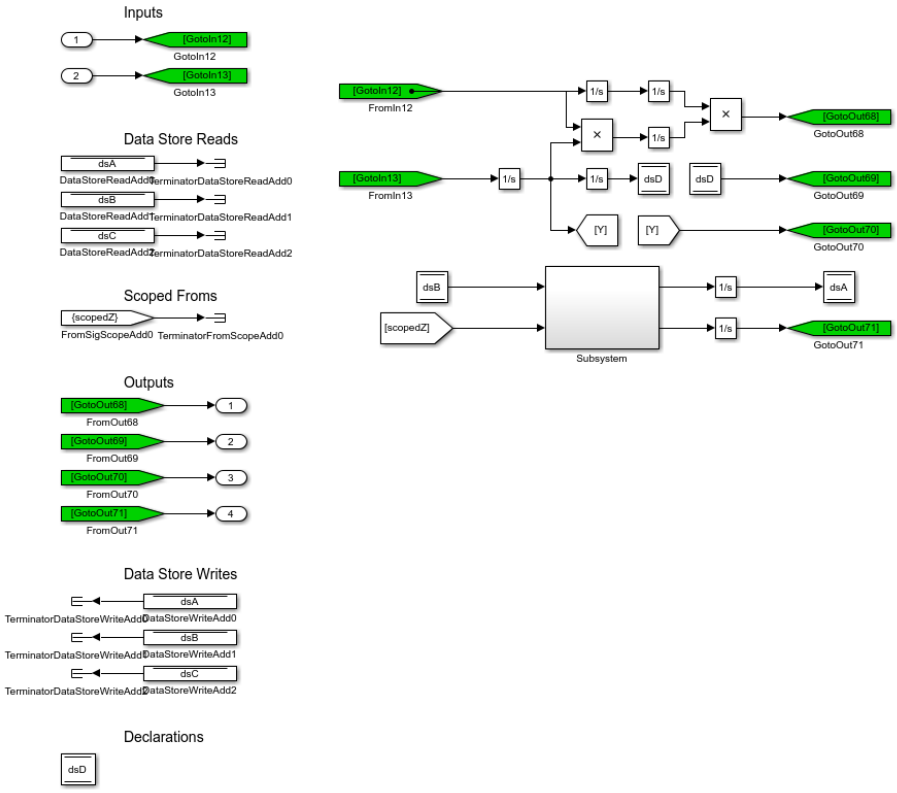

Figure 6: Weak signature (shown in the model) for Figure 4.

3. **Generate Signature with Updates** Let us re-generate the weak signature as done in Step 3, but this time with the `Enable Updates` option set to `Yes`. The difference in the signature shown in Figure 7. In general, for Data Store Memory and scoped Goto/From blocks that are both read from and written to in the subsystem, will appear together as "Updates" when enabled. In Figure 7, Data Store dsA, dsB, and dsC in the right signature are now grouped together under the Updates heading in the right signature. Updates can be enabled or disabled for both the weak and strong signatures.

4. **Augmenting for Test Harness** To augment a signature for use with a test harness, first the strong signature must be generated in the model with updates disabled. Once this is done, navigate to the system which requires testing. Right-click in the model and select the `Augment for Test Harness` option. The resulting subsystem is shown in Figure 8. The strong signature now contains a new "Inputs for Harness" section which

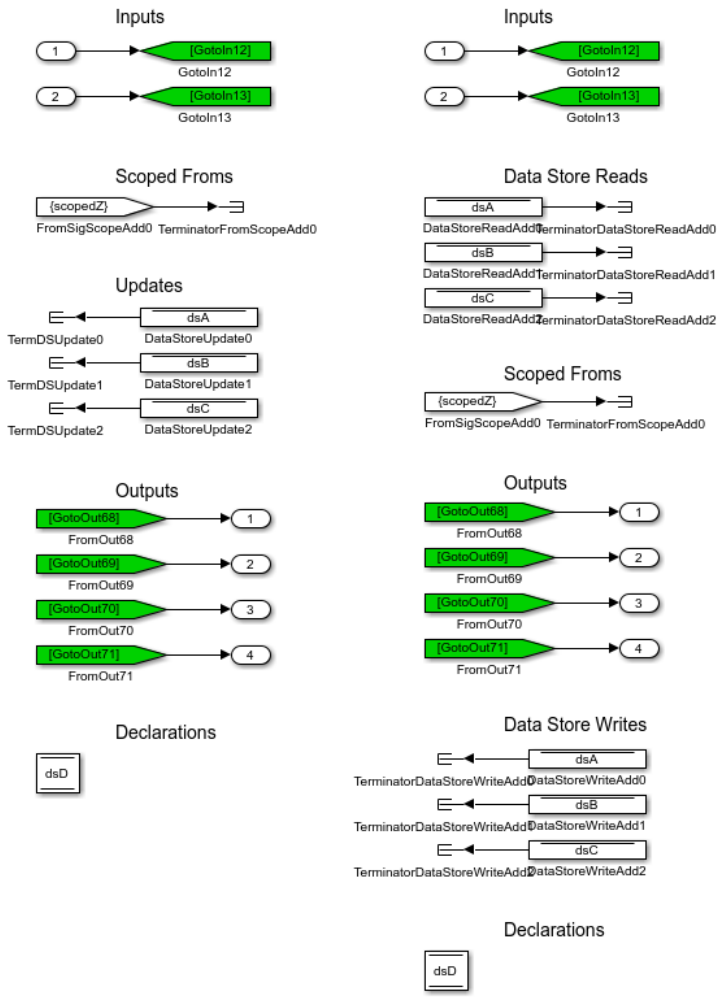

Figure 7: Difference in signatures between updates enables/disabled.

provides Inports for dsB and scopeZ so that testing will generate inputs for the implicit data, thereby exercising the subsystem more thoroughly during testing. Also included is an Outport for dsA, so that its data can be recorded during testing.

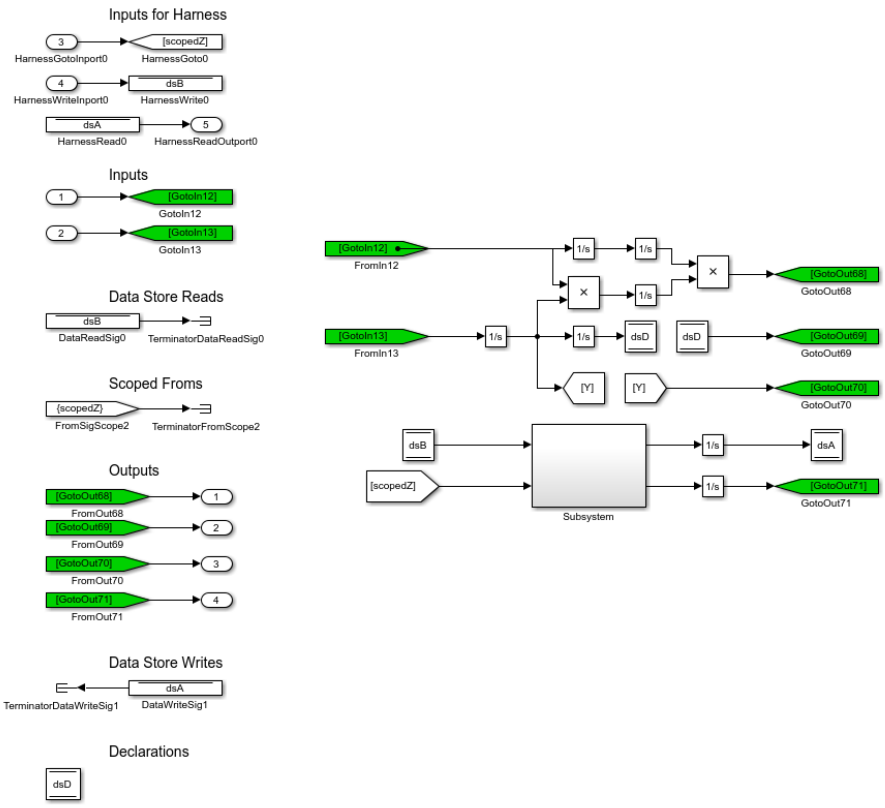

Figure 8: Subsystem1 of Figure 4 augmented for testing.

4 Matlab Commands

The tool can also be used via the MATLAB command line, with the following functions.

Function	`StrongSignature`
Syntax	`[metrics, signatures] = StrongSignature`(*address, exportType, hasUpdates, system, docFormat*)
Description	Generate the strong signature for *system*.
Inputs	*address*: Simulink model name. *exportType*: Number indicating whether to export as a model (0) or as documentation (1). *hasUpdates*: Number indicating whether reads and writes in the same subsystem are kept separate (0), or combined and listed as an update (1). *system*: Name of the system to generate the documentation for. It can be a specific subsystem name, or 'All' to document the entire hierarchy. *docFormat*: Number indicating which docmentation type to generate: `no doc` (0), `.txt` (1), `.tex` (2), `.doc` (3), else no doc. This parameter does not have an impact when *exportType* is 0.
Outputs	*metrics*: Cell array listing the systems and the sizes of their signatures (i.e. number of elements in the signature). *signatures*: Cell array of signature data for the system and its subsystems. Signature data includes: Subsystem, Size, Inports, Outports, ScopedFromTags, ScopedGotoTags, DataStoreReads, DataStoreWrites, Updates, GotoTagVisibilities, and DataStoreMemories.

Function	`WeakSignature`
Syntax	`[metrics, signatures] = WeakSignature`(*address, exportType, hasUpdates, system, docFormat*)
Description	Generate the weak signature for *system*.
Inputs	*address*: Simulink model name. *exportType*: Number indicating whether to export as a model (0) or as documentation (1). *hasUpdates*: Number indicating whether reads and writes in the same subsystem are kept separate (0), or combined and listed as an update (1). *system*: Name of the system to generate the documentation for. It can be a specific subsystem name, or 'All' to document the entire hierarchy. *docFormat*: Number indicating which docmentation type to generate: `no doc` (0), `.txt` (1), `.tex` (2), `.doc` (3), else no doc. This parameter does not have an impact when *exportType* is 0.
Outputs	*metrics*: Cell array listing subsystems and the sizes of their signatures (i.e., number of elements in the signature). *signatures*: Cell array of signature data for each subsystem. Signature data includes: Subsystem, Size, Inports, Outports, GlobalFroms, GlobalGotos, ScopedFromTags, ScopedGotoTags, DataStoreReads, DataStoreWrites, Updates, GotoTagVisibilities, and DataStoreMemories.

Function	`TestHarness`
Syntax	`TestHarness`(*system*)
Description	Augment *system* with a test harness.
Inputs	*system*: Simulink system path to generate the harness for.
Outputs	N/A