Documenting Simulink Designs of Embedded Systems

Alexander Schaap, Gordon Marks, Vera Pantelic, Mark Lawford, Gehan Selim, Alan Wassyng, Lucian Patcas
McMaster Centre for Software Certification, McMaster University
Hamilton, ON, Canada
{schaapal,marksgw,pantelv,lawford,selimg,wassyng,patcaslm}@mcmaster.ca

ABSTRACT

The importance of appropriate software design documentation has been well-established. Yet in industrial practice design documentation of large software systems is often out of date or entirely lacking in large part due to the effort required to produce and maintain useful design documents. While model-based design (MBD) partially addresses this problem, large complex models still require additional design documentation to enable development and maintenance. This paper introduces tool support for documenting the Software Design Description (SDD) of embedded systems developed using MBD with Simulink. In particular, the paper proposes a template for a SDD of a Simulink model. Then, the tool support we have developed for semi-automatic generation of SDDs from the template is introduced. The tool support integrates MathWorks' Simulink Report Generator and our previously developed Signature Tool that identifies the interfaces of Simulink subsystems.

CCS CONCEPTS

• **Software and its engineering → Maintaining software**; *Model-driven software engineering*; Software maintenance tools;

KEYWORDS

Software documentation, tools, MATLAB/Simulink, model-based design, Software Design Description, Simulink Report Generator

ACM Reference Format:
Alexander Schaap, Gordon Marks, Vera Pantelic, Mark Lawford, Gehan Selim, Alan Wassyng, Lucian Patcas. 2018. Documenting Simulink Designs of Embedded Systems. In *ACM/IEEE 21th International Conference on Model Driven Engineering Languages and Systems (MODELS '18 Companion), October 14–19, 2018, Copenhagen, Denmark.* ACM, New York, NY, USA, Article 4, 5 pages. https://doi.org/10.1145/3270112.3270115

1 INTRODUCTION

High-quality software documentation has been viewed by many authors as a major factor in the successful development and maintenance of high-quality software [8–10]. In practice, however, software documentation is often neglected due to cost/time pressures resulting in either a complete lack of documentation, or documentation that is poorly organized, incomplete, ambiguous, inaccurate, and not kept up-to-date during software maintenance [9].

Model-based design (MBD) has become a prevalent paradigm in development of modern embedded systems, with MATLAB/Simulink as a widely used MBD environment across industries. In order for model-based development to be successful in delivering software that is fit for purpose and maintainable, it seems reasonable to apply relevant software engineering practices in MBD with Simulink. In particular, our experience working with large automotive companies employing MBD with Simulink confirms that improvements in appropriate software documentation is likely to facilitate effective and efficient maintenance of large production-scale Simulink models. A common reason for the lack of an adequate documentation culture in MBD is the "model as documentation" perspective. The analogous perspective plagued traditional software engineering expressed by the "code is documentation" motto [8]. Arguments against the perspective that models are documentation are analogous to the ones presented in [8], applied to models instead of code: production-size models—similar to complex code—are notoriously hard to understand without proper abstractions. Different abstractions offering complementary views of models are essential in developing and maintaining models effectively and efficiently.

While good software engineering practices dictate that a number of documents be produced during software development, the focus of this work is on a model's *Software Design Description (SDD)*, essential in development and maintenance of large Simulink models of embedded systems. The SDD details the model's interface for users of the model and *how* the model implements its requirements for developers of the model by describing the internals of the model. The term Software Design Description is borrowed from traditional software engineering terminology. In this paper, we suggest a template for the SDD of Simulink models of an embedded system, and propose semi-automated tool support for the SDD's production.

Although we have identified a strong need in industry for SDD templates for Simulink models, there seems to be little existing work on documenting Simulink models. Rau [1] describes an integrated approach to documenting Simulink models, however the resulting tool is not available. In particular, the work advocates integration of a model and documentation (making documentation available from the model) so that the two can be maintained simultaneously. Further, Rau emphasizes the need to precisely define what is to be documented and how, so that developers can fill in the provided fields. It appears MathWorks has incorporated many of Rau's suggestions into *Simulink Report Generator*, a tool that automatically generates documentation for Simulink models and simulations. In particular, the introduction of DocBlocks into Simulink allows developers to document directly in a model. In

MODELS '18 Companion, October 14–19, 2018, Copenhagen, Denmark

ACM ISBN 978-1-4503-5965-8/18/10…$15.00
https://doi.org/10.1145/3270112.3270115

fact, we use Simulink Report Generator to support the generation of the design documentation suggested in this paper. While Simulink Report Generator itself offers a template for a model's design documentation, the information in the template is very low-level, automatically extracted from executable Simulink models. Consequently, the resulting document offers very little abstraction from the Simulink implementations, which limits its use in understanding, maintaining, and testing designs. This is exactly what this paper hopes to rectify by proposing a SDD template and automating generation of SDDs based on the template with an in-house tool that leverages Simulink Report Generator.

It is also worth noting that our work, although inspired by experience in the automotive industry and applied to automotive software, generalizes to documenting Simulink models across industries.

The main contributions of the work presented in this paper are:

(1) We define a template for SDDs of Simulink models of embedded systems. Our experience points to the lack of the definition of documentation content as a major obstacle to effective utilization of documentation in software development. First, developers are often not aware of what information should be included in the documentation. This is especially true for domain experts, experts in a specific field of application (power electronics, motor control, *etc.*), who typically are unfamiliar with software engineering concepts. Second, even if a template for documentation exists, it often only loosely defines the desired content. This issue leads to inconsistent documentation across different Simulink models within the same embedded system, effectively making collaboration between stakeholders very hard, even impossible, as the stakeholders do not know where to find the needed information.
(2) We developed a tool, the *Simulink Design Documenter*, to highly automate production of SDDs for Simulink models. The tool uses MathWorks' Simulink Report Generator and an in-house tool, the *Signature Tool* [2, 7], to partially automate production of SDDs from Simulink models. Simulink Design Documenter is available via MATLAB Central[1]. A demo for the tool is available via YouTube[2].

The remainder of this paper is organized as follows. Section 2 presents the SDD template and illustrates it with an example. Subsequently, Section 3 highlights the tool support provided for our template. Finally, Section 4 presents conclusions.

2 SDD TEMPLATE

It seems obvious that the content of a Simulink SDD can draw parallels with a SDD in traditional software engineering. Therefore, existing recommendations on traditional software design documentation can be adopted, including, e.g., the ones given in standards such as [4]. The running example in this paper will be a Simulink model designed to estimate the rotor position in an electric motor of a hybrid electric vehicle. The system[3] was developed as part of a research project with a large automotive OEM (Original Equipment Manufacturer). The focus is on the signal-injection subsystem shown in Fig. 1, but the approach is applicable to the whole model. The proposed SDD table of contents is shown in Listing 1.

[1]https://www.mathworks.com/matlabcentral/fileexchange/63252
[2]https://youtu.be/WBmHbhPmgRs
[3]The term "system" in this section is used to indicate both a Simulink model and a subsystem.

Listing 1: Outline of SDD generated by the tool

```
Title Page
Changelog
Table of Contents
1 Introduction
  1.1 Document Purpose
  1.2 Scope
2 Preliminaries and Notation
  2.1 Acronyms
  2.2 Definitions
  2.3 Notation
3 Design Description of "System of Interest"
  3.1 Purpose
  3.2 Interface
    3.2.1 Inputs
    3.2.2 Outputs
    3.2.3 Updates
  3.3 Calibrations
  3.4 "System of Interest" Requirements Specification
  3.5 Internal Design
    3.5.1 System Snapshot
    3.5.2 Subsystems and Functions
    3.5.3 Local Declarations
    3.5.4 Description
    3.5.5 Rationale
    3.5.6 Anticipated Changes
    3.5.7 Design Description of Child Subsystem 1
          (of "System of Interest")
          <Structure recurses back to Section 3.1>
    3.5.8 Design Description of Child Subsystem 2
          (of "System of Interest")
          <Structure recurses back to Section 3.1>
    ...
4 Appendix
```

The remainder of this section describes items in Listing 1.

Title Page & Changelog. The sections are included as traditional parts of live documentation.

Introduction. Chapter 1 contains the purpose and the scope of the document, *not* the purpose and the scope of the Simulink design. An example purpose:

> The purpose of this document is to provide a detailed design description for this system. To provide the description, the system will be described in a hierarchical fashion starting with the top-level and progressively working toward lower level descriptions of the most significant subsystems of the system.

Preliminaries & Notation. Chapter 2 contains subsections for definitions, acronyms, and the notation used in the document. Each subsection is in turn divided into general and system specific subsections. An example of the definitions is shown below:

2.2 Definitions.

General Definitions.

State of Charge (SOC): The percentage of usable charge (Ampere-hours) contained in a battery relative to its usable capacity when fully charged. While 0% and 100% are not absolute limits, they are recommended extremes for cell state. From the user's perspective, it is the equivalent of a fuel gauge for (hybrid) electric vehicles. The inverse of the SOC is Depth of Discharge (DoD, also a percentage). If the battery is fully charged, the SOC is 100% and the DoD is 0%.

System-Specific Definitions.

N/A

Design Description. Chapter 3 documents the design of the model and its constituent subsystems. It is divided into subsections covering Purpose, Interface, Calibrations, Requirements Specification, and Internal Design.

Purpose. A succinct description of *what* the system is designed to accomplish, e.g.:

> This subsystem injects sinusoidal signals and voltage pulses, which are to be utilized for the motor for rotor position estimation and magnetic polarity detection.

Interface. A Simulink system has Inports and Outports—explicit inputs and outputs of the system, respectively. Further, there exist two mechanisms in Simulink that result in implicit data flow: Data Stores and Goto/From mechanisms. A detailed analysis of data flow mechanisms in Simulink can be found in [2]. The interface of a system in a SDD captures both explicit and implicit data flow through the system, and includes lists of all the system's inputs, outputs and updates [2]. Inputs include both Inports and implicit inputs corresponding to Data Stores defined outside of the system that are only read from in the current system or in any of its children, as well as Froms reading from Gotos defined and located outside of the system. Similarly, outputs include Outports and the implicit outputs corresponding to Data Stores defined outside the system that are only written to in the current system or any of its children, as well as Gotos located in the system or any of its children, accessible from outside of the system. Updates include all Data Stores defined outside of the system that are both read from and written to in the system or any of its children. Further, for each of the data items in the interface, its data type, range (if applicable), units and description are included. An example is shown below:

3.5.8.4.8.2. Interface.

3.5.8.4.8.2.1. Inputs.

Table 3.3. Inports

Name	Data Type	Min	Max	Unit	Description
State	uint16	0	1	N/A	Boolean to enable or disable the inverter. (From the CAN bus.)

Data Store Reads.

N/A

Global Froms.

N/A

Scoped Froms.

N/A

Calibrations. Lists the model's configurable constants. The section is not applicable to subsystems, but only the top-level system (model) being documented.

Requirements Specification. Documents black-box behaviour of a system. Maintaining traceability between the documented requirements and the components of the design that fulfill them is very important. When the Simulink system implements requirements documented in an external document (in e.g., MS Word), the developer should create traceability links between blocks in the system and the requirements document. Any links to external requirements should be displayed in a table:

Link#	Link Description	Link Target (Document)
1.	"Estimate rotor position without encoder sensor"	..\..\..\requirements\srs.docx

Internal Design. Describes the internal details of the Simulink system. It contains: the System Snapshot, Subsystems and Functions, Local Declarations, Description, Rationale, and Anticipated Changes sections as well as nested Design Descriptions of any subsystems under the current system that are useful toward understanding the overall system.

System Snapshot. This should be an image of the system of interest as shown in Fig. 1.

3.5.8.4.8.4.1. System Snapshot.

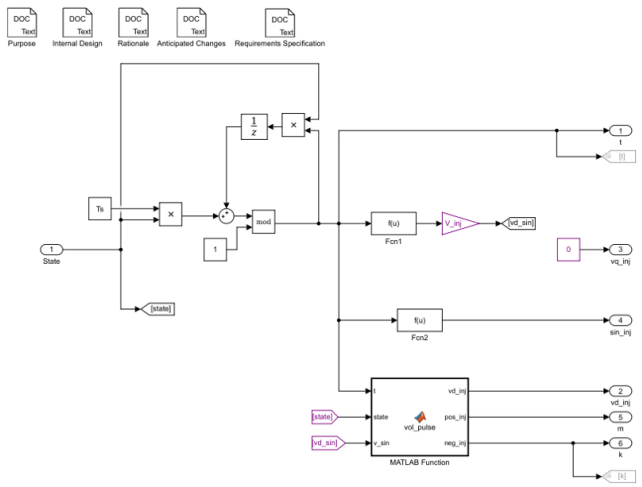

Figure 1: Example System Snapshot

Subsystems and Functions. Contains a list of subsystems, libraries, model references and functions included in the system, as illustrated below for the system from Fig. 1:

Block	Description
MATLAB Function	Determine when to inject a voltage pulse or high frequency sinusoidal wave.

Local Declarations. Includes a table of declarations local to the system including local data store declarations and Goto Tag Visibility blocks that define accessibility of Goto block tags.

Description. A description of how the internal structure of the system fulfills its purpose. It should describe internal algorithms, signals, and constants. For example:

> If the 'State' boolean input is 0 (or false), t (time) stays at 0. If 'State' is 1, then time (t) monotonically increases by *Ts* until it reaches 1 (at which time it is reset to 0).
> Note that when State changes from 1 to 0, the time stays constant for one step. This does not make a difference for the end result. All other outputs are calculated based on time and state.

Rationale. Although there exists wide consensus on the importance of capturing and using a design rationale [5, 6], the concept is still typically neglected in industrial software engineering practice. Design rationale captures reasons behind design decisions. The rationale includes the alternatives that were considered in the design process, and presents justification for selecting a design over the alternatives. Rationale captures important knowledge about a design, especially in organizations where there is a high staff turnover. It enables efficient maintenance by saving developers' time on exploring design alternatives that a previous developer (or perhaps the same one) already considered. Documenting rationale also saves valuable resources during design reviews: again, the reviewer may consider alternatives and compare them to the final design. The rationale can be captured with different levels of formality. While most of the research in the area focuses on an informal capture of design rationale, there has been some work done on a formal representation of design rationale for model-based designs [3]. However, *how* to document the rationale is not the focus of this paper. An example rationale is listed below:

3.5.8.4.8.4.6. Rationale.

The decision was made to inject voltage on the d-axis instead of the q-axis because experimentation demonstrated that this yields better results. Time increases monotonically and stays constant for one time step before dropping to zero when State changes to 0. This has no effect on the result, and is a result of an iterative design process. The MATLAB function vol_pulse redefines Ts, which should be provided as an input by a constant block reading from the accompanying MATLAB file. This function is structured for clarity and ease of experimentation, not efficiency. vd_sin is always calculated, again for simplicity rather than efficiency (but could be done conditionally instead).

Anticipated Changes. Describes what portions of the system are likely to change so that the design and implementation can accommodate such changes.

Appendix. The last chapter is Appendix and may contain any supplementary information as defined by the developer.

Of course, the proposed template will not fit the needs of every embedded system or organization. Instead, it is intended as a starting point for a template that can be adapted to suit particular needs. Section 3.3 discusses template customization.

3 TOOL SUPPORT

In this section, we discuss the tool, the Simulink Design Documenter, that automates the production of a SDD from the template described in the previous section. The Simulink Design Documenter leverages two tools: MathWorks' Simulink Report Generator and our Signature Tool. First, we will introduce the leveraged tools. Then, we will describe Simulink Design Documenter.

3.1 Supporting tools

3.1.1 Simulink Report Generator. Simulink Report Generator is a MathWorks tool that generates reports from Simulink models and simulations in a variety of file formats. Simulink Report Generator provides a graphical interface, which is used to modify report setup files (`.rpt` files) that specify what items the user wants to include in a report. To do this, the user selects and configures a series of built-in "components" that represent commands to execute in constructing a specific report. These components are analogous to lines of code in traditional imperative programming languages. For example, the "System Snapshot" component automatically adds a snapshot of a given Simulink system into the report.

The choice of Simulink Report Generator as the main tool to support the generation of SDDs from our template was natural—Simulink Report Generator appeared to be the only commercially available tool capable of significantly aiding in the documentation process for Simulink systems.

3.1.2 Signature Tool. The Signature Tool [2, 7] is an in-house tool that extracts the interface of Simulink systems. The Simulink Design Documenter uses the Signature Tool to generate the tables within both the Interface and Local Declarations sections of generated SDDs.

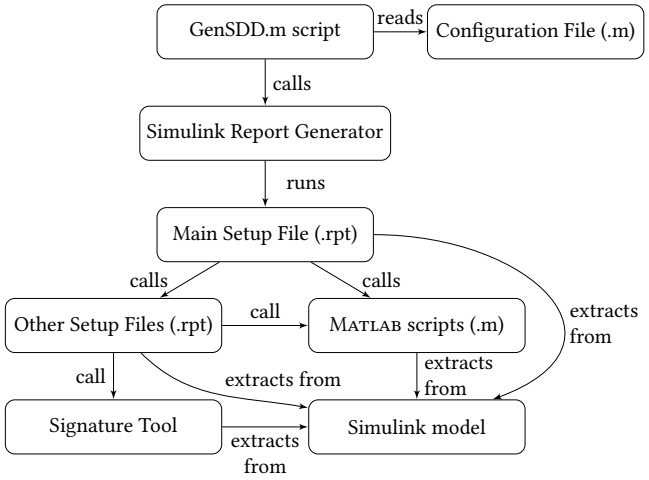

Figure 2: Architecture of Simulink Design Documenter

3.2 Simulink Design Documenter

Figure 2 provides an architectural view of the Simulink Design Documenter. The tool is run through the `GenSDD.m` script which reads the Configuration File and uses Simulink Report Generator to run the Main Setup File and create the SDD. The Main Setup File is created via Report Explorer (the GUI for Simulink Report Generator). Other Setup Files (also created using Report Explorer) are nested within the main one and are used to ensure consistent formatting as well as to construct the hierarchy of sections in the Design Description portion of the generated document. The Other Setup Files use the Signature Tool to get data for the Interface and Local Declarations sections of the SDD. Finally, MATLAB scripts are used to help with formatting and structuring the hierarchy of sections.

3.3 Using Simulink Design Documenter

This section explains how to generate a SDD using the Simulink Design Documenter. A more complete and detailed guide to creating SDD documents with the Simulink Design Documenter comes with the tool itself.

The first step is to create a configuration file for the system being documented by modifying a default file that comes with the tool. Configuration options allow the developer to set the author name(s), add an image to the title page, etc. The most notable option is the selection of subsystems to document. By default, the tool will generate documentation for all subsystems within the top three layers in the subsystem hierarchy of the system being documented.

Once the configuration file is in place, the user needs to fill out the Purpose, Internal Design, Rationale, and Anticipated Changes DocBlocks for the systems being documented. The content of each DocBlock will be copied to the corresponding section of the SDD once the SDD is generated. The content within DocBlocks can be in either ASCII text or RTF format. The Simulink Design Documenter provides a library of pre-configured DocBlocks that correspond to different sections and explains what content the user should provide. For example, a Purpose DocBlock is included for the overall system and each subsystem being documented. Changelog, System Specific Acronyms, System Specific Definitions, and System Specific Notation sections from the template are also added via DocBlocks. Of course, for each of these sections, only one block is needed for the whole model rather than per subsystem, because they pertain to the whole SDD.

The other introductory sections, the General Acronyms, General Definitions, and General Notation sections, can be included by creating corresponding ASCII text or RTF files on MATLAB's search path. The Document Purpose and Scope can be created similarly potentially leveraging default content provided with the tool. These sections are meant to include general information which would apply across numerous SDDs within an organization.

Requirements for a system are included via a DocBlock and/or RMI (Requirements Management Interface) links to external documentation, such as requirements and/or testing documentation. These links are only included when the appropriate option is set in the configuration file.

The system's calibrations may be added by indicating its calibrations file in the configuration file.

The interface tables and local declarations are included automatically and may show block names, physical units, minimum value, maximum value, data type, and description. Also, the content of sections Title Page, Table of Contents, System Snapshot, and Subsystems and Functions is automatically generated.

Once the aforementioned DocBlocks, RMI links, configurations, and external documents are in place, the tool automatically generates an SDD following the structure of Listing 1. Rerunning the tool regenerates the document from scratch and overwrites the previously generated document.

In order to customize the template itself, the developers need to edit the tool's report setup files, or other files included with the Simulink Design Documenter. For example, the changelog table skeleton can be edited using the familiar MS Word interface. As another example, a developer might want to swap two sections within the report. This would require only a simple modification: the developer would use the Report Explorer to locate the groups of components which generate each section within the appropriate report setup file(s), and would then reorder those components appropriately. Similar effort would be required for the addition of a new section into the report: this modification entails copying of components which create an existing section and then pasting them into the desired location in the report setup file, and changing the name of the section.

The tool has several limitations. For example, Simulink Report Generator lacks flexibility in formatting the generated documents. For example, the spacing between sections is often larger than desired.

4 CONCLUSION

In our collaboration with large automotive companies, we have witnessed how difficult and time consuming it is to document, adequately, complex software designs in Simulink models. This work addresses the issue by defining suggested content and format of the design documentation of Simulink models for large, complex embedded systems. This enables greater consistency within a document, as well as between documents inside an organization. The tool presented in this paper effectively removes much of the manual documentation burden from developers. Developers still need to concern themselves with providing relevant content, but the guidance provided by the tool helps even in this regard. Future work includes adding a GUI to improve the tool's usability, and performing the evaluation of the tool in an industrial setting.

REFERENCES

[1] Andreas Rau. [n. d.]. Integrated Specification and Documentation of SIMULINK Models. ([n. d.]). http://www.it-designers.info/uploads/media/iac2002.pdf International Automotive Conference.

[2] Marc Bender, Karen Laurin, Mark Lawford, Vera Pantelic, Alexandre Korobkine, Jeff Ong, Bennett Mackenzie, Monika Bialy, and Steven Postma. 2015. Signature Required: Making Simulink Data Flow and Interfaces Explicit. *Science of Computer Programming* 113, Part 1 (2015), 29–50. https://doi.org/10.1016/j.scico.2015.07.005 Model Driven Development (Selected & extended papers from MODELSWARD 2014).

[3] Adriana Pereira De Medeiros, Daniel Schwabe, and Bruno Feijó. 2005. Kuaba ontology: design rationale representation and reuse in model-based designs. In *Conceptual Modeling–ER 2005*. Springer, 241–255.

[4] IEEE. 2009. IEEE Standard for Information Technology – Systems Design – Software Design Descriptions. *IEEE Std 1016-2009 (Revision of IEEE Std 1016-1998)* (July 2009), 1–58.

[5] A. P. J. Jarczyk, P. Loffler, and F. M. Shipmann. 1992. Design rationale for software engineering: a survey. In *Proceedings of the Twenty-Fifth Hawaii International Conference on System Sciences*, Vol. ii. 577–586 vol.2. https://doi.org/10.1109/HICSS.1992.183309

[6] N. G. Leveson. 2000. Intent specifications: an approach to building human-centered specifications. *IEEE Transactions on Software Engineering* 26, 1 (Jan 2000), 15–35. https://doi.org/10.1109/32.825764

[7] Vera Pantelic, Steven Postma, Mark Lawford, Monika Jaskolka, Bennett Mackenzie, Alexandre Korobkine, Marc Bender, Jeff Ong, Gordon Marks, and Alan Wassyng. 2018. Software engineering practices and Simulink: bridging the gap. *International Journal on Software Tools for Technology Transfer* 20, 1 (01 Feb 2018), 95–117.

[8] David Lorge Parnas. 2011. *Precise Documentation: The Key to Better Software*. Springer Berlin Heidelberg, Berlin, Heidelberg, 125–148. https://doi.org/10.1007/978-3-642-15187-3_8

[9] David Lorge Parnas and Paul C Clements. 1986. A rational design process: How and why to fake it. *IEEE transactions on Software Engineering* 2 (1986), 251–257.

[10] Ian Sommerville. 2001. Software documentation. *Software Engineering* 2 (2001), 143–154.