APPLICATION OF A PROCESS-ORIENTED BUILD TOOL TO THE DEVELOPMENT OF A BM3 SLAVE CONTROLLER SOFTWARE MODULE

Purav Panchal*, Nina Sorokina §, Stephan Myschik*
*Institute for Aeronautical Engineering
§Department of Electrical Engineering
Universität der Bundeswehr München, 85521 Neubiberg, Germany

Konstantin Dmitriev†, Florian Holzapfel†
†Institute of Flight System Dynamics, Technische Universität München, 85748 Garching, Germany

Abstract

Software development of safety critical applications like battery controller, flight controller, medical devices, military weapon systems, etc. requires significant number of verification and validation steps to ensure that the software is compliant towards certification standards. This results in extensive documentation, strict methodologies, and verification activities, but also creates a space for researchers to invent automation techniques to make the software development process simpler. Critical information on how the development and verification tools are interlinked with each other during the development process is usually a part of intellectual property of large aerospace companies. Such information is not available publicly and this hinders the growth of startups and small/medium enterprises. To overcome such hurdles, a process-oriented build tool based on MathWorks' MATLAB and Simulink has been already developed and is used in many flight control applications. In this paper, an application of this build tool to develop a slave controller of a Battery Modular Multilevel Management (BM3) system while undergoing process development steps required by aerospace safety standards is presented. The tool provides a development environment with predefined model templates, block libraries, configuration settings and jobs for executing process-relevant tasks like automatic code generation, code verification, model verification, etc. The tool also ensures consistency of model artifacts and compatibility with downstream tools used for verification and validation on model and code level. The paper presents several verification and development results which authenticates the mentioned advantages of the build tool.

1. INTRODUCTION

Safety-critical applications refers to systems whose failure can affect human safety. Example of safety-critical software include applications in aerospace, automotive, railway, medical, and nuclear industries. Significance of safety-critical software development can be realized after looking at several fatal incidents that have taken place in the history of humankind. In 2018-2019, Lion Air Flight 610 and Ethiopian Airlines Flight 302 crashed probably due to reliability of Boeing's Maneuvering Characteristics Augmentation System (MCAS) on a single source of information [1]. In 1997, Korean Air Flight 801 crashed during landing. One of the possible causes of the crash was improper configuration of the minimum safe altitude warning system [2]. In 2015, Toyota recalled 112,500 vehicles due to possible power steering issues and electric vehicle safety problems [3].

After observing such incidents, lot of importance has been given to the safety assurance of such software. Hence, safety-critical software must follow lot of safety measures, documentation, and strict methodologies according to the standards. Secondly, large scale companies usually do not publicize the information on how the toolchain is setup and interlinked, hence smaller companies struggle to compete. In industries, software development lifecycle like V-Model with Model-based Development approach is applied to develop a structured environment and accelerate verification and validation steps. However, in safety-critical applications and specially in aviation, the implementation of change in requirements, e.g., adding a new feature after certification is expensive and requires lot of efforts and time. This problem is known as a 'big-freeze' problem [4]. To overcome these several problems, a process-oriented build tool called *'mrails'* is already developed and used in several complex flight control and avionics software development projects at Institute of Flight System Dynamics, TUM, and the Institute for Aeronautical Engineering at Universität der Bundeswehr München [5].

This build tool provides a structured but flexible development environment in MATLAB and Simulink framework [6]. The tool ensures smooth development of the functionality while maintaining a consistent artifact tree and traceability [7]. By doing so, tasks for demonstrating process compliance, e.g., for RTCA DO-178C/331, can be supported [8,9].

This research is intended to develop a slave controller for a battery system using the build tool and present several advantages of it. The slave controller is a part of battery controller consisting of one master controller and several

doi: 10.25967/570308

similar slave controllers for each submodule. The BM3 system is based on an integrated 3-switch inverter topology [10,11] and is proposed for an electric aircraft in context of the project ELAPSED [12] which focuses on developing an electric powertrain with a new propulsion system.

The structure of the paper is as follows: the following section describes the methodologies that are implemented in this research; the build tool *mrails* and the battery controller. Section 3 presents the controller development process, the verification and validation results are mentioned in Section 4. Finally, the future work and conclusions are discussed in Section 5 and 6 respectively.

2. METHODOLOGIES

2.1. Process-Oriented Build Tool

As the name implies, the build tool is developed to support the process compliant development of software according to standards like DO-178C/DO-331 [13]. The tool incorporates modular model-based design approaches and provides the developer with a friendly environment to perform development and verification activities without bothering about the configuration settings, tools interlinking and artifacts handling. Model-based design approach provides advantages like ease in readability, early detection of defects by unit testing and code reuse [14]. The tool incorporates a so-called lifecycle package which provides the necessary artifact containers, configuration settings, and automated verification jobs. These containers are used to create model and its elements like constants, parameters, tests, and buses. An overview of the build tool is shown in Fig. 1. Results from the automated code generation and verification tasks are accumulated and can be accessed by a web-based HTML report.

2.1.1. Advantages of the build tool

- *Automatic Code Generation*: The lifecycle package included in the tool provides code generation task. The task is divided into two parts – shared code generation and code generation. Shared code contains the code for interfaces of the model which once validated, do not need to change. Code generation follows a bottom-to-top approach with incremental build of only required models in the architecture hierarchy [15].

- *Toolchain Setup*: The build tool is based on MATLAB/Simulink and includes use of applications like Embedded Coder, Simulink Test Manager, Model Advisor and Design Verifier. Polyspace is used for code proving and defect analysis. The build tool configures these applications automatically with predefined configurations and collects artifacts after task execution. This relieves the stress of toolchain setup and maintains consistency in a distributed team.

- *Lifecycle Package*: The tool itself provides ability to incorporate different lifecycle packages. These packages contain development and verification jobs for the models and code. Along with artifacts handling, the package also provides containers to create model elements like top-level/reusable models, buses, test cases, parameters, etc. These model elements are created with settings to comply with DO-178C/DO-331 standards [13].

- *Incremental Verification and Traceability*: When a change is detected in a particular model or code, the design and code verification tasks are only performed on those artifacts avoiding re-verification. Consistency of the artifact is also checked, for e.g., whether the generated result matches to latest version of model. The verification results are displayed in a web-based HTML status report and the jobs can be traced back to the required model elements. This provides traceability of the artifacts.

- *Integration of Multiple Modules*: In case of modularized software, the distributed team should be able to integrate their software modules after development and verification. The build tool provides

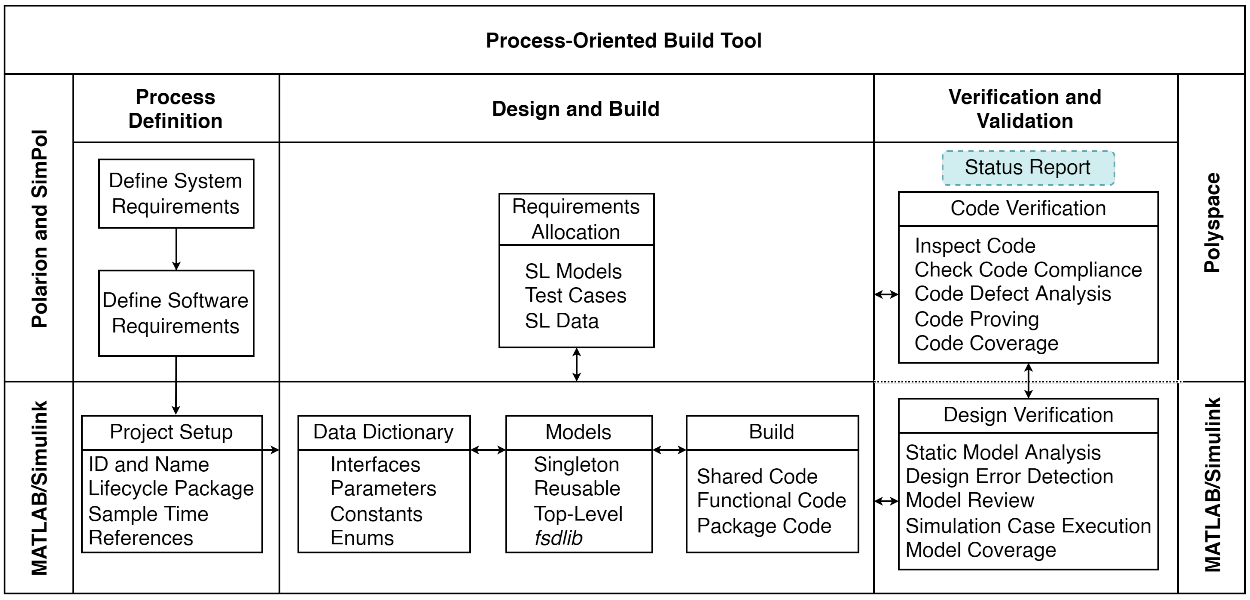


Fig. 1: Overview of the process-oriented build tool

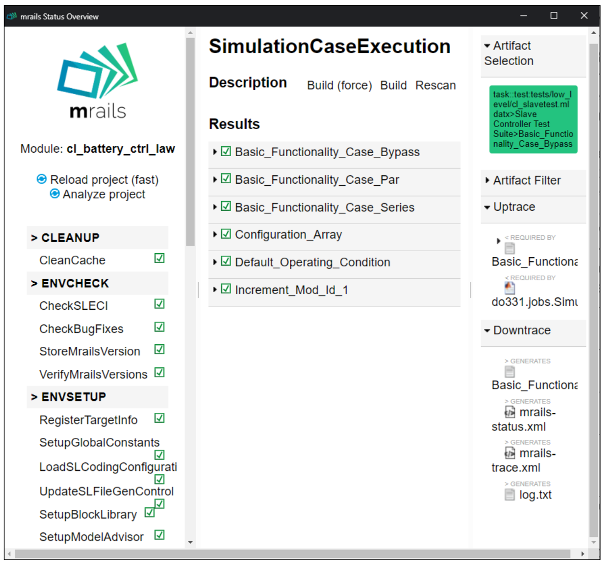
ability to handle the integration and differentiate the model elements required to avoid any changes to the individual modules [15]. This plays an important role in case of complex software architectures, where the top-level model consists of several submodules.

2.1.2. Process Workflow

The process flow is divided into three dependent stages: process setup, design and build, verification and validation as shown in Fig. 1.

- *Process Definition*: The process setup phase consists of setting up the system and software requirements. This can be done in Polarion which is an 'Application Lifecycle Management' tool used for managing requirements and achieving agility [16]. The next step is to create a MATLAB project by using build tool command '*mrails create-module*' The user is asked for the required lifecycle package (e.g., DO-331), sample time, module id and name. This creates a project with required configuration settings. The user can then add other projects as references which will be treated as dependencies by the tool [15].

- *Design and Build*: The models, interfaces, parameters, constants, and enums are created using containers provided by the tool with necessary settings to comply with guidelines. For requirements allocation, another tool called SimPol is used which links Polarion work items to MATLAB/Simulink elements like models, test cases, data, and code [17]. Models can be then built using a bottom-to-top code generation approach. This is executed in two stages: a) shared code generation that generates shared code for the model interfaces and b) functional code generation which incrementally generates code for the models. Detailed description of the code generation process is described in [15]. The tool contains custom Simulink block library '*fsdlib*' containing commonly used blocks with required parameter settings, for example, a protected division block is provided which contains switches to prevent division by zero [18].


mrails Status Overview

mrails

Module: **cl_battery_ctrl_law**

Reload project (fast)
Analyze project

> CLEANUP
CleanCache ☑
> ENVCHECK
CheckSLECI ☑
CheckBugFixes ☑
StoreMrailsVersion ☑
VerifyMrailsVersions ☑
> ENVSETUP
RegisterTargetInfo ☑
SetupGlobalConstants ☑
LoadSLCodingConfigurati ☑
UpdateSLFileGenControl ☑
SetupBlockLibrary ☑
SetupModelAdvisor ☑

SimulationCaseExecution

Description Build (force) Build Rescan

Results
☑ Basic_Functionality_Case_Bypass
☑ Basic_Functionality_Case_Par
☑ Basic_Functionality_Case_Series
☑ Configuration_Array
☑ Default_Operating_Condition
☑ Increment_Mod_Id_1

Artifact Selection
task::test:tests/low_level/cl_slavetest.mldatx>Slave Controller Test Suite>Basic_Functionality_Case_Bypass
Artifact Filter
Uptrace
< REQUIRED BY
Basic_Functiona
< REQUIRED BY
do331.jobs.Simu
Downtrace
> GENERATES
Basic_Functiona
> GENERATES
mrails-status.xml
> GENERATES
mrails-trace.xml
> GENERATES
log.txt

Fig. 2: Status Report

- *Verification and Validation*: The tool provides several automatic design and code verification jobs. These jobs are called by tool commands, for example, '*mrails staticmodelanalysis*' which runs a function stack that calls Simulink Model Advisor with custom and MathWorks' checks on the model. Likewise, code verification jobs like Code Defect Analysis and Code Proving use Polyspace tool [19]. Results of all the design and code verification jobs are available in a single web-based HTML status report as shown in Fig. 2. The report also provides uptrace and downtrace option to trace the affected files.

2.1.3 RELATED WORK

The build tool is continuously being improved and its capabilities are augmented. Recently, integration of multiple modules was developed for the build tool and applied on the same application as concerned in this paper. This will be presented in DASC 2022 conference [15]. Another application of this build tool is presented in [18], where an Incremental Nonlinear Dynamic Inversion INDI based flight controller is developed and verified using the build tool. A Continuous Integration (CI) setup is also being deployed for the improvement of the build tool [13].

2.2. Battery Modular Multilevel Management

The BM3 system is based on an integrated 3-switch inverter topology [10,11]. The inverter topology has several features like flexible interconnections between the battery cells to achieve optimum efficiency, match required load voltage, increase lifetime, and increase fault tolerance of the system.

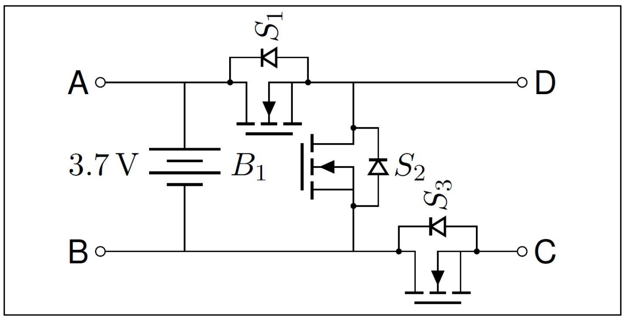


Fig. 3: BM3-Module with MOSFETs [11]

A submodule of BM3 system is shown in Fig. 3 which is controlled by a slave controller. The battery controller system structure is shown in Fig. 4. S1, S2 and S3 represent MOSFET switches and terminals 'A', and 'B' are connected to 'C' and 'D' terminals of the adjacent module respectively. Such kind of topology provides three different states of the module: serial, parallel and bypass. Principle advantages of BM3 module is to have a flexible output batterypack voltage, achieved by dynamically changing the cells interconnection between series and parallel states, inherited cell balancing and bypassing defective cells if needed.

The battery controller for BM3 module consists of two main components: one master and several slave controllers as shown in Fig. 4. The number of slave controllers depend on the number of BM3 modules used.

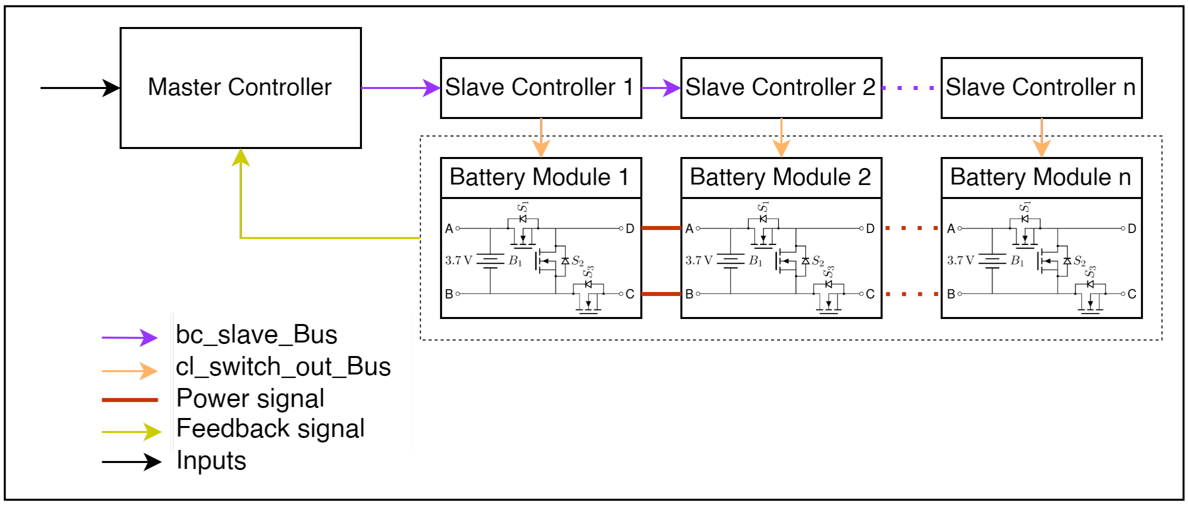


Fig. 4: Battery Controller Structure

The master controller receives all the necessary information like current state of each cell (temperature and voltage), current output voltage of the battery pack and DC required voltage via input and a feedback signal from the battery modules. Depending on these inputs, the master calculates required connection configuration of the BM3 modules and generates a configuration array which contains the configuration selecting value of each module. This is sent to the first slave controller via 'bc_slave_Bus' which also contains the module id. As stated before, three types of states are available. Series state is identified by value 1, parallel by value 2 and bypass which is also the default state by 0.

	S1	S2	S3
Series (1)	0	1	0
Parallel (2)	1	0	1
Bypass (0, default)	1	0	0

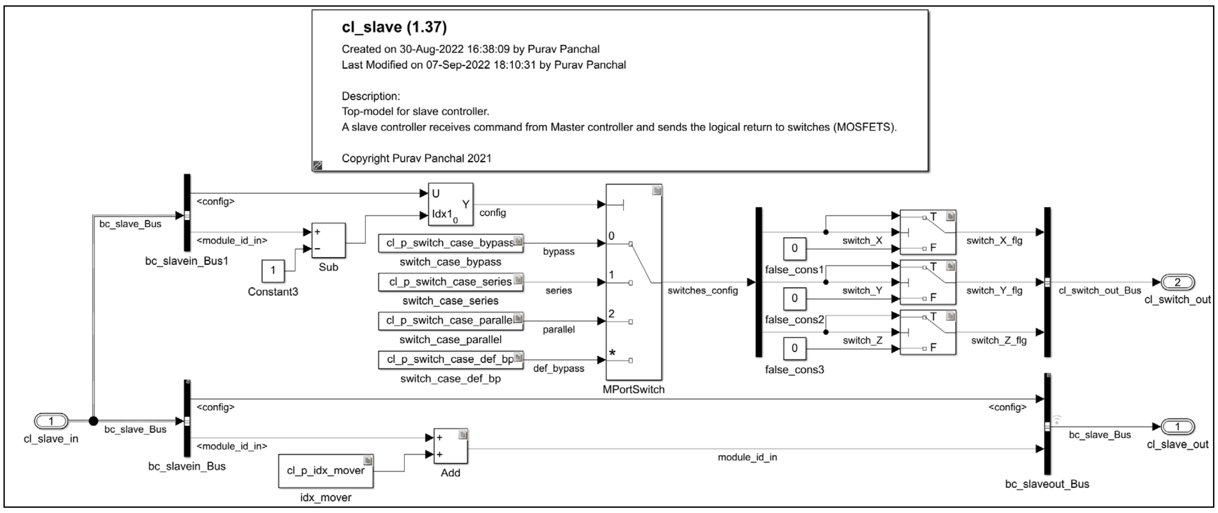
Table 1: Switch configuration according to module operating condition [20]

Depending on this selection value, the slave controller selects the configuration for the switches. This configuration is then sent via bus 'cl_switch_out_Bus'. The switch configuration is shown in Table 1 where the states are followed by a configuration selecting value shown in rounded brackets.

3. SLAVE CONTROLLER DEVELOPMENT

The slave controller logic is shown in Fig. 5. Input of the slave controller is a bus 'bc_slave_bus' containing two elements: 'config' and 'module_id_in'. A multi-port switch is used to select the desired switch configuration depending on the configuration selecting value for the module id.

According to the selected configuration, the switch out bus is created with required configuration the 'cl_switch_out_bus' then outputs Boolean flags for each switch and turns respective MOSFET switches on/off. The submodule index is incremented by 1 with a parameter 'cl_p_idx_mover'. The configuration array from the master controller is passed 'as-is' to the next slave controller along with the incremented module id. Following the process workflow as mentioned in 2.1.2, initially the system and software requirements are stored in a Polarion project. Few software requirements for the battery controller are shown in Fig. 6. These requirements are derived from the perspective of safety of the battery cells



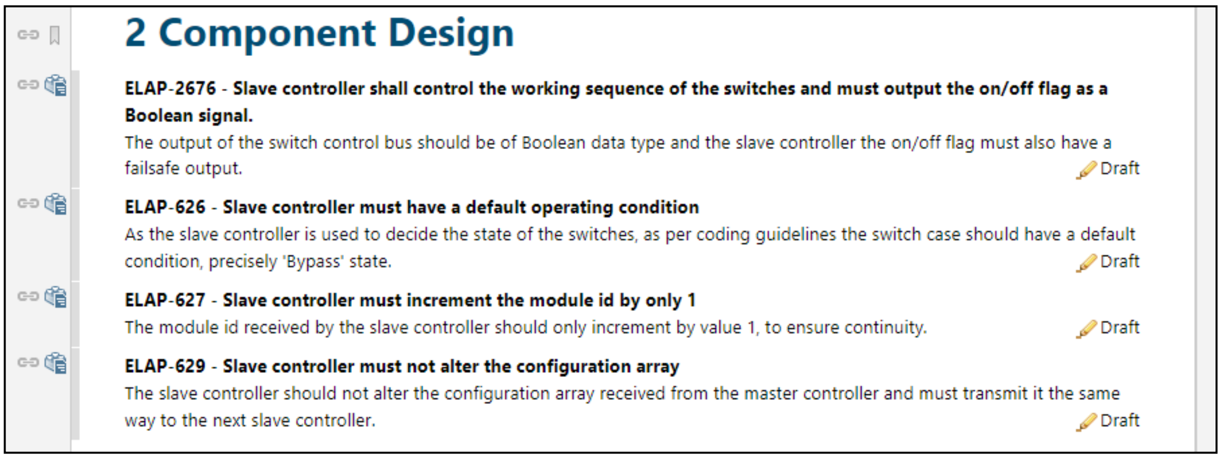
cl_slave (1.37)

Created on 30-Aug-2022 16:38:09 by Purav Panchal
Last Modified on 07-Sep-2022 18:10:31 by Purav Panchal

Description:
Top-model for slave controller.
A slave controller receives command from Master controller and sends the logical return to switches (MOSFETS).

Copyright Purav Panchal 2021

Fig. 5: Slave Controller Simulink Model



2 Component Design

ELAP-2676 - Slave controller shall control the working sequence of the switches and must output the on/off flag as a Boolean signal.
The output of the switch control bus should be of Boolean data type and the slave controller the on/off flag must also have a failsafe output. Draft

ELAP-626 - Slave controller must have a default operating condition
As the slave controller is used to decide the state of the switches, as per coding guidelines the switch case should have a default condition, precisely 'Bypass' state. Draft

ELAP-627 - Slave controller must increment the module id by only 1
The module id received by the slave controller should only increment by value 1, to ensure continuity. Draft

ELAP-629 - Slave controller must not alter the configuration array
The slave controller should not alter the configuration array received from the master controller and must transmit it the same way to the next slave controller. Draft

Fig. 6: Software requirements for slave controller

and modeling guidelines that were developed along the build tool [13]. In the design and build stage, a project is created with a module id 'cl', and a top-level model is created using the create command. This module refers to another module which has all the required global interfaces and parameters like 'bc_slave_Bus' and 'bc_p_no_of_modules', as of now. The use of another module for common interfaces can be justified by the fact that these interfaces once validated (design and code), will not change frequently and will maintain consistency during the development of master controller in future.

A custom Simulink library 'fsdlib' containing commonly used blocks with required parameter settings is used to design the top-level 'cl_slave' model as shown in Fig. 5. For each software requirement, a test case is developed. The tool also provides containers to create low-level and top-level test cases. The requirements are allocated to 'cl_slave' model and respective test cases.

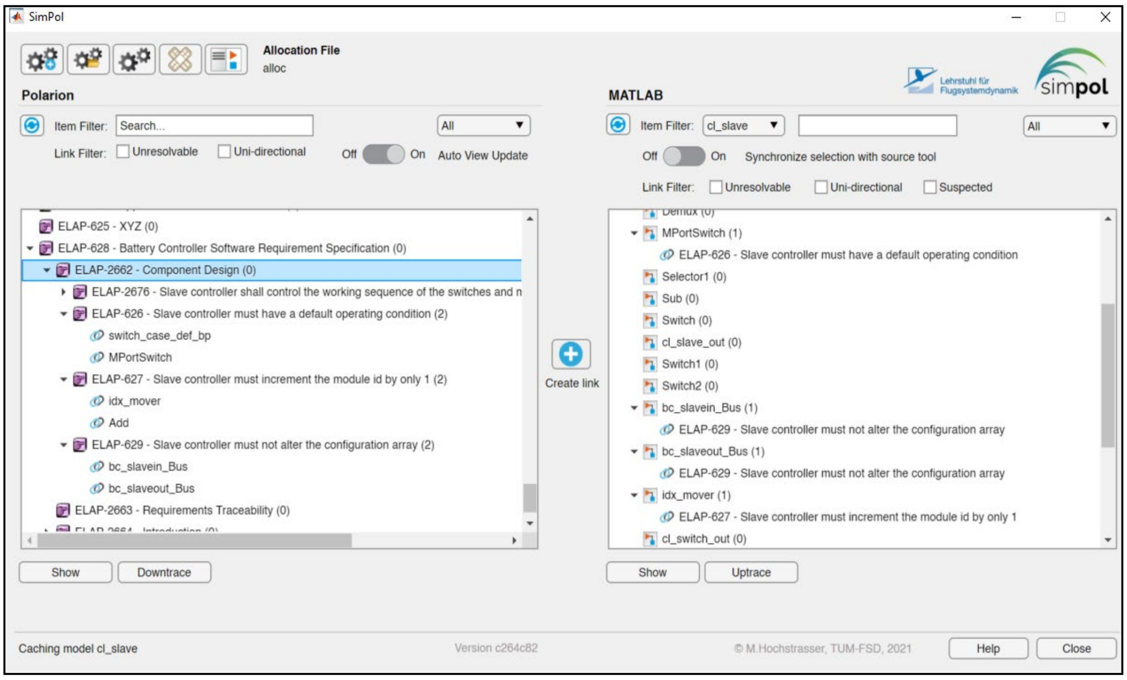
This is done using tool SimPol as shown in Fig. 7 where requirements are linked to the respective Simulink block elements. Similarly, the target can be changed to MATLAB test case and requirements can be then allocated to test cases. After the designing is finished, code generation can be executed. Parallelly, design verification can also be started. The tool provides direct commands to execute verification jobs, for example, static model analysis can be performed simply by calling '*mrails staticmodelanalysis*' command. Similarly other jobs are performed. The results are then accumulated in the status report as mentioned before. All available jobs are shown in Fig. 9.

4. VERIFICATION RESULTS

From the different available verification jobs, few important results from design and code verification along with traceability are discussed. All the results are available via an HTML status report of the build tool.

4.1. Design Verification

Design verification jobs include code static model analysis, design error detection, traceability, and model review.



SimPol

Allocation File
alloc

Polarion
Item Filter: Search... All
Link Filter: Unresolvable Uni-directional Off On Auto View Update

ELAP-625 - XYZ (0)
ELAP-628 - Battery Controller Software Requirement Specification (0)
ELAP-2662 - Component Design (0)
ELAP-2676 - Slave controller shall control the working sequence of the switches and n
ELAP-626 - Slave controller must have a default operating condition (2)
switch_case_def_bp
MPortSwitch
ELAP-627 - Slave controller must increment the module id by only 1 (2)
idx_mover
Add
ELAP-629 - Slave controller must not alter the configuration array (2)
bc_slavein_Bus
bc_slaveout_Bus
ELAP-2663 - Requirements Traceability (0)

Show Downtrace

Create link

MATLAB
Item Filter: cl_slave All
Off On Synchronize selection with source tool
Link Filter: Unresolvable Uni-directional Suspected

Demux (0)
MPortSwitch (1)
ELAP-626 - Slave controller must have a default operating condition
Selector1 (0)
Sub (0)
Switch (0)
cl_slave_out (0)
Switch1 (0)
Switch2 (0)
bc_slavein_Bus (1)
ELAP-629 - Slave controller must not alter the configuration array
bc_slaveout_Bus (1)
ELAP-629 - Slave controller must not alter the configuration array
idx_mover (1)
ELAP-627 - Slave controller must increment the module id by only 1
cl_switch_out (0)

Show Uptrace

Caching model cl_slave Version c264c82 © M.Hochstrasser, TUM-FSD, 2021 Help Close

Fig. 7: Requirements allocation using SimPol

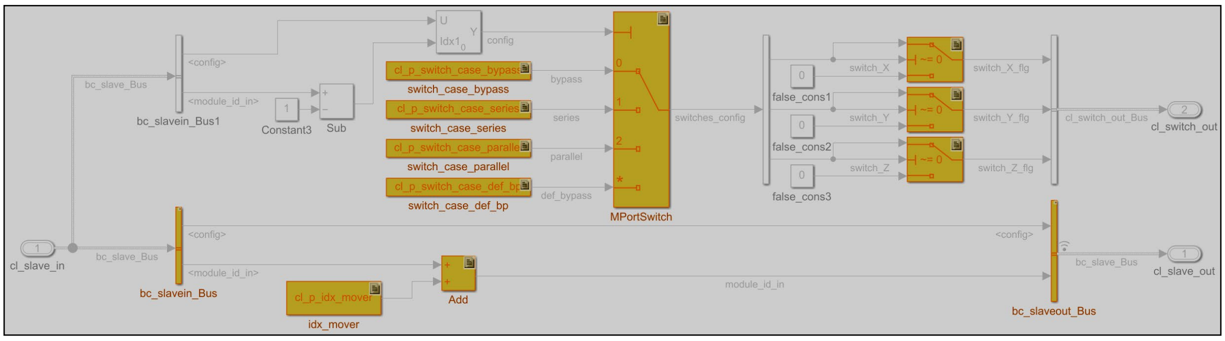


Fig. 8: Highlighted requirements on the model

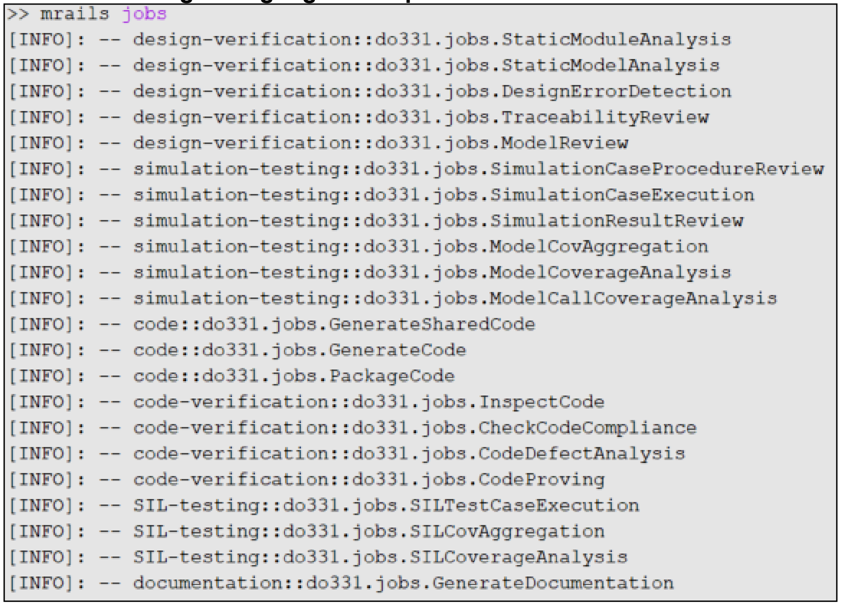

```
>> mrails jobs
[INFO]: -- design-verification::do331.jobs.StaticModuleAnalysis
[INFO]: -- design-verification::do331.jobs.StaticModelAnalysis
[INFO]: -- design-verification::do331.jobs.DesignErrorDetection
[INFO]: -- design-verification::do331.jobs.TraceabilityReview
[INFO]: -- design-verification::do331.jobs.ModelReview
[INFO]: -- simulation-testing::do331.jobs.SimulationCaseProcedureReview
[INFO]: -- simulation-testing::do331.jobs.SimulationCaseExecution
[INFO]: -- simulation-testing::do331.jobs.SimulationResultReview
[INFO]: -- simulation-testing::do331.jobs.ModelCovAggregation
[INFO]: -- simulation-testing::do331.jobs.ModelCoverageAnalysis
[INFO]: -- simulation-testing::do331.jobs.ModelCallCoverageAnalysis
[INFO]: -- code::do331.jobs.GenerateSharedCode
[INFO]: -- code::do331.jobs.GenerateCode
[INFO]: -- code::do331.jobs.PackageCode
[INFO]: -- code-verification::do331.jobs.InspectCode
[INFO]: -- code-verification::do331.jobs.CheckCodeCompliance
[INFO]: -- code-verification::do331.jobs.CodeDefectAnalysis
[INFO]: -- code-verification::do331.jobs.CodeProving
[INFO]: -- SIL-testing::do331.jobs.SILTestCaseExecution
[INFO]: -- SIL-testing::do331.jobs.SILCovAggregation
[INFO]: -- SIL-testing::do331.jobs.SILCoverageAnalysis
[INFO]: -- documentation::do331.jobs.GenerateDocumentation
```

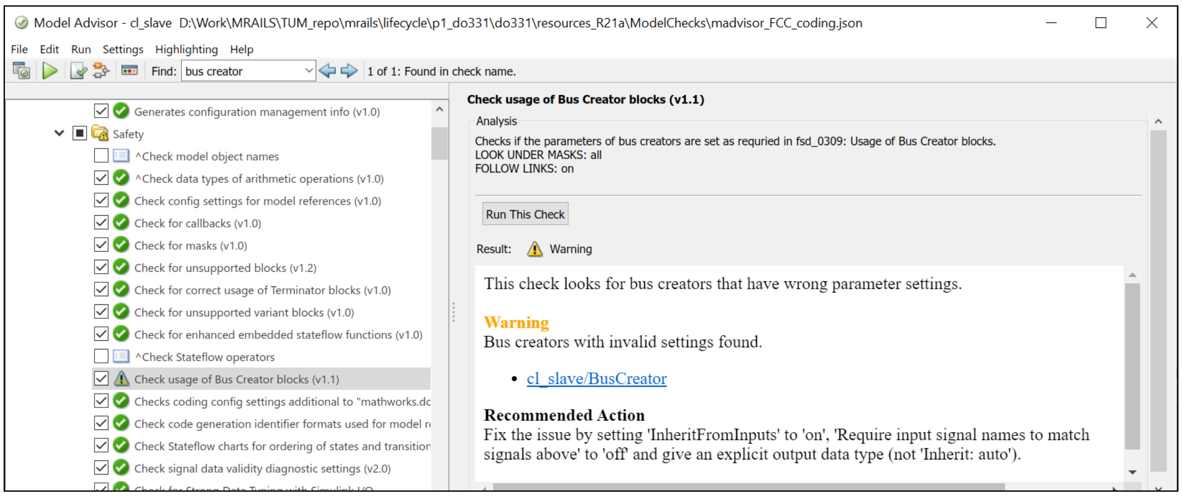
Fig. 9: Verification jobs provided by the build tool

4.1.1. Static Model Analysis

Static analysis of the model runs custom and MathWorks' checks on the model that are derived from the modeling guidelines and naming conventions [13]. Result of one custom check is shown in Fig. 10. The warning implies that the bus creators should inherit the signals name and avoid naming them again in the bus creator as this would lead to signal name mismatching. Similarly, all other warnings are checked and resolved to prevent complete rework at the end. The status report provides the required solution to solve warnings and the artifact is also traceable.

4.1.2. Design Error Detection

Design error detection job runs and accumulates the results from Design Verifier. Fig. 11 shows the build tool status window containing the results. The 'bc_slave_Bus' contains: config array and module-id parameter.


Model Advisor - cl_slave D:\Work\MRAILS\TUM_repo\mrails\lifecycle\p1_do331\do331\resources_R21a\ModelChecks\madvisor_FCC_coding.json
File Edit Run Settings Highlighting Help
Find: bus creator
1 of 1: Found in check name.
Run This Check

Generates configuration management info (v1.0)
Safety
^Check model object names
^Check data types of arithmetic operations (v1.0)
Check config settings for model references (v1.0)
Check for callbacks (v1.0)
Check for masks (v1.0)
Check for unsupported blocks (v1.2)
Check for correct usage of Terminator blocks (v1.0)
Check for unsupported variant blocks (v1.0)
Check for enhanced embedded stateflow functions (v1.0)
^Check Stateflow operators
Check usage of Bus Creator blocks (v1.1)
Checks coding config settings additional to "mathworks.dc
Check code generation identifier formats used for model r
Check Stateflow charts for ordering of states and transition
Check signal data validity diagnostic settings (v2.0)

Check usage of Bus Creator blocks (v1.1)

Analysis

Checks if the parameters of bus creators are set as requried in fsd_0309: Usage of Bus Creator blocks.
LOOK UNDER MASKS: all
FOLLOW LINKS: on

Result: Warning

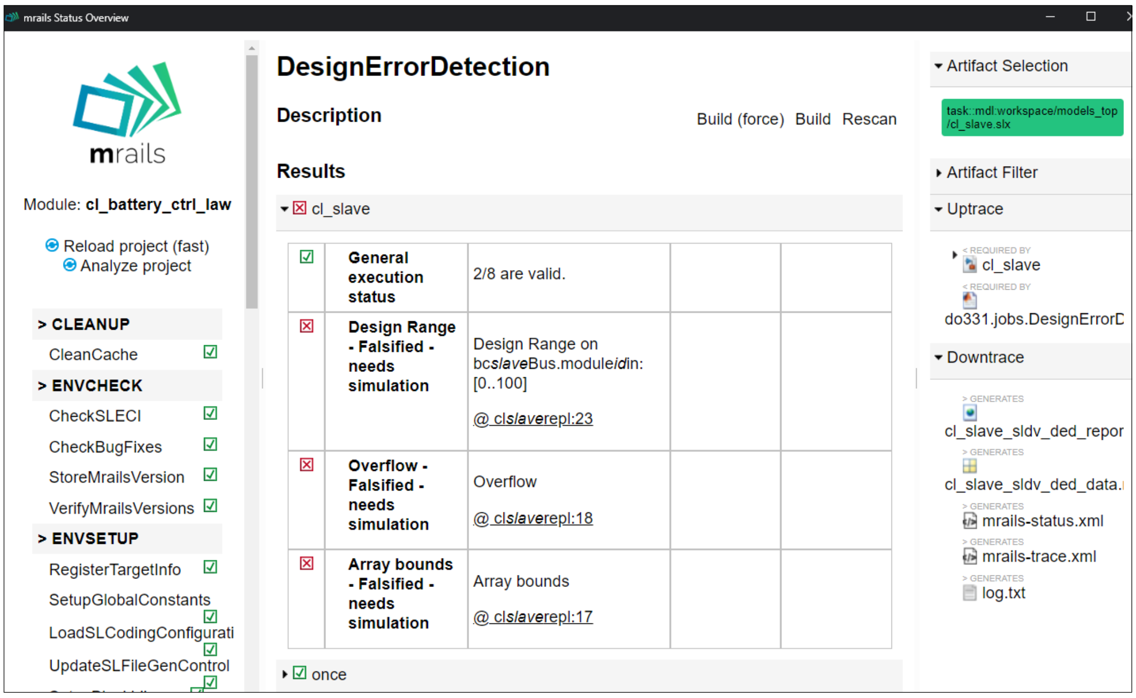
This check looks for bus creators that have wrong parameter settings.

Warning
Bus creators with invalid settings found.

- cl_slave/BusCreator

Recommended Action
Fix the issue by setting 'InheritFromInputs' to 'on', 'Require input signal names to match signals above' to 'off' and give an explicit output data type (not 'Inherit: auto').

Fig. 10: Part of Static Model Analysis Result


mrails Status Overview

mrails

Module: **cl_battery_ctrl_law**

Reload project (fast)
Analyze project

> CLEANUP
CleanCache ✓
> ENVCHECK
CheckSLECI ✓
CheckBugFixes ✓
StoreMrailsVersion ✓
VerifyMrailsVersions ✓
> ENVSETUP
RegisterTargetInfo ✓
SetupGlobalConstants ✓
LoadSLCodingConfigurati ✓
UpdateSLFileGenControl ✓

DesignErrorDetection

Description Build (force) Build Rescan

Results

▾ ☒ cl_slave

✓	**General execution status**	2/8 are valid.		
☒	**Design Range - Falsified - needs simulation**	Design Range on bcs*lave*Bus.module*id*in: [0..100] @ cl*slave*repl:23		
☒	**Overflow - Falsified - needs simulation**	Overflow @ cl*slave*repl:18		
☒	**Array bounds - Falsified - needs simulation**	Array bounds @ cl*slave*repl:17		

▸ ✓ once

▾ Artifact Selection
task::mdl:workspace/models_top/cl_slave.slx
▸ Artifact Filter
▾ Uptrace
< REQUIRED BY cl_slave
< REQUIRED BY do331.jobs.DesignErrorD
▾ Downtrace
> GENERATES cl_slave_sldv_ded_repor
> GENERATES cl_slave_sldv_ded_data.
> GENERATES mrails-status.xml
> GENERATES mrails-trace.xml
> GENERATES log.txt

Fig. 11: Design Error Detection Results

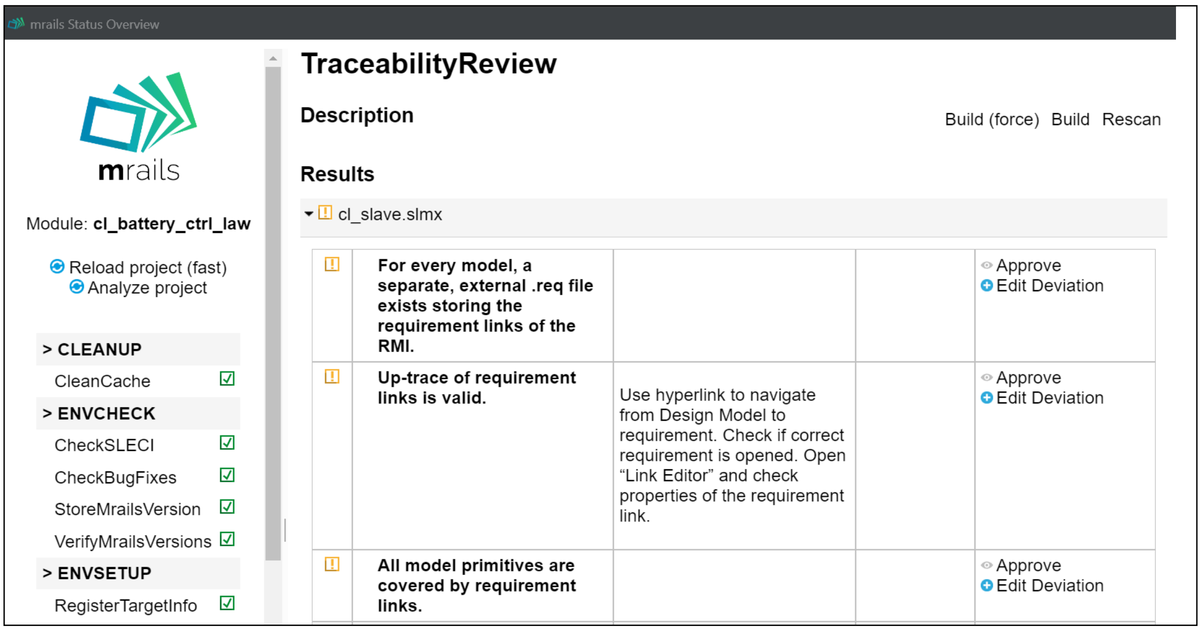
The module-id element has design range of [0 ... 100]. This is defined in the data dictionary during the interface creation. This implies that the value cannot exceed this range. However, the Design Verifier derives the ranges of all the signals using extreme input values and since the module id is increased by 1, for the maximum value of 100, the new module id will be incremented to 101 which exceeds the range. Similarly, when subtracting 1 from the minimum value 0, it creates an integer overflow which is the second error in the results and is also discussed in code proving section 4.3.2.

4.1.3. Traceability Review

Once the requirements are allocated to the models, traceability review can be performed. This is done manually by an engineer via the status report. Although many checks are performed automatically, few checks are difficult to automate, and hence manual review is required. The tool provides necessary checklist according to the guidelines which must be reviewed. Fig. 12 shows the Traceability Review section of the status report. Only few checklists are shown here due to content limitations. These checks are based on DO-331 (MB. A 4.1, 4.6) [7,9]. If the checklists are fulfilled, the reviewer must approve the checks and the report will be saved in the interface itself.

4.2. Simulation Testing

Simulation testing involves running the test cases in ‘normal’ mode. This verification task includes simulation


mrails Status Overview

mrails

Module: **cl_battery_ctrl_law**

Reload project (fast)
Analyze project

> CLEANUP
CleanCache ✓
> ENVCHECK
CheckSLECI ✓
CheckBugFixes ✓
StoreMrailsVersion ✓
VerifyMrailsVersions ✓
> ENVSETUP
RegisterTargetInfo ✓

TraceabilityReview

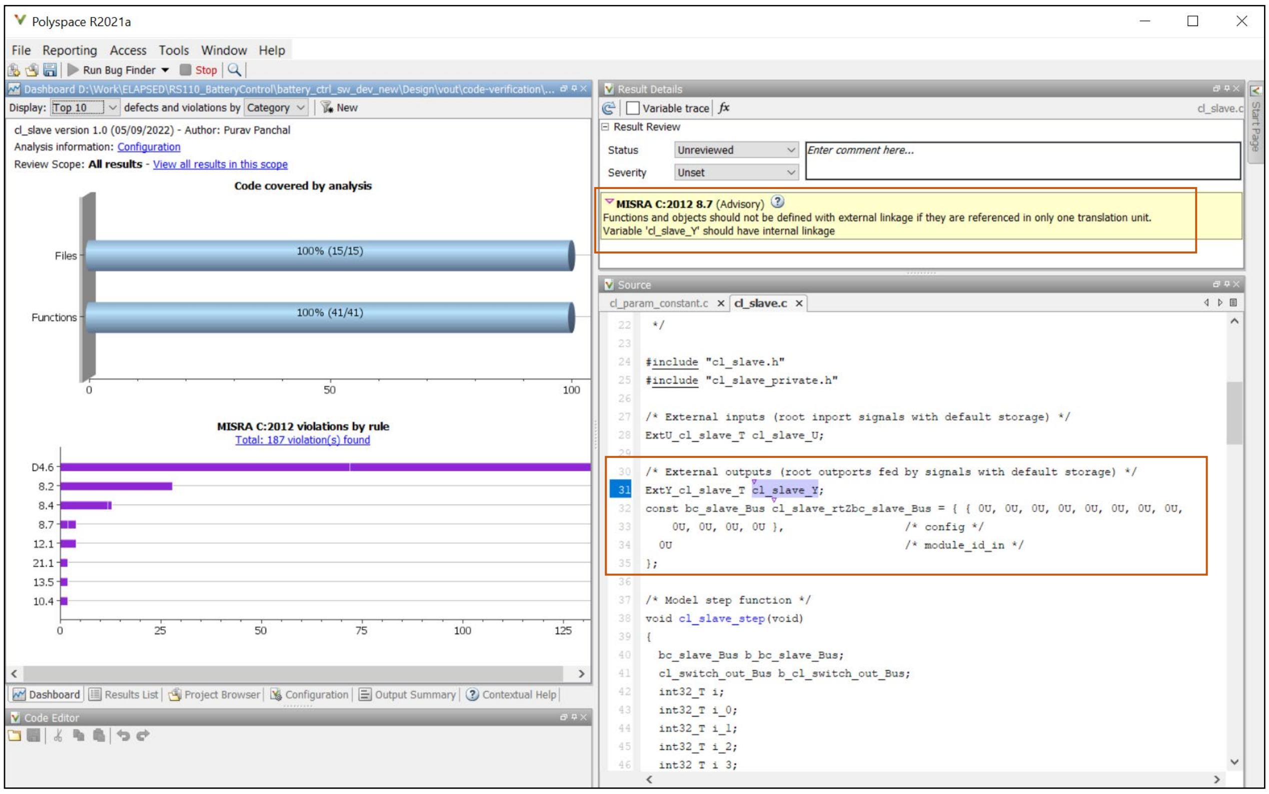
Description Build (force) Build Rescan

Results

▾ ! cl_slave.slmx

!	**For every model, a separate, external .req file exists storing the requirement links of the RMI.**			Approve / Edit Deviation
!	**Up-trace of requirement links is valid.**	Use hyperlink to navigate from Design Model to requirement. Check if correct requirement is opened. Open “Link Editor” and check properties of the requirement link.		Approve / Edit Deviation
!	**All model primitives are covered by requirement links.**			Approve / Edit Deviation

Fig. 12: Traceability Review Results


Polyspace R2021a
File Reporting Access Tools Window Help
Run Bug Finder
Stop
Dashboard D:\Work\ELAPSED\RS110_BatteryControl\battery_ctrl_sw_dev_new\Design\vout\code-verification\...
Display: Top 10 defects and violations by Category New
Code covered by analysis
Files 100% (15/15)
Functions 100% (41/41)
0 50 100
MISRA C:2012 violations by rule
Total: 187 violation(s) found
D4.6 8.2 8.4 8.7 12.1 21.1 13.5 10.4
0 25 50 75 100 125
Dashboard Results List Project Browser Configuration Output Summary Contextual Help
Code Editor
Result Details
Variable trace
Source
cl_param_constant.c cl_slave.c
Start Page

cl_slave version 1.0 (05/09/2022) - Author: Purav Panchal
Analysis information: Configuration
Review Scope: **All results** - View all results in this scope

Result Review
Status: Unreviewed
Severity: Unset
Enter comment here...

MISRA C:2012 8.7 (Advisory)
Functions and objects should not be defined with external linkage if they are referenced in only one translation unit.
Variable 'cl_slave_Y' should have internal linkage

```
 */

#include "cl_slave.h"
#include "cl_slave_private.h"

/* External inputs (root inport signals with default storage) */
ExtU_cl_slave_T cl_slave_U;

/* External outputs (root outports fed by signals with default storage) */
ExtY_cl_slave_T cl_slave_Y;
const bc_slave_Bus cl_slave_rtZbc_slave_Bus = { { 0U, 0U, 0U, 0U, 0U, 0U, 0U, 0U,
    0U, 0U, 0U, 0U },                   /* config */
  0U                                    /* module_id_in */
};

/* Model step function */
void cl_slave_step(void)
{
  bc_slave_Bus b_bc_slave_Bus;
  cl_switch_out_Bus b_cl_switch_out_Bus;
  int32_T i;
  int32_T i_0;
  int32_T i_1;
  int32_T i_2;
  int32_T i_3;
```

Fig. 13: Code Compliance Result - Polyspace Bug Finder

case execution and model coverage analysis. After all the tests are executed, aggregated model coverage is calculated. From the aggregated coverage, model coverage is extracted from the results by creating a coverage filter [5]. The build tool can also execute individual test cases or a set of test cases in simulation mode. Like the manual traceability review task shown in Fig. 12, simulation review is done using the checklist defined by the build tool based on DO-331 guidelines after collecting the model coverage. The result of this job is not in the scope of this paper.

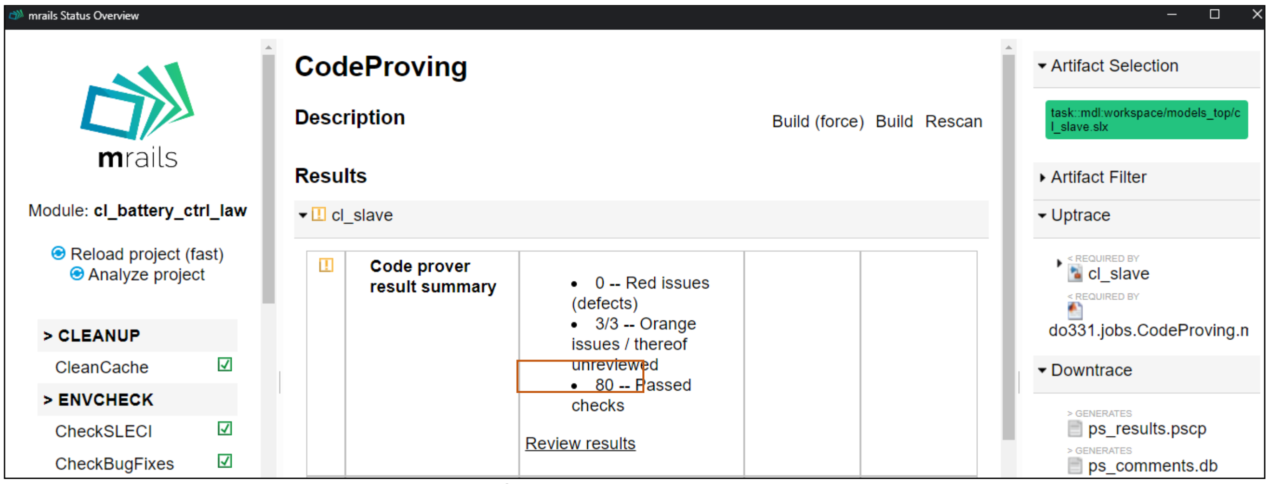
4.3. Code Verification

Code verification jobs include code inspection, checking code compliance, code defect analysis, code proving, and SIL testing. In this paper, code compliance and code proving results are discussed. Static code analysis helps in identifying possible run-time errors in source code, identify dead logic, division by zero and checks if the code meets the MISRA C 2012 compliance [19,18].

4.3.1. Code Compliance

Polyspace Bug Finder is used to check the code compliance with MISRA C 2012 guidelines. The build tool runs the Bug Finder with predefined configuration and all violations are collected. The result from Polyspace can be accessed via the status interface. Fig. 13 shows the result of code compliance check on the slave controller code.

The violation shown is related to MISRA C:2012 8.7 Advisory guideline [21], which requires the external function and objects that are referenced in only one translation unit should have internal linkage. In our case, the external input bus 'bc_slave_Bus' is only referenced in



Module: **cl_battery_ctrl_law**

CodeProving

Description

Results

cl_slave

Code prover result summary

- 0 -- Red issues (defects)
- 3/3 -- Orange issues / thereof unreviewed
- 80 -- Passed checks

Review results

task::mdl:workspace/models_top/cl_slave.slx

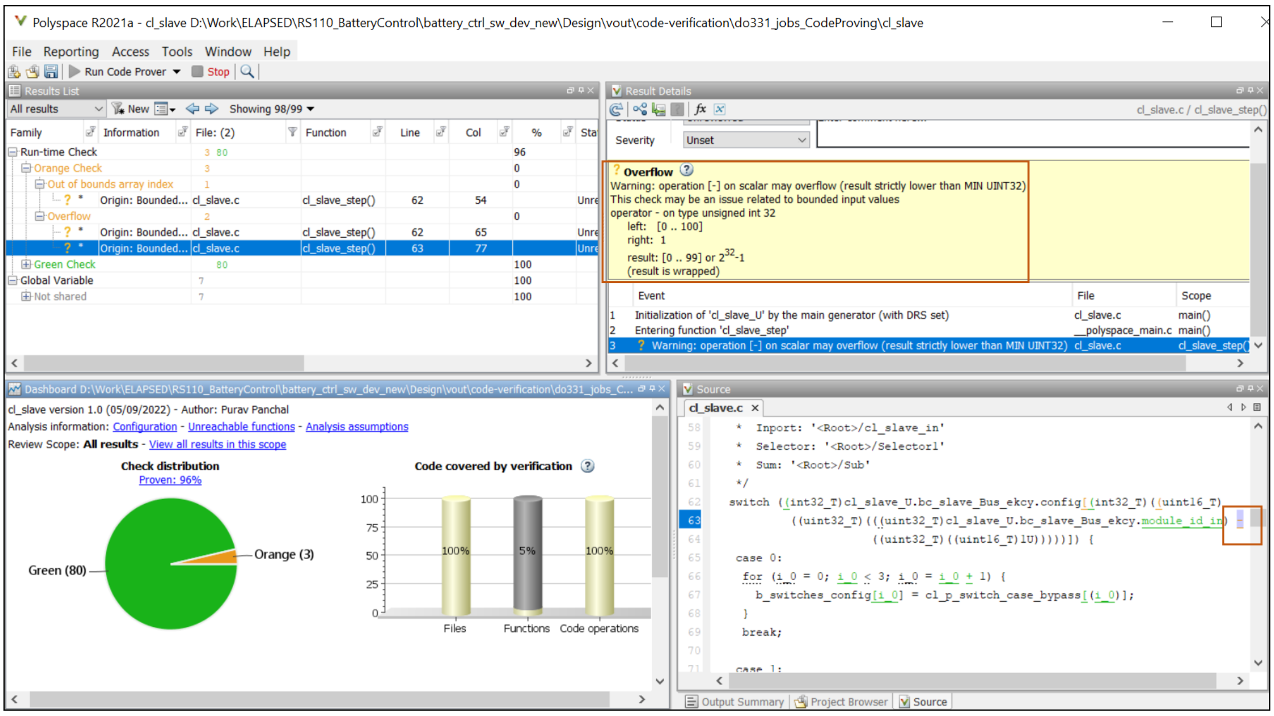
cl_slave

do331.jobs.CodeProving.n

ps_results.pscp

ps_comments.db

Fig. 14: Code Prover Result - Build Tool


Polyspace R2021a - cl_slave D:\Work\ELAPSED\RS110_BatteryControl\battery_ctrl_sw_dev_new\Design\vout\code-verification\do331_jobs_CodeProving\cl_slave
File Reporting Access Tools Window Help
Run Code Prover
Stop
Results List
All results
New
Showing 98/99
Result Details
cl_slave.c / cl_slave_step()
Severity
Unset
Dashboard D:\Work\ELAPSED\RS110_BatteryControl\battery_ctrl_sw_dev_new\Design\vout\code-verification\do331_jobs_C...
Check distribution
Proven: 96%
Green (80)
Orange (3)
Code covered by verification
100
75
50
25
0
100%
5%
100%
Files
Functions
Code operations
Source
cl_slave.c
Output Summary
Project Browser
Source

Family	Information	File: (2)	Function	Line	Col	%	Sta
Run-time Check		3 80				96	
Orange Check		3				0	
Out of bounds array index		1				0	
? *	Origin: Bounded...	cl_slave.c	cl_slave_step()	62	54		Unre
Overflow		2				0	
? *	Origin: Bounded...	cl_slave.c	cl_slave_step()	62	65		Unre
? *	Origin: Bounded...	cl_slave.c	cl_slave_step()	63	77		Unre
Green Check		80				100	
Global Variable		7				100	
Not shared		7				100	

```
? Overflow
Warning: operation [-] on scalar may overflow (result strictly lower than MIN UINT32)
This check may be an issue related to bounded input values
operator - on type unsigned int 32
    left:   [0 .. 100]
    right:  1
    result: [0 .. 99] or 2^32-1
    (result is wrapped)
```

	Event	File	Scope
1	Initialization of 'cl_slave_U' by the main generator (with DRS set)	cl_slave.c	main()
2	Entering function 'cl_slave_step'	__polyspace_main.c	main()
3	? Warning: operation [-] on scalar may overflow (result strictly lower than MIN UINT32)	cl_slave.c	cl_slave_step()

cl_slave version 1.0 (05/09/2022) - Author: Purav Panchal
Analysis information: Configuration - Unreachable functions - Analysis assumptions
Review Scope: **All results** - View all results in this scope

```
 *  Inport: '<Root>/cl_slave_in'
 *  Selector: '<Root>/Selector1'
 *  Sum: '<Root>/Sub'
 */
switch ((int32_T)cl_slave_U.bc_slave_Bus_ekcy.config[(int32_T)((uint16_T)
        ((uint32_T)(((uint32_T)cl_slave_U.bc_slave_Bus_ekcy.module_id_in) -
                ((uint32_T)((uint16_T)1U))))]) {
 case 0:
  for (i_0 = 0; i_0 < 3; i_0 = i_0 + 1) {
    b_switches_config[i_0] = cl_p_switch_case_bypass[(i_0)];
  }
  break;

 case 1:
```

Fig. 15: Code Prover Result - Polyspace Code Prover

one model and hence the violation occurs. However, this advisory guideline can be justified by the fact that this external bus 'bc_slave_Bus' will also be used during the master controller development and hence can have an external linkage.

4.3.2. Code Proving

Code proving helps in identifying run-time errors such as division by zero, integer overflow and unreachable code. The tool uses Polyspace Code Prover and accumulates the results which are easily accessible via the status report as shown in Fig. 14. The result states that the code has three orange issues/thereof unreviewed. Detailed result can be analyzed by clicking on 'Review Results' option in the status interface which opens the Polyspace Code Prover as shown in Fig. 15.

The warning which is marked in the result is related to the module-id element of the 'bc_slave_Bus'. This overflow was also detected in the Design Error Detection job and is caused by the incorrect range definition of the module-id element. The module-id element is of uint16 data type and has a range of [0 … 100]. The selector used in model has zero-based indexing mode due to the modeling guidelines supporting the fact that C language also uses zero based indexing. Due to this when the minimum value of module-id i.e., 0 is subtracted by 1, the subtraction block gives -1 which is out of the range of uint16 signal. Hence, an integer overflow is caused which was also detected during the design verification stage. To resolve this error, the limits of the signal elements are correctly defined, and additional switch is added in the model to check if the signal is exceeding the limits.

5. FUTURE WORK

As the slave controller is now developed and verified, the next goal is to test the controller in a hardware-in-the-loop (HIL) system. This will aid in verifying the failure conditions in real time. Initially a single slave controller will be tested, and then complete module stack will be tested. Along with this, development of master controller is also initiated in a similar fashion. The master controller will have all the necessary inputs from the battery which indicates the current operating condition like temperature and voltage. In future, a motor controller will also be developed using the build tool. Currently, a Continuous Integration platform is being deployed for all the development projects for ELAPSED [12].

On the build tool side, the tool is continuously improved by fixing bugs and resolving issues faced by developers. For example, an issue encountered recently was related to parameter handling during Code Proving job. As mentioned in [5], Polyspace Code Prover settings were changed with a hook to remove 'Parameter Constant' data ranges. However, the implemented function resulted in empty array elements and was fixed during this development. Likewise, other issues are also being resolved. Since the tool is used for various applications; we must make sure that new developments should not hinder the existing projects. To do so, a CI server is being setup for the development of the tool itself.

6. CONCLUSIONS

The paper has presented a model-based design application of a process-oriented build tool to develop and verify a battery slave controller for an BM3 module. The tool is introduced with its key advantages like automatic code generation, automatic toolchain setup, incremental verification and traceability, and handling of multiple modules. The process workflow of the build tool is described and later applied to develop the slave controller in MATLAB/Simulink. Important steps like requirement allocation and analyzing verification results are also presented. Polarion and SimPol tool were used to create and link requirements to Simulink models. Design and code verification include static model analysis, design error detection, traceability review, code compliance, and code proving. Easy creation and accessibility of the results is possible with the build tool and is realized in this paper.

This research is funded by dtec.bw – Digitization and Technology Research Center of the Bundeswehr [12].

REFERENCES

[1] Airport-Technology, “Ethiopian Airlines crash: what’s happened in the last two years?,” URL: https://www.airport-technology.com/analysis/ethiopian-airlines-crash-what-happened-last-two-years.

[2] Wikipedia, “Korean Air Flight 801 Crash,” URL: https://en.wikipedia.org/wiki/Korean_Air_Flight_801.

[3] TechTimes, “Toyota Recalls 112,500 Vehicles Due To Power Steering And Software Issues,” URL: https://www.techtimes.com/articles/39149/20150312/.

[4] Cleland-Huang, J., Agrawal, A., Vierhauser, M., and Mayr-Dorn, C., “Visualizing Change in Agile Safety-Critical Systems,” *IEEE Software*, Vol. 38, No. 3, 1 Jan. 2021, pp. 43–51. doi: 10.1109/MS.2020.3000104.

[5] Markus Tobias Hochstrasser, “Modular model-based development of safety-critical flight control software,” PhD Thesis, *Technischen Universität München,* Munich, Germany, 12 Jun. 2020.

[6] Hochstrasser, M., Myschik, S., and Holzapfel, F., “A Process-oriented Build Tool for Safety-critical Model-based Software Development,” *Proceedings of the 6th International Conference on Model-Driven Engineering and Software Development,* SCITEPRESS - Science and Technology Publications, 1 Jan. 2018, pp. 191–202.

[7] Hochstrasser, M., Myschik, S., and Holzapfel, F., “Application of a Process-Oriented Build Tool for Flight Controller Development Along a DO-178C/DO-331 Process,” *Model-Driven Engineering and Software Development,* edited by S. Hammoudi, L. F. Pires and B. Selic, Springer International Publishing, Cham, 1 Jan. 2019, pp. 380–405.

[8] RTCA, “DO-178C - Software Considerations in Airborne Systems and Equipment Certification,” RTCA, Incorporated, 1 Jan. 2011.

[9] RTCA, “DO-331 - Model-Based Development and Verification Supplement to DO-178C and DO-278A,” RTCA, Incorporated, 1 Jan. 2011.

[10] Manuel Kuder, Julian Schneider, Anton Kersten, Torbjörn Thiringer, Richard Eckerle, Thomas Weyh, “Battery Modular Multilevel Management (BM3) Converter applied at Battery Cell Level for Electric Vehicles and Energy Storages,” 1 Jan. 2020.

[11] Sorokina, N., Estaller, J., Kersten, A., Buberger, J., Kuder, M., et al., “Inverter and Battery Drive Cycle Efficiency Comparisons of Multilevel and Two-Level Traction Inverters for Battery Electric Vehicles,” *2021 IEEE International Conference on Environment and Electrical Engineering and 2021 IEEE Industrial and Commercial Power Systems Europe (EEEIC / I&CPS Europe),* IEEE, 1 Jan. 2021, pp. 1–8.

[12] dtec.bw, “Electric Aircraft Propulsion – die Zukunft der Flugzeugantriebe,” URL: https://dtecbw.de/home/forschung/unibw-m/projekt-elapsed.

[13] Dmitriev, K., Zafar, S. A., Schmiechen, K., Lai, Y., Saleab, M., et al., “A Lean and Highly-automated Model-Based Software Development Process Based on DO-178C/DO-331,” *2020 AIAA/IEEE 39th Digital Avionics Systems Conference (DASC),* IEEE, 1 Jan. 2020, pp. 1–10.

[14] Broy, M., Kirstan, S., Krcmar, H., and Schätz, B., “What is the Benefit of a Model-Based Design of Embedded Software Systems in the Car Industry?,” *Emerging Technologies for the Evolution and Maintenance of Software Models,* edited by J. Rech and C. Bunse, IGI Global, 1 Jan. 2012, pp. 343–369.

[15] Panchal, P., Myschik, S., Dmitriev, K., Bhardwaj, P., and Holzapfel, F. (eds.), *Handling Complex System Architectures with a DO-178C/DO-331 Process-Oriented Build Tool,* 2022, 1 Jan. 2022.

[16] Siemens, Polarion PLM Automation, https://polarion.plm.automation.siemens.com/.

[17] FSD, SimPol - Simulink® – Polarion® Connector, https://www.fsd.lrg.tum.de/software/simpol/.

[18] Panchal, P., Myschik, S., Dmitriev, K., and Holzapfel, F., “Application of a Process-Oriented Build Tool to an INDI-Based Flight Control Algorithm,” *AIAA AVIATION 2022 Forum,* American Institute of Aeronautics and Astronautics, Reston, Virginia, 1 Jan. 2022.

[19] MathWorks, Polyspace, https://www.mathworks.com/products/polyspace.html.

[20] Grupp Wolfgang, Hoegerl Tobias, Wiedenmann Andreas, Estaller Julian, Sorokina Nina, et al., “Investigation of Different Driver Topologies for Application in Modular Multilevel Systems,” *PCIM Europe 2022; International Exhibition and Conference for Power Electronics, Intelligent Motion, Renewable Energy and Energy Management*, 1 Jan. 2022, pp. 1–9.

[21] MISRA, “MISRA C:2012 Amendment 2,”.