DOI: 10.13009/EUCASS2023-953
Aerospace Europe Conference 2023 – 10TH EUCASS – 9TH CEAS

Model-Based Development of a Library with Standard Functions for Safety-Critical Flight Control Laws

Dmitry Chernetsov†, Bryan Laabs*, Ralph Paul** and Robert Luckner**
* Technische Universität Berlin, Marchstrasse 12-14, 10587 Berlin
** Leichtwerk AG, Hermann-Blenk-Str. 38, 38108 Braunschweig
dimitry.chernetsov@tu-berlin.de - bryan.laabs@tu-berlin.de - ralph.paul@leichtwerk.de - robert.luckner@tu-berlin.de
† Corresponding Author

Abstract

The establishment of a model element library enhances the efficiency of safety-critical model-based software development and requires specific deliberation. TU Berlin established a model-based software development process that follows RTCA DO-178C/DO-331 guidance where the individual advantages of Simulink and SCADE environments are used. The paper emphasises the functional equivalence between Simulink and SCADE models by specification and design of basic library modules common to both environments. The library modules are tested in SCADE, and Simulink on a Host PC, and the target platform. The results show that the functional and numerical deviations in different environments are within expected tolerances and are acceptable for flight control applications.

1. Introduction

Flight control laws are an essential part of complex and safety-critical flight control systems that have to be developed in compliance with certification specifications (e.g., CS-25 [1]) and the recommended processes defined in SAE 4754A [2] and RTCA DO-178C [3]. The design of flight control algorithms and their implementation in software requires appropriate tools, a compliant development process, a team with interdisciplinary knowledge, and a high level of expertise in software development and control engineering domains. Model-Based Development (MBD) improves the efficiency of the development of complex and safety-critical flight control laws [4]. The graphical representation of software in block diagram format is commonly used in the flight control domain. The advantages are:

- Block diagrams are standard in engineering, and they allow the development of the functions to take place at a higher level of abstraction than equations,
- Design models that are a representation of block diagrams in a formal, computer-readable, and executable format providing a universal communication mean between different stakeholders,
- Software can be generated from the design models automatically using reliable code generators,
- Verification activities in the software life cycle can start very early, as the design models are executable for dynamic testing and well-suited for model review.

These facts reduce the project risks and the time-to-market as well as save personal resources [5].
MATLAB/Simulink® and SCADE Suite® are integrated development environments (IDEs) used for MBD in the aerospace domain. MATLAB/Simulink is widely used in control engineering, simulation and model-based software development with a large number of users in teaching, research and industry. However, the use of MathWorks Embedded Coder for code generation requires a qualification of additional verification tools according to RTCA DO-330 [6] ensuring the absence of errors after code generation. On the other hand, Ansys SCADE Suite IDE is a tool suite for developing safety-critical embedded software. It uses the high-level language Scade[1] that is formally defined, declarative and deterministic [7] allowing an error-free transformation from Scade to C code. The SCADE Suite KCG code generator is pre-qualified according to RTCA DO-330.
In the project CERTT-FBW[2], TU Berlin demonstrated a development process for flight control software, in which flight control laws are designed in Simulink, the model is automatically translated into Scade, and the code is generated by SCADE Suite KCG in a qualified manner. This approach combines the advantages of both tools [8] and is improved in the projects FCL-Methods and IBAS[3]. In both projects the flight control software for the Leichtwerk's "StratoStreamer" high-altitude pseudo satellite (HAPS) aircraft [9] (see Figure 1-1) is developed. In the context of this research, a model element library FCLib was implemented. The development process strictly follows the RTCA DO-178C and RTCA DO-331 [10], for Design Assurance Level C (DAL-C) software. The library contains essential modules that are used in flight control software.

[1] Note, "Scade" is the modelling language and "SCADE" the name of ANSYS's tool suite and its components.
[2] The research project CERTT-FBW is funded by the Federal Ministry of Economic Affairs and Energy (BMWI) in the National Aerospace Research Program (LUFO V).
[3] See Acknowledgement.

DOI: 10.13009/EUCASS2023-953

Figure 1-1: “StratoStreamer” HAPS aircraft [11]

Simulink and SCADE IDEs offer an extensive set of library elements of different complexity and for different purposes. The elements are often not sufficiently documented[12], without details of the implementation or traceability to requirements, description of the used design methods and test results. Many Simulink blocks have no equivalent SCADE blocks [12] , which prohibits their usage in the presented development process. The SCADE Suite provides a set of library elements described only on a functional level without certification data. Several SCADE Suite library modules include external custom code or reference other C-Code libraries like math.h [13], for which the functional correctness is assumed but not formally proven.

Reference [14] describes the workflow of how to reach reproducible numerical results in Simulink and on target hardware with already developed mathematical libraries that were qualified according to European Space Agency regulations. The same C-code Mathematical Library for Flight Software (MLFS)[4] is used in the native Simulink simulation, in stand-alone applications on a standard host PC and on the target. Additionally, a set of modelling and compilation guidelines is described to achieve reproducibility within all used environments.

Reference [15] describes a purely model-based development of elementary mathematics functions in Simulink. The main objectives are the improvement of the worst-case execution time on target and the demonstration of formal correctness of developed functions in terms of desired precision.

Both studies mentioned above only focus on elementary math functions which is not sufficient for the considered development process. The use of external code in [14] lessens the advantage of qualified code generation in SCADE due to leaving the qualified scope of the Scade language. In addition, the simultaneous use of manually written code and MBD unnecessarily complicates the development process concerning DO-178C. The focus of reference [15] lies on tool qualification aspects whereas the certification aspects regarding DO-331 are not addressed.

The FCLib comprises mathematical functions and typical elements from control engineering like integrators, limiters, signal filters, look-up tables, digital logic, and navigation equations. The key objective of the development approach is to ensure that Simulink and SCADE models behave identically. A further key objective is the establishment of a documented and verified library that supports small teams with limited personnel resources to develop safety-critical flight control software.

This paper describes the development approach for the FCLib and explains essential process steps using examples of FCLib module design and verification on a host PC and target. In Section 2 the development process, Simulink to SCADE mapping rules, and translation procedure are introduced. In Section 3 the design steps are shown in more detail. Section 4 discusses the verification aspects of the library elements.

2. Model-Based Software Development Process

RTCA DO-331 a supplement to RTCA DO-178C outlines additional aspects of MBD for the development of safety-critical software for aviation systems. In this work, DO-331 is used to establish a model-based software development process for the development of Automatic Flight Control Laws software (AFCL SW) and Direct Flight Control Laws software (DFCL SW) for a full-authority Automatic Flight Control System (AFCS) of the “StratoStreamer” HAPS aircraft in the certified category. Both applications are integrated as segregated partitions into a flexible computing platform that uses the Integrated Modular Avionic (IMA) technology of the Aviotech company. The deployment of this IMA platform was already successfully demonstrated in [16] and [17].

The AFCL SW provides the following functions:

- Input signal pre-processing and signal filtering,
- Calculation of flight envelope limits and operational limits,
- Calculation of mode availability, flight state and mode of operation,
- Navigation and flight path calculation from waypoint commands and mission demands,
- Closed-loop control for longitudinal, vertical, and lateral guidance of the aircraft,

[4] MLFS is available on https://essr.esa.int/project/mlfs-mathematical-library-for-flight-software

DOI: 10.13009/EUCASS2023-953

- Calculation of individual commands for the various aircraft control elements,
- Automatic take-off and landing function.

The DFCL SW provides the following functions:

- Pre-processing of cockpit input signals,
- Functions for manual flight control,
- Function for system parameter identification during flight test,
- Switch of control elements command between AFCL SW and DFCL SW commands.

The DFCL SW is used for piloted flight tests. It is developed according to the same standards as the AFCL SW.
Figure 2-1 shows the scope of the Software Life Cycle (SLC) and data generated during software development. The focus is on establishment of methods and procedures for development, verification and configuration management, which are documented in:

- Plan for Software Aspects of Certification (PSAC),
- Software Development Plan (SDP),
- Software Verification Plan (SVP),
- Software Configuration Management Plan (SCMP).

These plans fulfil the DO-178C objectives for the planning process. Additionally, based on our experience in previous projects [8] and after several refinements through the SLC, a comprehensive software modelling and design standard (SMS/SDS) has been developed. The model element library FCLib is developed using the same plans and standards as the AFCL SW and DFCL SW.
Any FCLib baseline contains:

- FCLib as model element library in Simulink and in SCADE,
- Software Design Description (SDD),
- Software Verification Cases and Procedures (SVCP),
- Software Verification Results (SVR),
- Software Configuration Index (SCI).

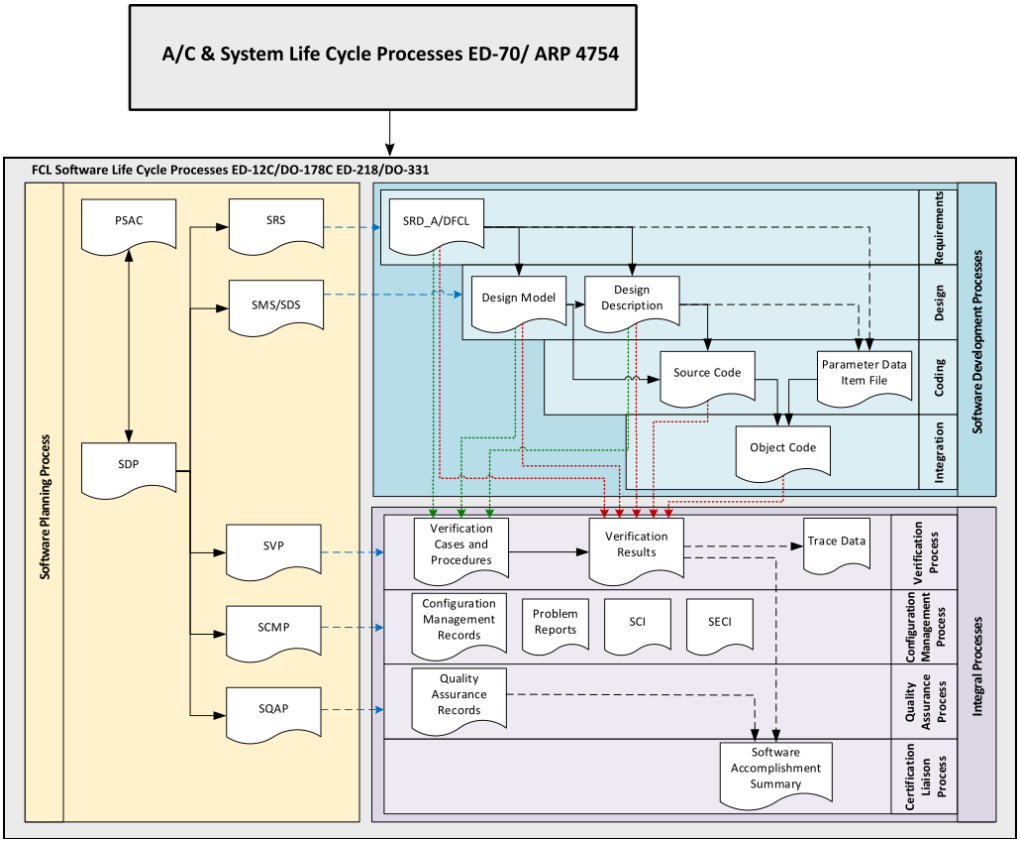

Figure 2-1: Software Life Cycle Process and Data according to DO-178C/DO-331

DOI: 10.13009/EUCASS2023-953

2.1 Software Development Environment

Table 2-1 summarizes definitions that are of particular importance.

Table 2-1: Term Definitions

Term	Definition
Design Model[1]	A model that defines any software design such as low-level requirements (LLR), software architecture, algorithms, component internal structures, data flow and/or control flow. A model used to generate Source Code is a Design Model. The Design Model is the SCADE Suite Model.
Prototype Model	A model that defines any software design such as low-level requirements, software architecture, algorithms, component internal structures, data flow and/or control flow. A model is called Prototype Model if no Source Code is generated from it. The Prototype Model is a Simulink Model.
Library Module	A Model Element that is included in a Model Element Library.
Model Element[1]	A unit from which a model is constructed.
Model Element Library[1]	A collection of model elements used as a baseline to construct a model. A model may or may not be developed using model element libraries.

[1] Definitions are adopted from [10], ANNEX MB.B. Glossary.

Figure 2-2 illustrates the development environment and tools used in the software development process described in the previous section. All development activities are performed on a Host PC and on the Processor-in-the-Loop (PiL) test rig that provides the environment and all tools described below.

The requirements for AFCL SW and DFCL SW are defined in textual form and managed with a requirement management tool. A special characteristic of our process is that the requirements, from which the *Prototype and Design Models* are developed, are on the system level. This approach implements the model usage example 5 of [10].

The executable Prototype Model is developed utilising Simulink. The Prototype Model is used to develop the Flight Control Functions (FCF) in closed-loop simulations with a high-fidelity Flight Mechanical Model (FMM) of the aircraft. The Prototype Model effectively represents both software architecture and software low-level requirements (LLR) and is a part of the system development process. However, it is essential to emphasise that the Prototype Model is not a Design Model, as neither source code nor executable code for the target is generated from this model.

The Prototype Model undergoes a comprehensive examination utilising the Model Examiner® (MXAM) tool by Model Engineering Solutions to ensure the translation readiness into Scade language and design standard compliance. This analysis verifies the model's adherence to the design rules, checks the complexity and structure and then the tool produces detailed reports that document findings. These reports serve as evidence for the compliance of the Prototype Model with the established design standards.

Afterwards, the Prototype Model is automatically translated into Scade using the SCADE Simulink Importer and the customised S2S (Simulink to Scade) tool that provides additional features. The S2S tool enables a modular model translation and enables the transfer of metadata (e.g., requirement traceability data). At this stage, the FCLib ensures the functional equivalence of the two models by applying mapping rules between FCLib modules in Simulink and SCADE. Testing the Design Model against the requirements utilised in the development of the Prototype Model makes a tool qualification for model translation unnecessary.

After the translation of the Prototype Model from Simulink to SCADE, it becomes the formal software Design Model. The Design Model encompasses the software architecture as well as the LLR. The LLR are represented by the safe state machine syntax and equations consisting of primitive operators of the Scade language and are fully documented in [18]. During the testing of the Design Model and by the Design Model Peer Review, all necessary verification evidence is generated to show compliance with the objectives of DO-178C/DO-331.

A qualified SCADE toolchain is employed for testing, report generation, and code generation to reach compliance with DO-178C/DO-331 objectives for the Design Model and the resulting source code:

1. SCADE Test Environment for Host is used for conducting Design Model simulations, demonstrating compliance of the Design Model with requirements.
2. SCADE Test Model Coverage (MTC) performs comprehensive analysis and reporting of coverage achieved through requirement-based testing at both the Design Model and source code levels.
3. SCADE Life Cycle Reporter guarantees the integrity and coherence of the generated Design Model Report (DMR), ensuring its alignment with the Design Model. The DMR is an important annex to the SSD document, and its assessment takes place during the Design Model Peer Review.
4. SCADE Suite KCG 6.6 Code Generator is responsible for the generation of source code from the Design Model.

DOI: 10.13009/EUCASS2023-953

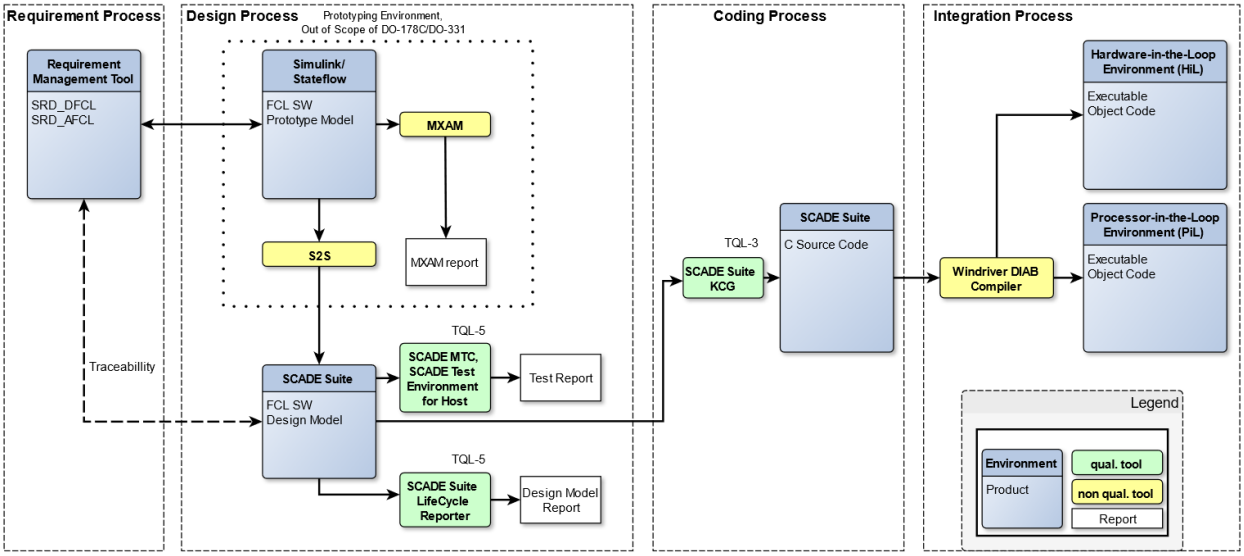

Figure 2-2: Software Development Environment

The system integration process starts with the software pre-integration on a Processor-in-the-Loop (PiL) test rig, which represents a single lane of the Core Processing Module (CPM) of the IMA platform. The executable object code (EOC) is generated with the WindRiver® DIAB Compiler and is then loaded into the PiL environment. Inside the PiL environment, the Design Model tests are replicated with attention turned to memory and timing aspects.
The EOC is delivered to Leichtwerk AG for Hardware in the Loop (HiL) tests after the successful completion of PiL testing. The HiL test rig at Leichtwerk AG is a full replica of the AFCS IMA platform and encloses all required functions.

2.2 Mapping of Primitive Operators between Scade and Simulink

The permissible extent of operators available for library design in Simulink is constrained by the functionally equivalent elements of the Scade language. In this work, the Scade 6.6 language version is used for software design within the SCADE IDE and its basic features are described here.
Scade is a synchronous language, using the synchronous hypothesis. This hypothesis states that model outputs are generated instantaneously in response to inputs. All communications and computations within a model are considered to be instantaneous [19]. This approach is well suited for real-time embedded systems as it guarantees deterministic operation within finite time and memory frame [7].
Scade 6.6 as a formal language incorporates mathematical formalism, which enables automatic analysis ensuring the correctness and absence of ambiguities in Scade models [7]. Formal analysis is exerted inside SCADE Suite Checker as an integral component of the SCADE suite for examination of semantic and syntactic accuracy.
In SCADE IDE the Scade models have textual and graphical representations, as illustrated by the example of the FCLib HOLD module in Figure 2-3 and Figure 2-4. This module retains the output *y_out* at its value of the previous execution cycle *yk1* as long as the input condition *b1s* remains *false*. Within the Scade language definition, the HOLD module is considered a User-Defined Operator (UDO), which is a hierarchically structured Scade model. Each UDO can be depicted as a node or function containing equations or state machines. Nodes represent operators with an internal state, necessitating the storage of past values in memory. In contrast to nodes, functions are operators without internal states and memory. In the given example, the UDOs Initial Condition (IC) and Switch are visually represented by yellow rectangles, with IC being a node and Switch being a function. The blue square denoting "Followed By" (FBY) is a primitive operator belonging to the fundamental expressions of the Scade 6.6 language and is a unit delay with an initialisation value. The control and data flow of the Scade model is expressed through equations, which combine various primitive operators.
All primitive operators are elements of the formal Scade 6.6 language and belong to the pre-qualified scope of the code generator SCADE Suite KCG. Primitive operators can be categorised as outlined in Table 2-2 and include essential elements used in algorithms such as simple arithmetic (plus, minus, division), logical (lower than, AND), bit-wise operations (bit-wise OR, bit-wise NOT), structure and array operations (select array element, concatenation), temporals (Init, FBY) and flow control (If..Then..Else). The higher order primitives include activate condition, Map and Fold iterators to apply functions or nodes to array items. Furthermore, Table 2-2 provides examples of the mapping between Scade primitives and elementary Simulink blocks. The full scope of allowed primitive operators and their mapping is defined by the Software Design Standard (SDS) and Software Design Description (SDD) of FCLIb.

DOI: 10.13009/EUCASS2023-953

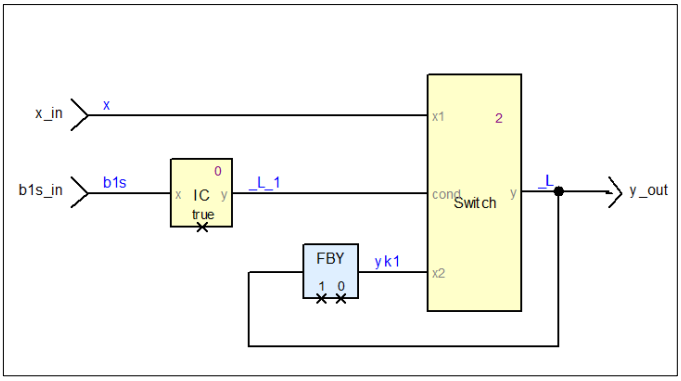

Figure 2-3: Graphical Scade Representation

```
package FCLib_HOLD
  node HOLD(x_in: float64; b1res_in: bool)
  returns (y_out: float64)
  var
    yk1: float64;
    x: float64;
    _L_1: bool;
    b1res: bool;
    _L: float64;
  let
    x = x_in;
    b1res = b1res_in;
    _L_1 = #0 IC(b1res, true);
    yk1 = #1 fby (_L; 1; 0._f64);
    _L = #2 Switch(x, _L_1, yk1);
    y_out = _L;
  tel;

end;
```

Figure 2-4: Textual Scade Representation

Table 2-2: Primitive Operators Examples and Mapping from Simulink to SCADE

Category	Scade Primitive Block	Simulink Block
Arithmetic	Plus	
Bitwise arithmetic	Bitwise AND	Bitwise AND
Relational	Equal	==
Boolean	AND	
Structure/Array	Data Array	Mux
Temporal	Init	[sig_ini]
Flow switches	If..Then..Else	
Higher Order	Activate	

2.3 Simulink Model Translation with S2S using the FCLib

The process of translation involves the conversion of Simulink models into SCADE Suite using the proprietary command line tool S2S, implemented in Python. S2S automates the import activities of the SCADE Suite Simulink Importer tool. Figure 2-5 illustrates the model translation process, which consists of two main tasks :

1. Exporting a Prototype Model from Simulink/Stateflow into JavaScript Object Notation (JSON) files.
2. Importing the JSON files into the Design Model in SCADE with configurable import options.

Moreover, S2S enables the reimport of the Prototype Model sub-modules. To achieve this, S2S analyses the model hierarchy of an existing Design Model and invokes the Simulink Importer with the appropriate configuration files (.cfg-file) for reimporting. The .cfg-file contains rules for reusing the imported operators of the created Scade models. The .cfg-files are either automatically generated during the translation process or manually created and customised by the user for specific mapping rules. Customised .cfg-file (*lib_map.cfg*) is used for mapping FCLib modules between the Prototype and Design Model. Consequently, the imported Scade model does not include translations of the FCLib library models from Simulink but rather instances of the library modules from the FCLib Scade models. Furthermore, the mapping rules, which gover the translation of elementary Simulink blocks to Scade primitives are defined in the *op_map.cfg* file.

The functional equivalence between the Prototype and Design Model is ensured by strict mapping rules between Model Elements of the FCLib in Simulink and Scade, as well as compliance of the Prototype Model with design standards and subsequent testing.

DOI: 10.13009/EUCASS2023-953

Additionally, the S2S tool adds annotations to the generated Design Model through the SCADE Python API to preserve data that is not retained by the Simulink Importer itself. An annotation is a note associated with an object in SCADE IDE and is used to add extra information to the model. This information includes requirements traceability data, design decisions, editorial comments as well as information from the Interface Control Document, and Git version control information for configuration management.
Upon successful translation, the software designer uses the SCADE Checker tool to validate the semantic and syntactic correctness of the Design Model. The SCADE Checker tool is qualified and belongs to the scope of the pre-qualified code generator SCADE Suite KCG.

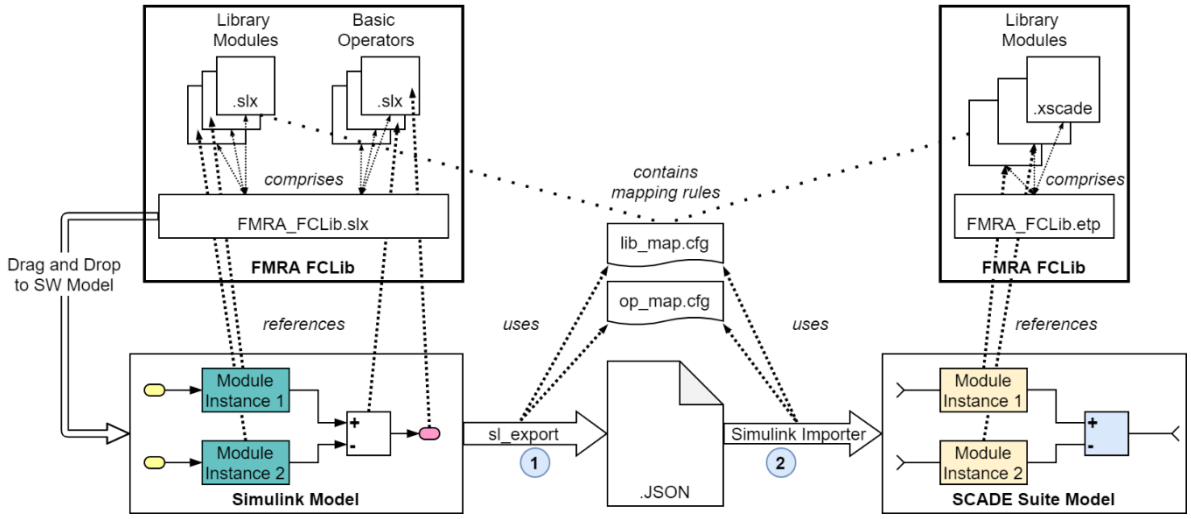

Figure 2-5: Translation procedure for Simulink models using the FCLib

3. Library Specification and Design

The development involves three distinct steps for specification and design (1) the elementary blocks of Simulink are defined, which can be effectively mapped to suitable Scade primitive operators, (2) the functional scope of the library is determined, (3) each library module is specified, implemented, and documented in the Software Design Document (SDD). Automatic design model report is generated for all library modules in the SCADE IDE.

3.1 Functional Scope of the Library

The FCLib comprises 48 library modules, whereas the supplementary math library (MATHLib) comprises 44 library modules, which are fully integrated into the FCLib within Simulink and SCADE Suite. The library is categorised into three types:

- time-independent modules,
- time-dependent modules,
- MATHLib modules.

Time-independent modules encompass functions where the output is fully determined by the inputs within a single execution cycle. On the other hand, time-dependent modules, rely on both the inputs and one or more stored values (internal states) from previous execution cycles. Consequently, time-dependent library modules necessitate memory access and initialisation. MATHLib modules are exclusively time-independent and are implemented as functions.
In addition to temporal considerations, the categorisation of FCLib modules based on their functional purpose within flight control laws is depicted in Table 3-1.
The FCLib modules encompass limitations functions to maintain predefined or safe value ranges, such as envelope limits or control surface deflections. Look-Up tables are commonly employed for controller gain scheduling. The desired controller structure can be constructed using integrators and derivatives. Discrete filters, such as first-order low-pass filters, can be used to modify or amplify specific characteristics of the signals. Digital logic modules incorporate elements necessary for logical decision-making. Examples include signal edge indication, monostable or flip-flops. Navigation equations can be employed to calculate the flight path based on commanded waypoints. These equations use trigonometric double-precision mathematical functions (MATHLib) to achieve the calculation precision needed for navigation algorithms. The MATHLib components encompass functions that assess specific attributes (e.g., presence of NaNs or INFs) of floating-point numbers represented in their binary form, aiming to simplify and accelerate the computation of mathematical functions. These elementary operators and approximation algorithms are outlined in [20].

DOI: 10.13009/EUCASS2023-953

Table 3-1: FCLib Module Categories and Examples

FCLib Category	FCLib Module Examples	ID
Time-independent		
Limitation & protections	Limiter Dead Space	LIM DEAD
Look-up Tables	1-D Look-Up Table 2-D Look-Up Table	LINT1 LINT2
Navigation equations	The course of an orthodrome Target coordinates calculation	NAVCRS NAVTGT
Math operators	Absolute Value Signum	ABS SIGN
Time-dependent		
Discrete Filter and Transfer Functions	First-order low-pass filter Integrator	FILTLP1a INT
Data Consolidation	synchronisation of two signals	SYNC
Digital Logic	Flip-Flop with reset priority Conditional memory Confirm trailing (falling) edge	FLIPR HOLD CONFT
MATHLib		
floating point numbers	Not a Number Check 64 bit Subnormal Check 32 bit	f64NAN f32SUBN
math functions	single sine double square root	SINF SQRT

3.2 Specification, Implementation and Documentation

DO-331 requires the definition of library-specific requirements like interface conventions, timing aspects, operational conditions, and functional aspects. The Software Design Document (SDD) serves the dual purpose of both specification and design documentation and is an important item within the FCLib Life Cycle. It encompasses the essential information regarding the usage rules, and general properties of the FCLib such as model execution basics, general library parameters (e.g., used sample time), data types, numerical limits, and primitive operator mapping rules. When a specific function is required in multiple instances within the AFCL or DFCL application, it is included as a library module in the FCLib and described in the SDD. The functionality, interface definition, and description of each module correspond to the software High-Level Requirements (HLR) in terms of RTCA DO-178C. As the FCLib consists of library modules that implement elementary functions, these specifications directly serve as documentation for the library modules within the SDD. Therefore, there is no separate establishment of a Software Requirement Document (SRD). The specification is independent of the used modelling environment. With the assistance of pseudocode and the requirements, the implementation can be carried out in various environments directly, including Simulink, SCADE, or other programming languages. This characteristic provides the flexibility to adapt the specification to different modelling environments, allowing for straightforward implementation in the desired environment. As a result, it facilitates the convenient reuse and expansion of the FCLib in various projects, further enhancing its versatility and applicability.

Each library module is comprehensively described in a distinct section, presenting all necessary information. The way each module is specified, programmed, and documented is exemplified through the FCLib discrete first-order low-pass filter FILTLP1a.

Figure 3-1 shows the graphical representation in both Simulink and SCADE is a part of every FCLib module description in SDD. Each library module is equivalent to a Subsystem in Simulink and a User-Defined Operator in SCADE. Both visual representations display input and output names, as well as module identifiers. The inputs and outputs in SCADE correspond to the Inports and Outports in Simulink. Similarly, the mask parameters in Simulink correspond to the hidden inputs in SCADE.

DOI: 10.13009/EUCASS2023-953

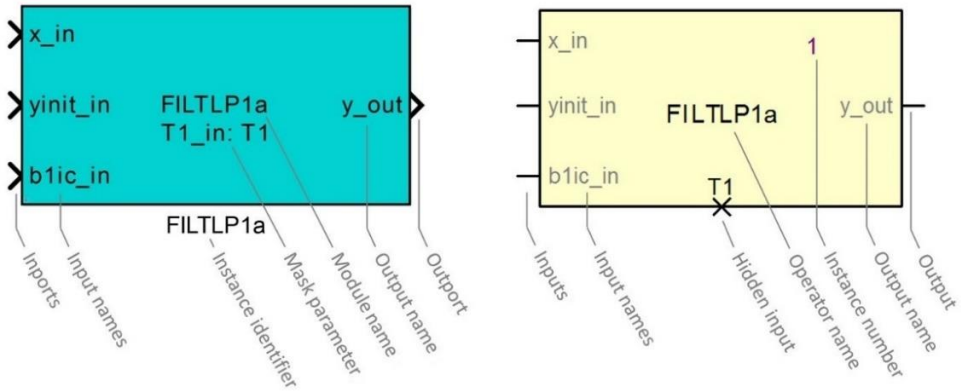

Figure 3-1: Graphical Representation of FCLib modules in Simulink (left) and SCADE (right)

The module interface description includes input, output as well as hidden inputs definition. Additionally, the internal states and used library parameters need to be identified. Figure 3-2 shows the definition of input variables for the FILTLP1a module in SDD. Each variable has its unique identification within Simulink and SCADE (input/output name), a symbol for the pseudocode description (algorithm variable), data type, value range and a short description.

(a) Input Interface

Input Name	Algorithm variable	Data Type Simulink	Data Type SCADE	Range	Description
x_in	x_k	single/ double	*float*	$[-FLT\_MAX, FLT\_MAX]$/ $[-DBL\_MAX, DBL\_MAX]$	Input value
yinit_in	y_{init}	single/ double	*float*	$[-FLT\_MAX, FLT\_MAX]$/ $[-DBL\_MAX, DBL\_MAX]$	Initial output value
b1ic_in	ic	boolean	bool	$[true, false]$	Initial condition flag

Figure 3-2: Exemplary definition of input interface variables for FILTLP1a module in SDD

The pseudocode in Figure 3-3 provides a formal definition of the algorithm, serving as a comprehensive instruction set for implementing the FCLib module. The temporal aspects of the algorithm are outlined by the initialisation and cyclic execution sections. In conjunction with the previously described interface and internal states, this algorithm is unambiguously defined and can be implemented.

The software High-Level Requirements (HLR) for the library modules correspond to the functionality description of each module given in SDD. The only difference to a general way of writing the requirements is the absence of the verb "shall". As Figure 3-5 shows, parts of the description of a library module which correspond to a requirement are parenthesised by the “§”-character and a consecutive Roman numeral. Each library module additionally defines a unique tag for each requirement in the Subsection “Traceability”. The tags are composed of the keyword “REQ”, the identifier of the module and a three-digit number that corresponds to the Roman numeral in the module description. The pseudocode and HLRs from the tagged functionality description do not provide an explicit design that is implemented in the modelling environment, but with this information, the designer can implement the model directly in Simulink and the tester creates the test scenarios to perform verification of the FCLib module.

After implementing all library modules in Simulink, they are then translated into Scade. To fully document the architecture, data, and control flow of each module in the SCADE Suite, the SCADE LifeCycle Reporter is used to automatically generate a Design Model Report (DMR) for all FCLib modules. The use of the SCADE LifeCycle Reporter as a pre-qualified tool ensures the correctness of the report generation.

The DMR provides comprehensive details about the module interface, including its data type, module hierarchy, comments, and transferred metadata from Simulink. Additionally, it includes the graphical representation of the Scade language control and data flow, along with all sub-operators. This graphical representation, as shown in Figure 3-4, enhances the understanding of the module's functionality and facilitates analysis and review processes.

DOI: 10.13009/EUCASS2023-953

Initialisation

$y_{k-1} = 0$

$x_{k-1} = 0$

$a = 0$

$b = 0$

$init = true$

Cyclic Execution

If $init$

$$a = \frac{\frac{2T_1}{\Delta t} - 1}{\frac{2T_1}{\Delta t} + 1}$$

$$b = \frac{1}{\frac{2T_1}{\Delta t} + 1}$$

If $init$ || ic

$y_k = y_{init}$

$init = false$

Else

$y_k = a \cdot y_{k-1} + b(x_k + x_{k-1})$

$y_{k-1} = y_k$

$x_{k-1} = x_k$

Figure 3-3: Pseudocode example of FILTLP1a

§(I) The continuous filter constant T_1 is a constant §. It is a mask parameter in Simulink and a hidden input in SCADE.

Re-arranging $F(z)$ yields:

§(II) $y_k = a \cdot y_{k-1} + b(x_k + x_{k-1})$,

with constant factors a, b:

$$a = \frac{\frac{2T_1}{\Delta t} - 1}{\frac{2T_1}{\Delta t} + 1} \text{ and}$$

$$b = \frac{1}{\frac{2T_1}{\Delta t} + 1} \text{ §.}$$

§(III) In the first execution cycle or if the input ic is $true$, the output y_k is set to y_{init} §.

Traceability

§(I) – REQ_FILTLP1a_001

§(II) – REQ_FILTLP1a_002

§(III) – REQ_FILTLP1a_003

Figure 3-5: Software HLRs for FILTLP1a module

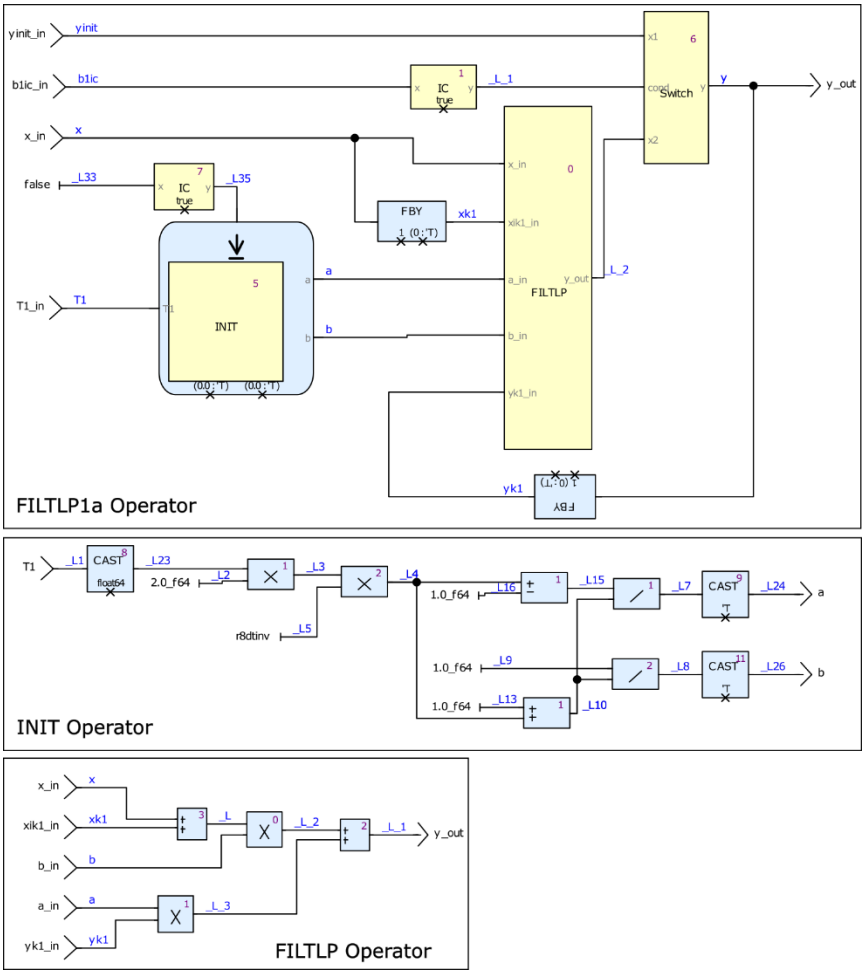

Figure 3-4: View of FILTLP1a module in Design Model Report

4. Library Verification Approach

This chapter explains the test and analysis methods employed for library verification and highlights the significant results obtained. These methods, evaluate the library timing performance and behaviour.

4.1 FCLib Test and Analysis Methods

The test process involves the requirement-based definition of test cases. The requirements identified in the SDD serve as the basis for the test case specifications in a Software Verification and Procedures Document (SVCP). A test case specification provides precise instructions on how to test a specific requirement. It is used to develop the test sequence for the SCADE Test Environment for HOST (TEH). Figure 4-1 shows a test case example for the DBZ module (divide by zero protection). A test case specification includes a unique test case identification, the corresponding requirement tag from which the test case is derived, a description of pre-conditions, required inputs, post-conditions, and expected results.

A test sequence comprises the set of input values and expected output values that represent one or multiple test cases. The test sequences are defined as ASCII comma-separated value (CSV) files and subsequently entered into the TEH. To generate these test sequences, MATLAB scripts are created based on the test case specifications. The expected values are defined using pseudocode and the requirements outlined in SDD, enabling the automated generation of the test sequences. To demonstrate the functional equivalence of the FCLib modules in both SCADE and Simulink, the test sequences developed for TEH are executed in Simulink as well.

DOI: 10.13009/EUCASS2023-953

Test Case ID	TC_FCLib_DBZ_001
Description	Verify that the output value is equal to input when absolute value of input is greater than division-by-zero protection threshold.
Satisfies Requirements	REQ_DBZ_001
Pre-Conditions	None.
Required Inputs	For `x_in` select a value greater than division-by-zero protection threshold `eps_in`. Use different equivalence classes for `x_in` and `eps_in`.
Post-Conditions	None.
Expected Results	The output `y_out` is equal to `x_in` input.
Remarks	None.

Figure 4-1: Example of Test Case Specification for the DBZ Module

The test sequences are prepared for the simulation of individual modules with TEH. These simulations encompass three distinct types of tests.

(1) *Instrumented tests* are applied when a polymorphic FCLib operator is tested. The internal data type definition of a polymorphic operator depends on the block usage context. For testing purposes, these blocks are instrumented with inputs and outputs of all possible data types according to their specification given in the SDD (see Figure 4-2), enabling the execution of the instrumented models.

(2) *Direct tests* are applied on non-polymorphic FCLib operators that have predefined data types. Such modules are executable and can be stimulated by input vectors from test sequences without any instrumentation.

(3) *MATHLib tests* are applied to math operators which are part of FCLib. The MATHLib modules require additional instrumentation to achieve better accuracy for stimulation values in test sequences. Reading the input stimulation of floating-point data type from the CSV file yields a loss of accuracy, which leads to deviations in the test results of the library module. To address this issue, the library module inputs are instrumented with union-type operators. These operators facilitate the transfer of an unsigned integer representation into any other data type and vice versa. This conversion allows stimulation with exact bit representations of floating-point numbers. The union-type operators (e.g., block UTuint64_f64 in SIN instrumentation, Figure 4-3) are implemented as imported C functions in SCADE and as S-Functions in Simulink. Different types of MATHLib instrumentations are applied:

- requirements-based tests and precision analysis,
- testing simple mathematical identities e.g. $\sin(-x) = -\sin(x)$, see upper diagram in Figure 4-3,
- error propagation analysis in mathematical identities e.g. $\exp(x - v) = \exp(x) * \exp(-v)$, see bottom diagram in Figure 4-3.

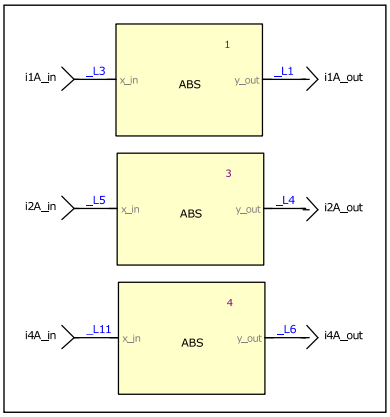

Figure 4-2: Instrumentation of polymorph ABS module

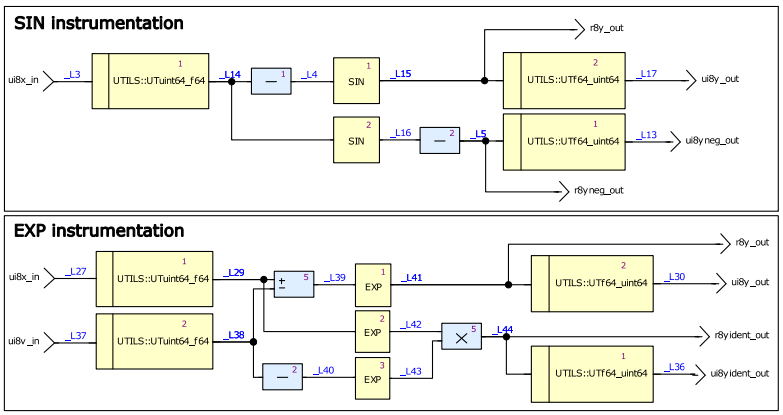

Figure 4-3: Instrumentation of SIN and EXP MATHLib modules

The structural model coverage analysis is performed with the SCADE Test Model Coverage (TMC) tool. The FCLib coverage analysis ensures that all elements and paths of the FCLib modules in SCADE have been stimulated and activated by requirement-based test cases. The considered coverage metrics are Decision Coverage (DC) as well as Data and Control (D&C) coupling that has to be achieved for the software DAL level C.

Both TEH and TMC are qualified tools according to DO-330. Tool qualification is required when the output of tools is used to automate, reduce, or eliminate objectives of the software development process defined by DO-178C/DO331. TEH's qualification ensures the correct evaluation of test results, while TMC's qualification ensures the proper acquisition of coverage results and generation of coverage reports.

One significant timing concern in the AFCL software is the utilisation of double precision math functions. The PowerPC MPC5567 target processor lacks a double-precision floating-point unit, necessitating the use of emulation routines for double-precision float computations. Consequently, the extensive execution time of navigation functions that use multiple double-precision math functions leads to exceeding the allocated Worst-Case Execution Time (WCET) for the AFCL SW. To address this issue, the execution time of FCLib math functions and the Wind River

DOI: 10.13009/EUCASS2023-953

Diab Compiler® math functions provided along with the compiler, are compared. Two approaches are used for timing analysis:

(1) Leichtwerk AG operates the Absint aiT WCET Analyzer tool, which performs static analyses of a task's intrinsic cache and pipeline behaviour based on formal cache and pipeline models. This analysis enables the computation of accurate and tight upper bounds for the WCET. The analysis is individually performed for each MATHLib module instance using compiled object code.

(2) Execution time measurement is carried out on the target processor in the Processor-in-the-Loop (PiL) test environment. The PiL configuration allows the execution of a desired software partition without the IMA platform's timing limits. This analysis is conducted by integrating each MATHLib instance on the PiL as an AFCL partition for the measurement campaign.

4.2 Evaluation of FCLib Test Results and Analysis

The functional tests performed on the FCLib modules, involving Boolean data types and integer operations, demonstrate compliance with the expected results. To evaluate the floating-point modules, tolerances for pass-fail criteria are defined in Table 4-1. Notable deviations are observed in single precision second-order low pass filter module FILPLP2a and the navigation equation modules.

All test sequences were repeated in Simulink. All FCLib modules in Simulink demonstrate the same behaviour and the same numerical deviations from expected results as in SCADE.

One of the reasons for these deviations is attributed to the format CSV files of the test sequence. The textual representation of the floating-point numbers leads to a loss of accuracy when the stimulation values are read.

Functional tests are executed and subsequent precision analysis is performed for instrumented MATHLib modules. The expected values for test sequences are generated with native mathematical functions in MATLAB. Figure 4-5 and Figure 4-6 depict the numerical deviations of double precision sine and single precision cosine from expected values. The deviations observed in cosine in the primary input range $[-2\pi, 2\pi]$ remain below the tolerance values. However, for input values exceeding $|x| > 8 \cdot 10^3$, the deviations violate the required numerical tolerances. Within the context of the AFCL software, these deviations are considered not critical since the relevant input range for the COSF module is limited to $[-2\pi, 2\pi]$.

Table 4-1: Tolerance Settings for Floating-Point Variables

FCLib Modules	ε_{rel}		ε_{abs}	
	single	double	single	double
time-independent	10^{-6}	10^{-8}	10^{-6}	10^{-8}
time-dependent	10^{-4}	10^{-6}	10^{-5}	10^{-6}

After the execution of requirement-based test sequences, the structural coverage analysis reveals complete coverage for both DC and D&C coupling metrics in all time-independent and math modules. However, a few time-dependent modules exhibit minor coverage gaps. Figure 4-6 provides an example of such a gap in the DERIV module, which is responsible for signal differentiation. In this instance, the initial value of the unit delay primitive FBY does not impact the module's output because the initial condition (IC) operator overwrites the output by zero in the first execution cycle. This behaviour violates the C&D coupling metric. The use of the unit delay is necessary to avoid an algebraic loop, while the DERIV module requires a specific initialisation value on its output. To mitigate this issue, a justification is provided, explaining the reasons why such a gap is deemed acceptable. These justifications flow into the generation of the coverage report, ensuring that the reasons for the observed gaps are considered and appropriately documented.

The coverage results achieved for the FCLib are leveraged for the coverage analysis of DFCL and AFCL software. In instances where an integrated FCLib module does not reach full coverage, the coverage gaps are justified by referring back to the coverage results obtained from the library's test. This approach allows the effective reuse of existing coverage information and provides a basis for explaining observed coverage gaps within the integrated software.

Results in Table 4-2 indicate that MATHLib implementations of sine, cosine, and tangent functions require significantly less time (52% to 66%) compared to DIAB functions. However, MATHLib modules for arc cosine, arc sine, and arc tangent functions require nearly twice the time compared to DIAB functions. The timing results of Absint analysis and PiL measurement correlate with each other. Absint analysis provides a more conservative estimate compared to PiL measurement.

DOI: 10.13009/EUCASS2023-953

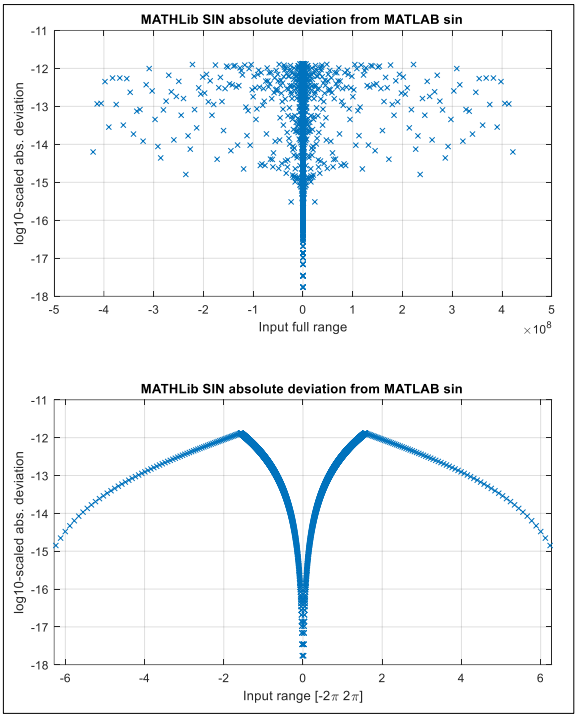

Figure 4-4: double precision sine (SIN) deviation from MATLAB sine

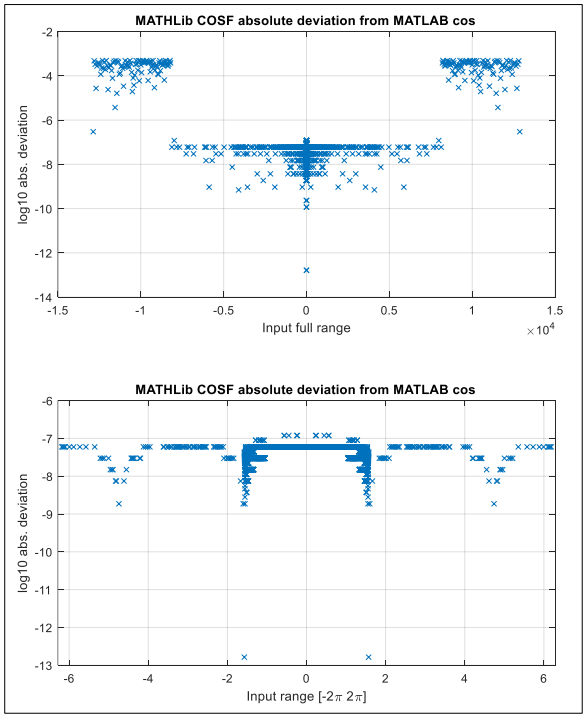

Figure 4-5: single precision cosine (COSF) deviation from MATLAB cosine

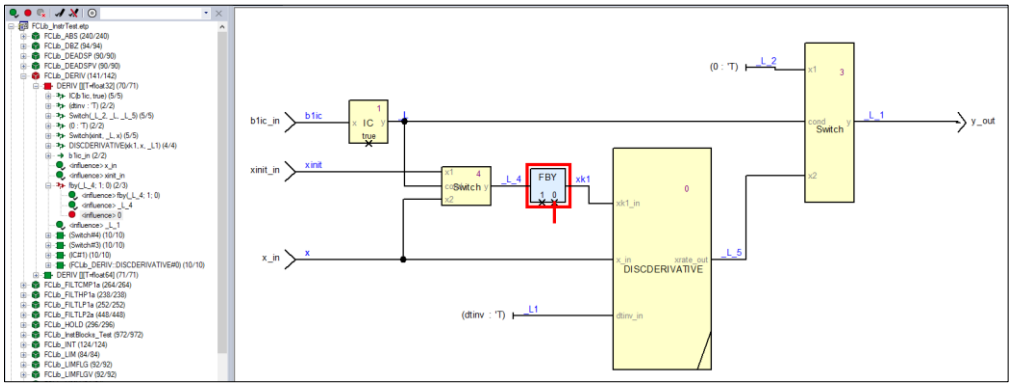

Figure 4-6: Visualisation of a coverage gap in the FBY primitive in the DERIV module in SCADE IDE

Table 4-2: WCET analysis for selected MATHLib modules

Module	Max. DIAB [µs]	Max. MATHLib [µs]	Ratio DIAB/MATHLib
PiL results			
COS	735	628	120 %
SIN	734	576	133 %
TAN	832	603	145 %
ACOS	533	932	52 %
ASIN	536	930	53 %
ATAN	460	798	52 %
Absint results			
COS	1353	893	151 %
SIN	1353	827	164 %
TAN	1405	725	194 %
ACOS	529	1124	47 %
ASIN	550	1123	49 %
ATAN	399	857	47 %

DOI: 10.13009/EUCASS2023-953

5. Conclusion

The development process for flight control laws includes two processes: the system development process (SAE 4754A) and the software development process (RTCA DO-178C and DO-331). The efficiency of the FCL development can be significantly increased through model-based development, automation in testing and by using ready-made library elements. For the design of the flight control functions, Mathworks Simulink is a common tool, whereas for the FCL software design, Ansys SCADE Suite has the advantage to significantly reduce workload in the certification process, as the transition from block diagram to executable code is formally proven. A process that combines the advantages of both tools was developed. This process includes the automated translation of Simulink models to Scade. It was applied to design and verify the FCLib library elements – and then for the DFCL SW and the AFCL SW.

The elements of the library FCLib play an important role in the translation procedure. The re-use of test and coverage results, significantly reduce the verification effort, especially repetitive unit tests to ensure 100% Decision Coverage and Control & Data coupling. The FCLib provide a wide range of functionality and can be customised according to specific project requirements. This versatility enables its usage as a stand-alone library in both Simulink and SCADE environments. The FCLib has also been successfully employed in the MODULAR[5] to demonstrate efficient processes for flight control law and actuator control software development.

The library design enables rapid extension to accommodate specific requirements for new library modules. The availability of a well-documented and verified library aids small teams with limited personnel resources when developing safety-critical software.

Finally, it is recommended to further investigate numerical accuracy, particularly concerning mathematical modules. One potential solution is to explore more precise approximation methods for these modules and assess their impact on execution time.

Acknowledgement

The presented work is the result of the FCL-Methods research project at the Department for Flight Mechanics, Flight Control and Aeroelasticity, Technische Universität Berlin. FCL-Methods was funded and supervised by Leichtwerk AG. The results and methods were refined in the scope of the subsequent joint project IBAS in the context of the public sponsorship program Niedersächsische Luftfahrtförderrichtlinie funded by NBank.

References

[1] EASA. Nov. 2021. Certification Specifications and Acceptable Means of Compliance for Large Aeroplanes (CS-25): Amendment 27, CS-25.

[2] ARP4754A. 2010. Guidelines for Development of Civil Aircraft and Systems. S-18 Aircraft and Sys Dev and Safety Assessment Committee, SAE International.

[3] RTCA DO-178C. Dec. 2011. Software Considerations in Airbone Systems and Equipment Certification. Radio Technical Commission for Aeronautics, RTCA.

[4] Torres-Pomales,W. Dec. 2014. "Is Model-Based Development a Favorable Approach for Complex and Safety-Critical Computer Systems on Commercial Aircraft?". NASA, Langley Research Center.

[5] Schmidt, D.C. 2006. “Guest Editor’s Introduction: Model-Driven Engineering”. In: *Computer*. vol. 39. no. 2. pp. 25–31.

[6] RTCA DO-330. Dec. 2011. Software Tool Qualification Considerations. Radio Technical Commission for Aeronautics, RTCA.

[7] Colaço, J.-L., Pagano, B., and Pouzet, M. 2017. SCADE 6: A formal language for embedded critical software development (invited paper). In *2017 International Symposium on Theoretical Aspects of Software Engineering (TASE)*. pp. 1–11.

[8] Walde, G., and Luckner, R. 2016. Bridging the tool gap for model-based design from flight control function design in Simulink to software design in SCADE. In: *2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC)*. pp. 1–10.

[9] Kickert, R. 2021. High Altitude Platform Stations (HAPS). In: *DGLR Magazin Luft- und Raumfahrt*. vol. 2/2021. pp.17-19.

[10] RTCA DO-331. Dec. 2011. Model-Based Development and Verification Supplemant to DO-178C and DO-278A. Radio Technical Commission for Aeronautics, RTCA.

[11] Leichtwerk AG. 2021. High Altitude Platform Stations (HAPS)-A Future Key Element of Broadband Infrastructure. https://www.leichtwerk.de/fileadmin/start/HAPS_WhitePaper_A11_e.pdf (accessed Jul. 5, 2023).

[12] Bourke, T. Carcenac,F. Colaço, J.-L. Pagano, B. Pasteur, C. and Pouzet, M. 2017. A Synchronous Look at the Simulink Standard Library. In: *ACM Trans. Embed. Comput. Syst.* vol. 16.

[5] Research project in cooperation with Liebherr Aerospace Lindenberg (Bundesministerium Wirtschaft und Energie, Luftfahrtforschungsprogramm LUFO VI-1)

DOI: 10.13009/EUCASS2023-953

[13] ANSYS, Inc. Nov 2020. SCADE Suite Libraries Manual. ANSYS SCADE Products.
[14] Arregi, A. Schriever, F. Arias, C. and Jung, A. 2019. Ensuring Numerical Reproducibility for Model-Based Software Engineering. doi: 10.13009/EUCASS2019-790.
[15] Nürnberger, K. 2019. Development of Elementary Mathematics Functions in an Avionics Context. PgD Thesis. Technische Universität München.
[16] Gorke, S. Riebeling, R. Kraus, F. and Reichel,R. 2013. Flexible platform approach for fly-by-wire systems. In: *2013 IEEE/AIAA 32nd Digital Avionics Systems Conference (DASC 2013): East Syracuse, New York, USA, 5 - 10 October 2013*, East Syracuse, NY, USA.
[17] Luckner, R. Dalldorff, L. and Reichel, R. 2014. A utility aircraft for remote sensing missions with a high-precision automatic flight control system. In: *2014 IEEE Intenational Conference on Aerospace Electronics and Remote Sensing Technology (ICARES 2014).* pp. 1–11.
[18] ANSYS, Inc. Nov. 2020. Scade Language Reference Manual. ANSYS SCADE Products.
[19] Benveniste, A. and Berry, G. 1991. The synchronous approach to reactive and real-time systems. In: *Proc. IEEE*, vol. 79, no. 9, pp. 1270–1282.
[20] Cody, W.J. and Waite, W.M. 1980. Software manual for the elementary functions (Prentice-Hall series in computational mathematics). Englewood Cliffs, NJ: Prentice-Hall.