Aurora

QQen Evaluation Report

D3.6.1

Document Code: AUR-ESC-RP-0007
Document Version: 1.0
Document Date: 31/07/2021
Internal Reference: DOC00XXYYYY

SENER Aeroespacial

POLITÉCNICA

SPACE

esc Aerospace

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101004291

Signature Control

Written	Checked	Approved Configuration Management	Approved Quality Assurance	Approved Project Management
A. Lyko Antonin Lyko Digitally signed by Antonin Lyko DN: C=CZ, O=ESC, CN=Antonin Lyko, E=antonin.lyko@evolvsys.cz Reason: I am the author of this document Location: your signing location here Date: 2021.08.30 09:52:50+02'00' Foxit PDF Reader Version: 11.0.0	V. Gómez P. Česák Digitally signed by petr.cesak@evolvsys.cz Date: 2021-08-27 18:01:00	R. Talavera Ana Isabel Rodriguez Firmado digitalmente por Ana Isabel Rodriguez Fecha: 2021.10.21 09:35:37 +02'00'	Alfonso López Alfonso López Firmado digitalmente por: Alfonso López ND: CN = Alfonso López email = alfonso.lopez@aeroespacial.sener C = ES O = SENER Aeroespacial S.A. OU = Quality Division Fecha: 2021.10.20 14:51:13 +02'00'	A. Rodríguez Ana Isabel Rodriguez Firmado digitalmente por Ana Isabel Rodriguez Fecha: 2021.10.20 14:39:22 +02'00'
Date and Signature	Date and Signature	Date and Signature	Date and Signature	Date and Signature

Signature not needed if electronically approved by route

Changes Record

Rev	Date	Author	Affected section	Changes
1.0	2021-07-31	A. Lyko	All	Initial version - QGen Debugger - SIL - chapter 7.

Index

1. Introduction

1.1.Purpose

The purpose of this document is to evaluate QGen toolchain by means of practical exercises performed on proprietary internal AOCS model/s or its subsystems from former Euclid mission.

The whole process shall be described with all observed constraints and benefits.

1.2. Scope

The appropriate tools under scope of evaluation shall be the main tools from QGen toolchain. Model Verifier, Code Generator, Debugger and `qgenpil`.

Workflow for each tool or exercise shall be described.

1.3. Prerequisites

The user shall be familiar with Matlab and Simulink.

The user shall have set-up the QGen toolchain and development environment as described in [RD04].

2. Related Documentation

2.1. Applicable Documents

AD #	Title	Reference	Issue	Rev
[AD01]	AURORA Grant Agreement	GA number 101004291	-	-
[AD02]	Quality Assurance Management Plan	AUR-SAE-PL-0001	-	1
[AD03]	AURORA SW Development Plan	AUR-SAE-PL-0002	-	1

Table 1 Applicable documents

2.2. Reference Documents

RD #	Title	Reference	Issue	Rev
[RD01]	Space engineering Software	ECSS-E-ST-40	C	-
[RD02]	Space Software Product Assurance	ECSS-Q-ST-80	C	-
[RD03]	QGen User Guide	January 12, 2021	-	21.1
[RD04]	QGen toolset and SW Development Environment	AUR-ESC-RP-0022	-	1.1
[RD05]	MATLAB Overview			
[RD06]	Processor-in-the-loop simulation on embedded Linux boards			
[RD07]	Software-in-the-Loop Testing glossary			
[RD08]	Get Started with Simulink			
[RD09]	Signal Builder user guide			

Table 2 Reference documents

2.3. Acronyms

Acronym	Description
AD	Applicable Document
CLI	Command Line Interface
GUI	Graphical User Interface
IDE	Integrated Development Environment
PIL	Processor in the loop
RD	Reference document
SIL	Software in the loop
TBW	To be written

Table 3 Acronyms

2.4. Terms and Definitions

Acronym	Description
AdaCore	SW development tools supplier company
Matlab	Programming and numeric computing platform used to analyse data, develop algorithms, and create models [RD05].
Processor in the loop	In processor-in-the-loop (PIL) simulation, the generated code from model runs directly on the target hardware, which means you can test models on the hardware using the same test cases as on the host [RD06].
QGen	Qualifiable code generator and static analyser for Simulink(R) and Stateflow(R)
Software in the loop	The Production Software Code is incorporated into the mathematical simulation that contains the models of the Physical System [RD07].
Simulink	Block diagram environment for simulation and Model-Based Design integrated with MATLAB [RD08][RD07].

Table 4 Terms and Definitions

3. Overview

This chapter shall be completed within the further version releases of this document.

4. System Requirements

This chapter shall be completed within the further version releases of this document.

The following section and table stand only as a placeholder.

4.1. Traceability

Requirement ID	Requirement title	Document section
TBD	TBD	TBD

Table 5 Placeholder

5. QGen Model Verifier

This chapter shall be completed within the further version releases of this document.

6. QGen Code Generator

This chapter shall be completed within the further version releases of this document.

7. QGen Model Debugger - software-in-the-loop

At the point of an already generated code from the user's chosen model, the next logical step would be to verify that the generated code behaves as its model reference. This can be performed on the host computer, so called software-in-the-loop (SIL), or on the target platform, so called processor-in-the-loop (PIL).

From the evaluation perspective, the QGen Debugger functionalities may be distinguished into two categories:

- Debugger
- Software-in-the-loop (SIL)

The debugger functionalities provide features like performing execution steps over the generated code lines side to side the reference model, setting breakpoints within the code or the model, reading variable current values etc., where additional details are described in document [RD03].

The software-in-the-loop test functionalities provide features like calling out the generated code with the prepared set of input values, gathering the calculated output values and comparing them with the expected results, measuring the min, max, avg. execution time of all simulation steps and providing the report with the test results.

From the generated code verification perspective, the SIL test would be the first step and the final verification goal on the host platform. The Debugger functionalities would come handy in place mostly in case of some irregularities and for issue solving activities after which the SIL test with pass result would still be the next step.

Therefore, the following sections focus mainly on describing the SIL exercises with the QGen Debugger tool.

7.1. SIL Theory

In theory, the reason behind the SIL is to verify that the transition from the model perspective to the generated (manually or automatically) code perspective was done without introducing any new unintentional or unexpected behavior/issues for defined use-cases – test-cases.

This is done by passing the same set of input values into the system model as into the system executable (code). Ideally theirs output values shall be identical.

In practice we can measure some small differences between those two sets of output values. Therefore, some small acceptable tolerance of output values shall be defined, e.g.: 1E-10.

By a test-case, the set of input-output pair values (obtained by running model simulation) is meant. This Test-case is then used as a reference for the validation of the code behavior.

Figure 1 displays software-in-the-loop test ideal workflow.

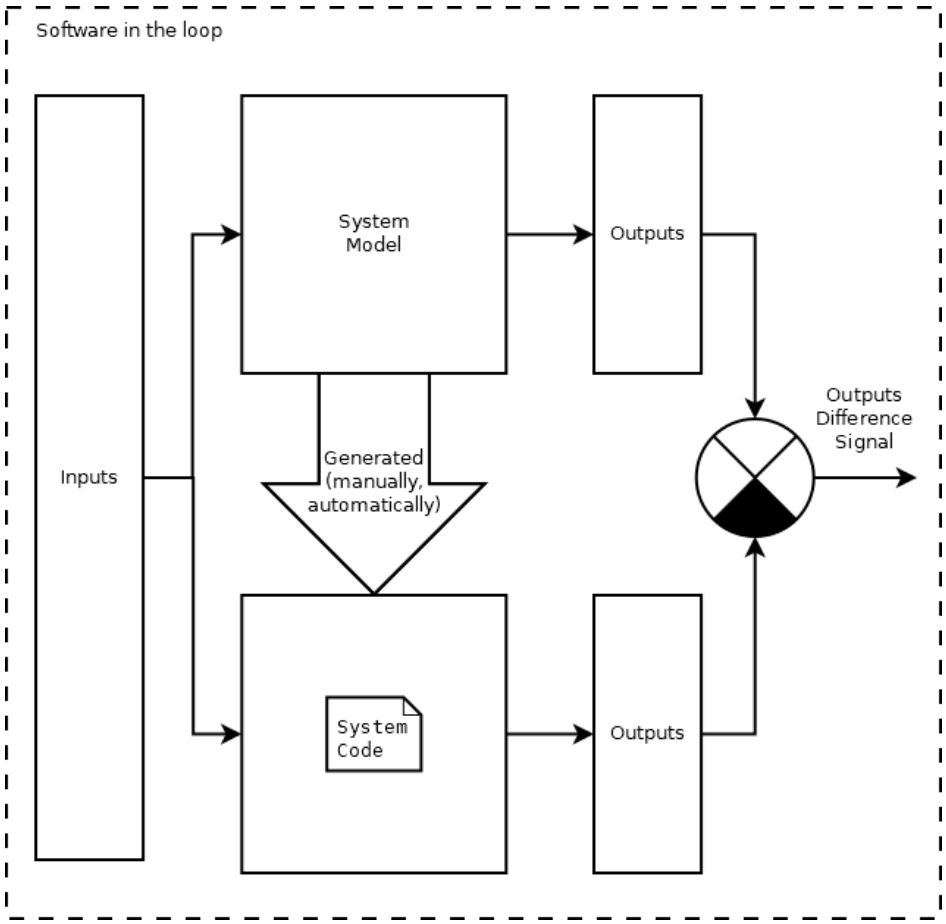

Figure 1 SIL Theory

7.2. QGen Debugger Workflow

Document [RD03] in chapter 5.2.1 describes three types of QGen Debugger workflows based on the starting point within the whole process:

- The first time to debug the model
- When the model has not changed
- When the model has changed

Figure 2 tries to visualize and summarize the main essence of those workflow types.

The first workflow, "first time to debug the model", guides the user, beginning in the model perspective, throughout intermediate steps, towards to running SIL and obtaining test summary report in form of explicitly defined simulation steps when the output values were outside of the chosen tolerance.

Based on Figure 2, the workflow "first time to debug model" would follow the sequence:
[A/B I., A II., A III., B II., B III.].

Second workflow, "when model has not changed", advice to start debugging IDE – GNAT Studio immediately without performing preceding steps originated in the user model, assuming this was already done at least once.

Based on Figure 2, the workflow "when model has not changed" would start directly at step:
[B III].

Third workflow, "when model has changed", advice to use similar procedure to the first workflow, "first time to debug model", except of using GUI to trigger the intermediate XMI model export, it is advised to use CLI function instead, skip running the simulation and thus not-updating simulation results and only re-generate system code within Debugger IDE – GNAT Studio (at this point, the information about the model for the code generation shall be taken from intermediate XMI model export – so there is no direct connection to/need for the Simulink model itself) - and start debugging. This workflow also assumes that the first workflow, "first time to debug the model" was performed at least once.

Based on Figure 2, the workflow "when model has changed" would follow only the B branch:
[A/B I., B II., B III.].

For Aurora QGen Debugger evaluation, the first workflow, "first time to debug the model", was chosen to be used, as:

- It is needed to be performed at least once when using other workflows
- It contains all steps included in other two workflows plus additional steps

The whole evaluation process shall therefore start at step A/B I. from Figure 2.

It is important to mention, that all transitions between steps from Figure 2 are automated by QGen and user does not have to worry about implementation details of intermediate steps. User can just trigger transition from one step to another via GUI (or in some cases via CLI).

In particular, the simulation model (A II.), simulation log csv (A III.), model export XMI (B II.), system code (B III.), simulation framework code (B III.), the SIL executable (B III.) and summary test report (B III.) are generated by QGen automatically.

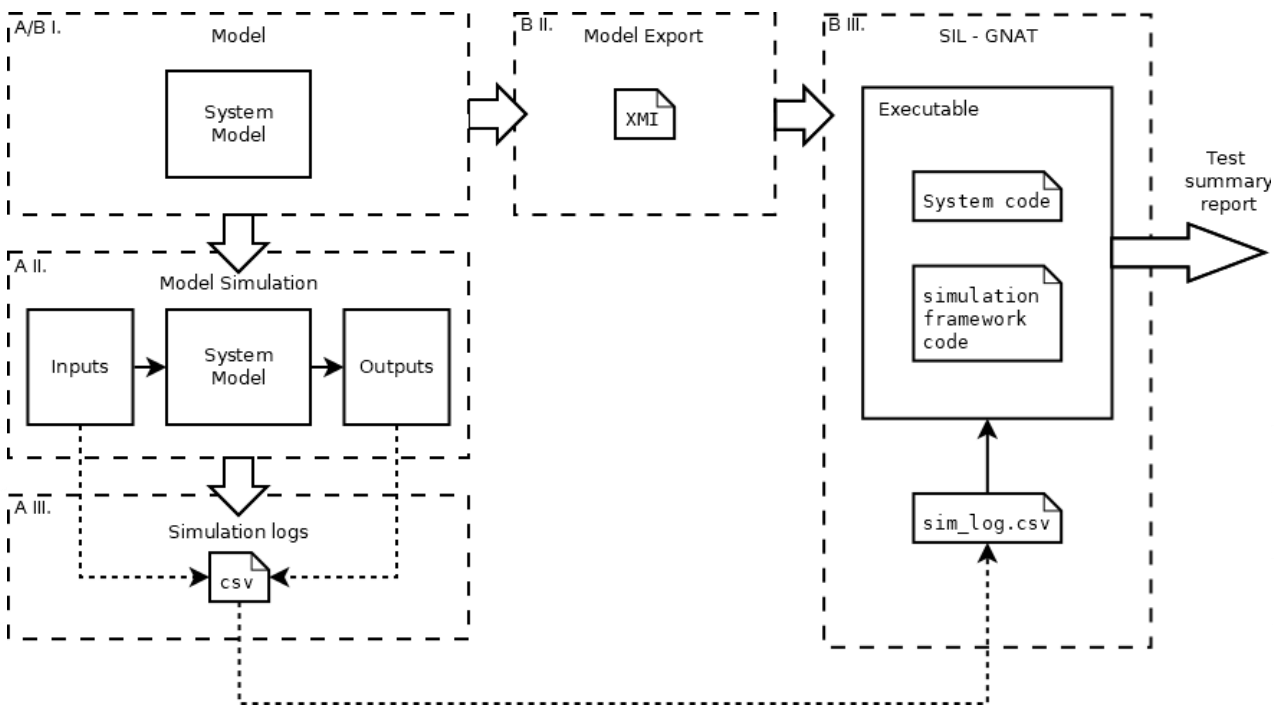

Figure 2 QGen Debugger – SIL- workflow

7.2.1. Simulation Framework Code

As mentioned in section 7.2, the 'simulation framework code' is generated by QGen automatically and consists of two main parts: the model/simulation specific part and the model independent or generic part.

The main functionalities included within the 'simulation framework code' are:

- Parsing the SIL executable arguments, which are the:
  - 'Simulation log csv file' path (A III., B. III.),
  - Chosen 'tolerance' for this SIL execution,
  - Chosen name of the 'test summary report' output file.
- Parsing the 'simulation log csv' file with built-in knowledge about the port and signals order within.
- Calling the system code in iterations and providing the appropriate input values from 'simulation log csv' file for each simulation step and obtaining the calculated system output values.
- Measuring the execution time of the 'system code' for each simulation step and determining the min., max., and avg. execution time.
- Comparing the obtained output values from the 'system code' with the expected output values from the 'simulation log csv' file (taken from model simulation) for each simulation step
- Logging all simulation steps with the 'system code' output values outside of the acceptable tolerance interval
- Generating the output 'test summary report' file

Some of those functionalities are generic and model independent like capability to parsing CLI arguments, working with files (reading, writing), measuring the execution time, or generating the output report file.

Some functionalities are model/simulation specific, e.g.: build-in knowledge about the 'simulation log csv' file structure (number and dimension of input-output ports may vary for each model, each simulation may have different number steps to be simulated) or calling the system code (based on the arguments used with `qgenc`, the generated system code may use different type of interfaces – separated function parameters, inputs and outputs bundled into structures or use global variables instead of function parameters).

Therefore the 'simulation framework code' shall be generated for each specific model and its simulation.

The legal section within the 'simulation framework code' states, that there is no warranty and user may change those files.

7.2.2. Simulation Log CSV

Based on the 'simulation framework code' (described in section 7.2.1, B III. in Figure 2) and 'simulation log csv' file (A III., B III. in Figure 2) analysis, the magic keyword, called 'Reset', was revealed.

When putting the 'Reset' string into the 'simulation log csv' file on a separated line, the 'simulation framework code' will treat this as test-scenario separator and will reset all its internal variables, re-initialize the 'system code' and start a new SIL test.

7.3. SIL Evaluation workflow

7.3.1. Prerequisites

As mentioned in section 7.2, for the QGen Debugger – SIL – evaluation was chosen type of workflow, which begins in the model perspective and ends with SIL test summary report. Therefore, all necessary intermediate steps are exercised.

This type of workflow is called "the first time to debug model" in [RD03] nomenclature and in the Figure 2 perspective is represented by sequence of steps: [A/B I., A II., A III., B II., B III.].

Base on everything so far said, it may seem that the only thing needed for beginning the SIL exercise is the 'system model' itself for the step A/B I. from Figure 2 as a starting point. This statement is not completely true. The second essential ingredient for valuable SIL exercise is a defined test-case or test-cases of interest via the set of chosen input values, step A II. from Figure 2.

To summarize the SIL starting point discussion, the essential prerequisites for the QGen Debugger – SIL – evaluation are:

- System model (step A/B I. in Figure 2)
- Test scenario input values (step A II. in Figure 2)

Note: the reference output values used within the SIL workflow are generated during the model simulation (step A II. in Figure 2).

Both of those two used prerequisite-items were provided by the Euclid mission project as its heritage.

7.3.2. Input Signals

As defined in document [RD03], the tool used in QGen Debugger workflow for input values definition is a so called 'Signal Builder'.

Based on a simple and quick user familiarization with the Signal Builder tool, the conclusion was that signals can be defined only manually. Therefore, the first idea how to import the user defined input signals, was via the 'simulation log csv' file from step A III. and B III. in Figure 2.

This approach worked fine for the simplest model examples with one-dimensional ports, where inputs and outputs signals were ordered accordingly in columns (in1, in2, …, out1, out2).

After using model with multi-dimensional ports [e.g.: 3x1, etc.], the order of the signals within the 'simulation log csv' file got mixed up (e.g.: out2, in2, out1, in1, …). Therefore, the automated transition from input-output values to simulation log csv file would not be possible without the explicit knowledge of the signals order rule.

Note: Document [RD03] does not contain direct instructions how to import the test-scenario signal values, only a reference to the Signal Builder documentation, which was not at first analyzed deeply enough.

Therefore, upon a request how to properly import the user defined input values into the QGen Debugger workflow, and propagate them till the end – SIL executable, the AdaCore support team recommended to use the CLI interface function for generating the model simulation (A II. in Figure 2) together with input signals already imported within the Signal Builder:

```
>> qgen_make_sim_model('mymodel.slx', '--csv', 'input_values.csv');
```

Where:

- mymodel.slx is the name of the model for which the simulation shall be generated
- input_values.csv is file which define the set of input values for a particular test scenario

An example of a data format for a '`input_values.csv`' file is the following:

```
, 0.0, 0.1, 0.2, 0.3, 0.4
in1_signal_name, 0.0, 1.0, 2.0, 3.0, 0.4
in2_2x1_signal1_name, 0.0, 0.0, 0.0, 0.0, 0.0
in2_2x1_signal2_name, 1.0, 1.0, 1.0, 1.0, 1.0
```

The first line must start with comma character and contain the timeseries for all following signals. Other lines contain the signal values, where first item of each line represent the signal name. For signal with higher dimension than 1 (e.g.: [2x1], etc.), each dimension is placed on separated line.

Based on the `qgen_make_sim_model` help output, several csv files may be provided to generate simulation model for several test-scenarios in a row (not limited to one test-case).

7.3.3. Used Evaluation Workflow

The Figure 3 tries to summarize previous statements into a one picture describing the used evaluation workflow.

When QGen simulation model is generated (A II. in Figure 3), this is a good opportunity to validate transition from Euclid Simulation model (D II. in Figure 3) to QGen Simulation model (A II. in Figure 3). Both, the inputs, and outputs signals shall be equal (E I. and E II. in Figure 3) if no error occurs throughout the process, including the self-csv generation (C I.).

For the comparison purpose, input and output values within the simulation models (A II., D II.) were captured, using the 'bus creator' and 'to file' block and *.mat file format.

To summarize the final evaluation workflow, the used sequence would be the following:

1. Run Euclid Simulation model and prepare the inputs and output values (D II.)
2. Generate the csv file containing the input signals (C I.)
3. Generate the QGen simulation model by `qgen_make_sim_model` and by referencing the csv file containing the input signals and model itself (A II., C I., A/B I.)
4. Run the QGen model simulation (A II.)
5. Compare the Euclid and QGen simulation input-output values (E I. and E II.)
6. Generate the simulation log csv file (A III.)
7. Start GNAT – generate system code, model-specific simulation framework code and build executable (B II., B III.)
8. Run SIL, check summary report (B III.)

Notes:

The used evaluation workflow was based on a wrong assumption, that input signals can be defined only manually within the Signal Builder and AdaCore support team specific workflow recommendation. Although it is important to mention, that within the AdaCore support request, the importance of an automatable solution was stated. And indeed, current evaluation workflow can be automated at least within the Matlab perspective.

The activities focusing on automated solution within the GNAT studio (simulation framework code generation and specific project file *.gpr generation were not successful because of limited time allocation and other task priorities).

In fact, during writing sessions of this document, the information about possible imports of input signals into the Signal Builder [RD09] was revealed thanks to the originally overlooked link within document [RD03], section 6.2.

Based on the quick analysis it looks like imports are possible from several sources, csv, xls and mat format, where mat files could be the most suitable and 'handy' for the evaluation purposes.

Because the imports are triggered via GUI, it is not clear if automated solution would be possible in this way, whereas the QGen CLI supports it.

Following observation were therefore based on the workflow described in this section 7.3.3.

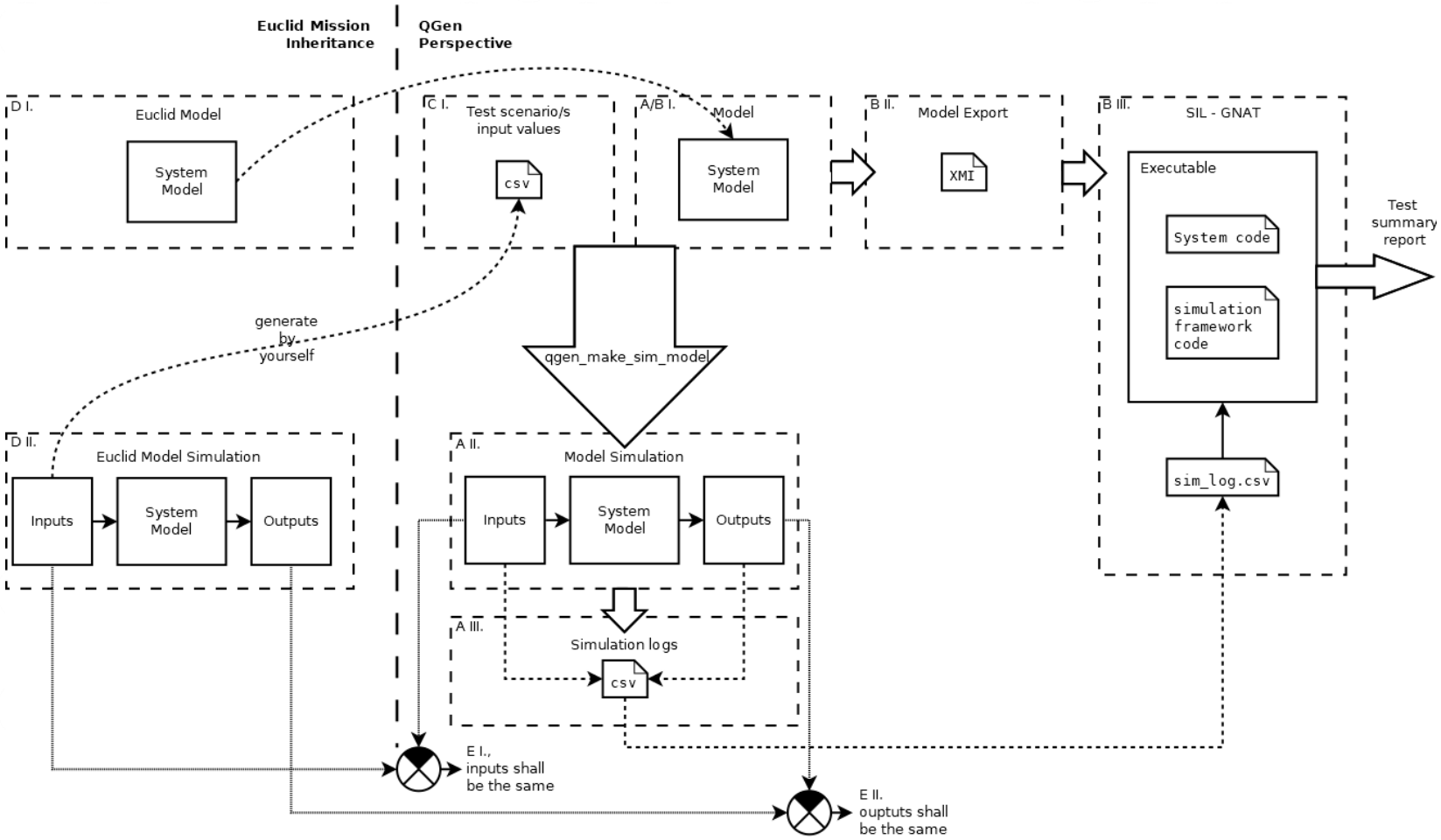

Figure 3 QGen Debugger – SIL – evaluation workflow

7.4. Exercises

Based on the used evaluation workflow, described in section 7.3.3, several models from Euclid mission were exercised this way.

Because those models are proprietary, only the generally used techniques and relations between models will be stated for sake of explanation clarity but without any implementation details.

The following Table 6 tries to explain the hierarchy between the exercised top model and its sub-systems: ([Y] – yes exercised directly, [N] – not exercised directly, only indirectly via top-model)

Top model	Referenced sub-systems
sam_ctrl [Y]	sam_ctrl_at [Y]
	sam_ctrl_dz [Y]
	sam_ctrl_rd [N]
	sam_ctrl_sa [N]

Table 6 Exercised models hierarchy

7.4.1. Exercise 1: sam_ctrl_dz

The first QGen Debugger SIL exercise performed on one of the Euclid's heritage models was done with the sam_crtl_dz sub-system.

Table 7 summarize the general sam_crtl_dz model properties:

Property	Value
Referenced Library	yes
Referenced external sub-system	no
Multi-dimensional ports	yes
Referenced WS parameters	yes
Ref. WS param. Types	Structure
	Matrix
Used Enums within the model	no

Table 7 sam_ctrl_dz general properties

The following table summarizes the status of each workflow step with observed constraints/issues and used fixes:

Workflow step	Performed	Status	Constrains / Issues	Fix
1. (D II.) **- Run Euclid Simulation** **- prepare inputs-output values**	yes	PASS	-	-

Workflow step	Performed	Status	Constrains / Issues	Fix
2. (C I.) **- Generate test scenario csv file (input values only)**	yes	PASS	-	-
3. (C I., A/B I., A II.) **- Generate QGen simulation model** **(run `qgen_make_sim_model`)**	yes	PASS	-	-
4. (A II.) **- Run QGen simulation model**	yes	FAIL → PASS	E1_I1 WS param. not referenced automatically	Add param. reference manually: ModelRef → Block Parameters →Arguments
4. (A II.) **- Run QGen simulation model**	yes	FAIL → PASS	E1_C1 "Model will not inherit sample time because the block 'SignalBuilder/ FromWs' disallows it"	Set: Simulation → Model Configuration Parameters → Solver → Periodic sample time constraint: Unconstrained
4. (A II.) **- Run QGen simulation model**	yes	UPDATE → PASS	E1_C2 The simulation model did not inherit the simulation start, stop and step time from the Signal Builder signals (csv time-series)	Set manually or via CLI: simulation start, stop and step time.
5. (E I., E II.) **- Compare Euclid vs QGen simulation input-output values**	no	-	-	-
6. (A III.) **Generate simulation log csv file**	yes	PASS	-	-

Workflow step	Performed	Status	Constrains / Issues	Fix
7. (B III.) **- generate system code**	yes	FAIL → PASS	E1_C3 Cannot open input file: model Input file not foundd	Model path shall not contain any spaces. The model XMI path is passed to the CLI `qgenc`. Spaces probably breaks the internal arguments parser.
7. (B III.) **- generate simulation code** **- build executable**	yes	FAIL → PASS	E1_I2 Compilation fails. In generated simulation code, system initialization function, the WS parameters were connected incorrectly.	Connect WS parameters manually, located at `qgen_base_workspace.h` file
8. (B III.) **- run the SIL executable** **- check the output report**	yes	PASS	-	-

Table 8 sam_ctrl_dz exercise result

The root-cause of the issue E1_I1 was not investigated, nor AdaCore support team was asked about it. To reveal if the whole workflow is performable to the SIL report, stated fix was applied and exercise continued to the next steps.

The consequence of the constraint E1_C1,, its root-cause or other possible solutions were not investigated. Related solution was applied, and exercise continued to the next step. Although it may be important to mention this constraint. Expert Simulink users may be aware of other consequences or different solutions.

The constraint E1_C2 caused, that start time of the generated simulation model (A II.) was set to 0.0s, stop time to 10.0s and step size to 'auto'. This happened even though the time-series information was already included in the 'test scenario csv' file (C I.) and properly included into the Signal Builder. As a fix of this was used a manual or via CLI update of those values.

Constraint E1_C3 message is self-explaining, although its root-cause may not. Information about this constraint was not found in document [RD03], although not using spaces in directories names and paths is considered generally as a good practice.

After investigating the root-cause of the issue E1_I2, several problems were revealed in the generated 'simulation framework code' within the initialization function and passing the system configuration parameters in (in model perspective they were in workspace).

The implementation method, of passing the model WS parameters into the initialization function as an argument and internally copying them into the component internal memory seem like a good practice. It is the one of necessary steps for utilizing the same model across the project only with different configuration parameters – like a class-objects relations. Although, in this exercise, model was used only once, with constant set of configuration parameters.

Another benefit of this type of component configuration parameters implementation is, that it allows to integrator, to change the component configuration during the run-time.

It was not investigated if this behavior (type of system-code configuration parameters implementation) can be changed or configured within QGen code generator `qgenc`.

The first problem with passing the system configuration parameters into the initialization function was with wrong arguments data type casting. Because of this, the compilator reports an error and executable was not built.

The second problem with passing the system configuration parameters into the initialization function was, that the local structure which shall hold the configuration parameters values was just declared, but not initialized. Its data may be random or empty.

After connecting the proper parameters structure from the `qgen_base_workspace.h` file (which is generated by QGen automatically) into the initialization function, an executable was successfully built.

After running the SIL executable and passing the 'simulation log csv' file in, the minimum acceptable tolerance of 5E-12 was found by experiment over several iterations. After minimizing acceptable tolerance even more, the output report file was containing each failed simulation step with explicitly defined the expected and obtained output value.

Because of the successful SIL output results, there were no suspicions to question the transition from the Euclid simulation model (D II.) to QGen simulation model (A II.) and therefore the evaluation workflow step no. 5. (E I. and E II.) was not performed. The SIL validates only the transition from the system/simulation model (A/B I., A II.) to the system code behavior (A III.), therefore the SIL result has no relation to the transition from the Euclid model simulation (D II.) to the QGen simulation model (A II.). Therefore, this assumption was not right.

After experimenting with multiple test scenarios placed in one 'simulation log csv' file, separated by 'Reset' keyword as explained in section 7.2.2 and intentionally letting the test failed by setting zero tolerance to observe the format of the output test report file, the following issue was found:

- The first test case correctly contained the failed simulation steps
- The second test case also contained the failed simulation steps from previous test case

The failed simulation steps were being reported over several test-cases in cumulative way. Because of that, for further test cases was difficult to determine, which simulation steps indeed failed in those test cases (and which reported failures was from previous test cases).

As mentioned in section 7.2.1, for the 'simulation framework code', which is responsible for generating the output report file, there are no warranties and user may change those files.

Because of this, the fix in form of proper re-initialization of the 'simulation framework code' internal variables, on the event when 'Reset' keyword occurred, was applied. This solution led to cleared-up test report.

7.4.2. Exercise 2: sam_ctrl

The second exercised model from the Euclid mission heritage was the top-model sam_ctrl, which reference all the other sub-systems from the Table 6, including already exercised sam_ctrl_dz sub-system from the section 7.4.1.

Table 9 summarize the sam_ctrl top-model general properties:

Property	Value
Referenced Library	yes
Referenced external sub-system	yes
Multi-dimensional ports	yes
Referenced WS parameters (indirectly via ref. sub-sys.)	yes
Ref. WS param. Types	Structure
	Matrix
Use Enums within the model	yes

Table 9 sam_crtl top-model properties

The following table summarizes the exercised state of each step from evaluation workflow 7.3.3 for sam_ctrl top model:

Workflow step	Performed	Status	Constrains / Issues	Fix
1. (D II.) **- Run Euclid Simulation** **- prepare inputs-output values**	yes	PASS	-	-
2. (C I.) **- Generate test scenario csv file (input values only)**	yes	PASS	-	-
3. (C I., A/B I., A II.) **- Generate QGen simulation model** **(run `qgen_make_sim_model`)**	yes	FAIL → PASS	E2_I1 Usage of Enums type prevent from generating the QGen simulation model (triggered from GUI and CLI) "Cannot call the constructor of 'SAM_STATE' outside of its enumeration block".	Significant update of the `qgen_make_sim_model.m` file by AdaCore (a wavefront release to v21.1).

Workflow step	Performed	Status	Constrains / Issues	Fix
4. (A II.) **- Run QGen simulation model**	yes	FAIL → PASS	E2_C1 "Model will not inherit sample time because the block 'SignalBuilder/ FromWs' disallows it"	Set: Simulation → Model Configuration Parameters → Solver → Periodic sample time constraint: Unconstrained
5. (E I., E II.) **- Compare Euclid vs QGen simulation input-output values**	no	-	-	-
6. (A III.) **Generate simulation log csv file**	yes	PASS	-	-
7. (B III.) **- generate system code** **- generate simulation code** **- build executable**	yes	PASS	-	-
8. (B III.) **- run the SIL executable** **- check the output report**	yes	PASS	-	-

Table 10 Top model sam_ctrl evaluation workflow states

The step 3. of the evaluation workflow 7.3.3 – the generation of the simulation model via `qgen_make_sim_model` CLI (or triggered from GUI without importing the input values via csv file) failed (E2_I1) for the sam_ctrl top model because of the usage of the enumeration types within the model.

The extensive AdaCore update of the `qgen_make_sim_model.m` file, as the wavefront release to the version 21.1, fixed the issues and simulation model could be successfully generated then.

Within the Signal Builder in QGen simulation model (A II.) were used for the Enum representation integer values. Therefore, the input part of the QGen simulation model (A II.) uses the 'convert' block, to cast the integer values back to the Enum representation before passing them in the model.

Note: the location of the `qgen_make_sim_model.m` file is `$QGEN_INSTALL/share/matlab`.

The previously mentioned file update also partially fixed the constraint E1_C2 from Exercise 1: sam_ctrl_dz. The simulation start and stop time were now propagated from the Signal Builder to the simulation model settings (C I., A II.). The simulation step was set to 'auto'.

The constrain E2_C1 and its status stays the same as constrain E1_C1 from exercise Exercise 1: sam_ctrl_dz.

All other steps from the evaluation workflow 7.3.3 passed without an issue. The problems with the model configuration parameters connection E1_I2 from the Exercise 1: sam_ctrl_dz did not reveal themselves – the parameters were connected correctly into the structures from `qgen_base_workspace.h` file.

It is important to mention, that even if the model from Exercise 1: sam_ctrl_dz, where the issue originally occurred, was referenced from the top-model in Exercise 2: sam_ctrl, the issue E1_I2 not happened this time.

The evaluation workflow step no. 5. was not performed with the same reasoning as mentioned in the exercise Exercise 1: sam_ctrl_dz.

7.4.3. Exercise 3: sam_ctrl_at

The third and last exercised model from the Euclid mission heritage was sam_ctrl_at.

Table 11 describes some of the general sam_ctrl_at model properties:

Property	Value
Referenced Library	yes
Referenced external sub-system	no
Multi-dimensional ports	yes
Referenced WS parameters	yes
Ref. WS param. Types	Structure
	Matrix
Use Enums within the model	no

Table 11 sam_ctrl_at model properties

Note: for this exercise an already receive wavefront release to the version 21.1 of the file `qgen_make_sim_model.m` was used (described in Exercise 2: sam_ctrl).

The following table summarizes the SIL results for the sam_crtl_at subsystem:

Workflow step	Performed	Status	Constrains / Issues	Fix
1. (D II.) **- Run Euclid Simulation** **- prepare inputs-output values**	yes	PASS	-	-
2. (C I.) **- Generate test scenario csv file (input values only)**	yes	PASS	-	-
3. (C I., A/B I., A II.) **- Generate QGen simulation model (run `qgen_make_sim_model`)**	yes	PASS	-	-
4. (A II.) **- Run QGen simulation model**	yes	FAIL → PASS	E3_I1 WS param. not referenced automatically	Add param. reference manually: ModelRef → Block Parameters →Arguments

Workflow step	Performed	Status	Constrains / Issues	Fix
4. (A II.) **- Run QGen simulation model**	yes	FAIL → PASS	E3_C1 "Model will not inherit sample time because the block 'SignalBuilder/ FromWs' disallows it"	Set: Simulation → Model Configuration Parameters → Solver → Periodic sample time constraint: Unconstrained
5. (E I., E II.) **- Compare Euclid vs QGen simulation input-output values**	yes	FAIL → PASS	E3_I2 outputs and the inputs were different in Euclid vs QGen simulation model	Update the csv file (C I.) self-generation procedure. Increase precision to 15 digits after decimal points, fix the order of the multidimensional signals.
6. (A III.) **Generate simulation log csv file**	yes	PASS	-	-
7. (B III.) **- generate system code** **- generate simulation code** **- build executable**	yes	PASS	-	-
8. (B III.) **- run the SIL executable** **- check the output report**	yes	FAIL → PASS	E3_I3 system model vs code outputs are outside of the acceptable tolerance	connect the same configuration parameters to the code simulation as were connected in model simulation

Table 12 sam_ctrl_at SIL result

The issue E3_I1 is the same as the issue E1_I1 in case of Exercise 1: sam_ctrl_dz. The model WS configuration parameters were not referenced in simulation model (A II.) and had to be linked manually.

The constrain E3_C1 is the same as the constrains E1_C1 and E2_C1 from the Exercise 1: sam_ctrl_dz respectively Exercise 2: sam_ctrl.

Let's skip the issue E3_I2 for now, since the evaluation workflow step no. 5 was introduced based on further issue.

From the Table 11 can be seen, that all remaining steps up to building a SIL executable passed. This was also the first impression of the overall status before truly running the SIL executable. The results were way off, far away from the expected tolerance (e.g.: expected: 4.269761517581880e+00, Found: 1.834243230426479e-312) as reported in Table 12 as issue **E3_I3**.

This phenomenon was the impulse to start investigating each evaluation workflow step transition. Based on this, the step no. 5 (E I., E II.) was introduced. This led to uncovering several issues with the self-written script for generating the csv file (C I.) with input values in it.

At first, the output precision had to be increased at least to 15 digits after decimal point.

Secondary, the rows and columns for the multidimensional signals were mixed-up and did not correspond to the description from section 7.3.2.

After those were fixed, the step no. 5 (E I., E II.) could pass.

Unfortunately, the csv file generation script update did not fix the issue **E3_I3** with the SIL execution results. And it makes perfect sense. The SIL examine only if the system model vs. system code produces the same output. If the csv (C I.) was corrupted, this would lead only to exercising unintentional test scenario, but should not have any effect on the model vs. code relation if the generation and all other steps went well.

After further investigation, the similar issue to the **E1_I2** from Exercise 1: sam_ctrl_dz was revealed. This time, the connection for the system configuration parameters were grammatically right (the wrong type casting did not happen this time – and therefore the simulation framework code with system code could be compiled without warning). But the linked parameters were again pointing to the empty uninitialized structure.

Therefore, the model simulation run with the manually setup system configuration parameters from the workspace whereas the code simulation run with some random configuration parameters. Based on that, in fact, two same systems were compared but with different configuration parameters and therefore, it is reasonable to expect, that theirs outputs could differs, as SIL properly reported.

After manual connection to the proper system configuration parameters from the `qgen_base_workspace.h` file, the SIL execution passed.

It is important to mention, that the sam_ctrl_at model was already indirectly exercised within the top model Exercise 2: sam_ctrl, where this issue was not observed.

7.5. Summary

Table 13 summarize the issues and constraints relevant for the QGen Debugger – SIL – (the self-created issues were skipped here – although they are described within the chapter 7) which were observed during the exercises on the models:

- Exercise 1: sam_ctrl_dz
- Exercise 2: sam_ctrl
- Exercise 3: sam_ctrl_at

Where:

-	means, not relevant or N/A
must be	means, the constraint must be set to the specific value
ERR	means, that the issue was observed within the exercise
OK	means, that the issue was not observed within the exercised

During the Exercise 2: sam_ctrl, the file `qgen_make_sim_model.m` was changed from v21.1 to the wavefront release version. Therefore, all issues and constrains, except the first one, are not relevant for the v21.1 within

the Exercise 2: sam_ctrl. This exercise was not after then performed with v21.1 of `qgen_make_sim_model.m` file but with the wavefront release version.

The Enums were used only within the Exercise 2: sam_ctrl, and therefore this issue (**E2_I1**) is not relevant for other two exercises.

The 'Periodic sample time constraint' was set to value 'unconstrained' for all exercises.

The wavefront release of `qgen_make_sim_model.m` file seems to fix the constraint (**E1_C2**) with the inheritance of the time-series from the Signal Builder into the simulation model settings.

The issue with the wrong system parameters connection (**E1_I2**, **E3_I2**) seems to occur only for some models (even with the wave front version of `qgen_make_sim_model.m` file). When it happened, it occurs within the 'QGen simulation model' and within the 'simulation framework code'.

The interesting thing is, that when the same model (Exercise 3: sam_ctrl_at, Exercise 1: sam_ctrl_dz) where the issue occurs is referenced from the top model (Exercise 2: sam_ctrl), the parameters seems to be connected correctly.

The root-cause of this was not investigated, but it is important to realize, that all the exercised models must be compatible with the QGen otherwise the code generation would not be possible. Therefore, the usage of an unsupported techniques could be rule out. At the same time, it cannot be clearly stated if this is a QGen issue (event though it looks likely to be) or if the issue was caused by some bad user (model) practice. On the other hand, the applied fixes are straight forward and easy to implement.

		qgen_make_sim_model.m			
		v21.1		wavefront release	
		1. sam_crtl_dz	2. sam_ctrl	3. sam_ctrl_at	
QGen simulation model	Sim. Model gen. failed - Enums prevent it	-	ERR	OK	-
	WS param. Not referenced automatically	ERR	-	OK	ERR
	Periodic sample time constraint: unconstrained	must be	-	must be	must be
	Start and Stop time not inherited from Signal Builder	ERR	-	OK	OK
Simulation framework code	Wrong type casting of system config. Param. (Init) - build failed	ERR	-	OK	OK
	Wrong connection of system config. Param. (Init) - empty struct.	ERR	-	OK	ERR

Table 13 QGen Debugger – SIL – evaluation – Issues/Constrains summary

The QGen toolchain is an active project under development, and therefore this exercises results can apply only for the specified version 21.1 of the QGen and the wavefront release of the `qgen_make_sim_model.m` file.

For the future exercises, e.g., the processor-in-the-loop (PIL), the version under evaluation shall be updated to the latest one available this day (for download), which is v21.2.

Possible re-execution of SIL exercises with the latest QGen version shall be considered, based on the Aurora project resources availability and priorities.

The AdaCore QGen team shall be informed about the observed issue with the system parameters connection on the simulation model and simulation code level.

The next exercised and evaluated tool shall be `qgenpil`. Based those results, the next version of this document shall be released.

8. QGen PIL - processor-in-the-loop

This feature will be exercised after MS4, thus, to be included in the next document release.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101004291