KRATES
inseneribüroo • engineering bureau

AdaCore
QGen

Integrating QGen in TASTE

IB Krates OÜ

Tõnu Näks

Talk outline

- An overview of the QGen qualifiable code generator
- QGen and TASTE integration
- Support for on-target testing
- Qualification concerns
- Upcoming developments

KRATES
inseneriburoo • engineering bureau

What is QGen?

A qualifiable and customizable code generator

from Simulink® and Stateflow® to SPARK and MISRA C

A formal model verifier

for runtime errors and functional properties

An open and extensible framework

to integrate heterogeneous models

KRATES
inseneriburoo • engineering bureau

QGen Development History

- Gene-Auto open source code generator (EUREKA project, 2006-2009)
- 11 partners, Leader: Continental Automotive France
- Project-P (France- and EU-funded collaborative R&D project, 2011-2015)
- 19 partners Leader: Continental Automotive France

KRATES
inseneriburoo • engineering bureau

QGen Development History

Industrial Users

Avionics

AIRBUS

Rockwell Collins

SAFRAN

THALES

Automotive

Continental

aboard Engineering

Space

ASTRIUM
AN EADS COMPANY

ThalesAlenia Space
A Thales / Finmeccanica Company

PROJECT P

Tech Providers

scilab enterprises

AdaCore

Atos

KRATES
inseneribüroo · engineering bureau

Altair

STInformatique SERVICES

acg SOLUTIONS

Academia

IRIT
CNRS · INPT · UPS · UT1 · UT2J

ONERA
THE FRENCH AEROSPACE LAB

Lab-STICC

Inria
INVENTEURS DU MONDE NUMÉRIQUE

École des Ponts ParisTech

KRATES
inseneribüroo · engineering bureau

QGen Main Features

Support for a large subset of Simulink®

- **Around 120 blocks, optional checks for MISRA Guidelines for Simulink®**
- **A safe subset of Stateflow® supported**

KRATES
inseneriburoo • engineering bureau

QGen Main Features

Support for a large subset of Simulink®

Code generation producing MISRA C and SPARK (formally provable subset of Ada 2012)

- **Readable and traceable code, no performance penalty**

KRATES
inseneriburoo • engineering bureau

QGen Main Features

- Support for a large subset of Simulink®
- Code generation producing MISRA C and SPARK (formally provable subset of Ada 2012)
- Integrated with compilation and testing frameworks
  - Integration with GNAT Pro compiler for qualified, end-to-end tool chain
  - Integration with GNATemulator and GNATcoverage for structural coverage analysis (up to MC/DC) without code instrumentation executing embedded object code

KRATES
inseneriburoo • engineering bureau

QGen Main Features

- Support for a large subset of Simulink®
- Code generation producing MISRA C and SPARK (formally provable subset of Ada 2012)
- Integrated with compilation and testing frameworks
- Includes a static model verifier
  - Focus on safety-critical systems
  - Finds Run-time errors (divisions by zero, overflows, …)
  - Finds Logical errors (dead execution paths)
  - Verifies Functional/safety properties (Simulink® assertion blocks)

KRATES
inseneriburoo • engineering bureau

QGen Main Features

- Support for a large subset of Simulink®
- Code generation producing MISRA C and SPARK (formally provable subset of Ada 2012)
- Integrated with compilation and testing frameworks
- Includes a static model verifier
- With QGen, you do not need a separate Stateflow modeling standard

Stateflow Safe subset built in: MISRA AC SLSF (guidelines)

- No model-level data, no overloading (Required)
- Single default unguarded transition per state (Required)
- Constrained mixing of Mealy and Moore semantics (Required)
- No backtracking (Required)
- No history junction (Required)

KRATES
inseneribüroo • engineering bureau

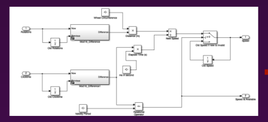
QGen Main Features

- Support for a large subset of Simulink®
- Code generation producing MISRA C and SPARK (formally provable subset of Ada 2012)
- Integrated with compilation and testing frameworks
- Includes a static model verifier
- With QGen, you do not need a separate Stateflow modeling standard
- Off-the-shelf qualification material
  - Including validation against Simulink® simulation
  - DO-178C, EN 50128, ISO-26262 TCL3

KRATES
inseneriburoo • engineering bureau

QGen: An open and extensible framework

Traditional Code Generators

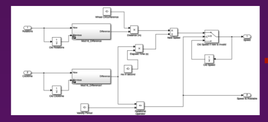


Simulink Model

Black Box

Source Code

QGen



Simulink Model

Access to intermediate representations

Modeling standard checking

Integration with UML

Additional verification

Makefile generation

Extract traceability data

Processor customization

Source Code

KRATES
inseneribüroo • engineering bureau

QGen: An open and extensible framework

“The gcc for modeling languages”

- QGen is designed to accept multiple languages in input, including in-house DSLs
- A single code generation style/strategy for all of your modeling languages
- XML-based model import at different abstraction levels
- Design model: blocks, signals, states, transitions, ….
- Code model: variables, statements

Selective model compilation

- QGen is composed of multiple model compilation steps
- It is possible to execute them selectively
- It is possible to import/export XML at any step

KRATES
inseneribüroo • engineering bureau

QGen highlights

Safe Simulink subset: ensure that a model is verifiable by construction

Decrease tool deployment costs and system verification costs

Complete qualification evidence

Decrease verification costs on generated code

Easy to customize

Decrease tool adaptation costs

Can be used to integrate with custom libraries, or other modeling languages such as UML, SysML, ...

Integrated with verification, compilation and testing tools

Decrease tool integration costs

More practical to use

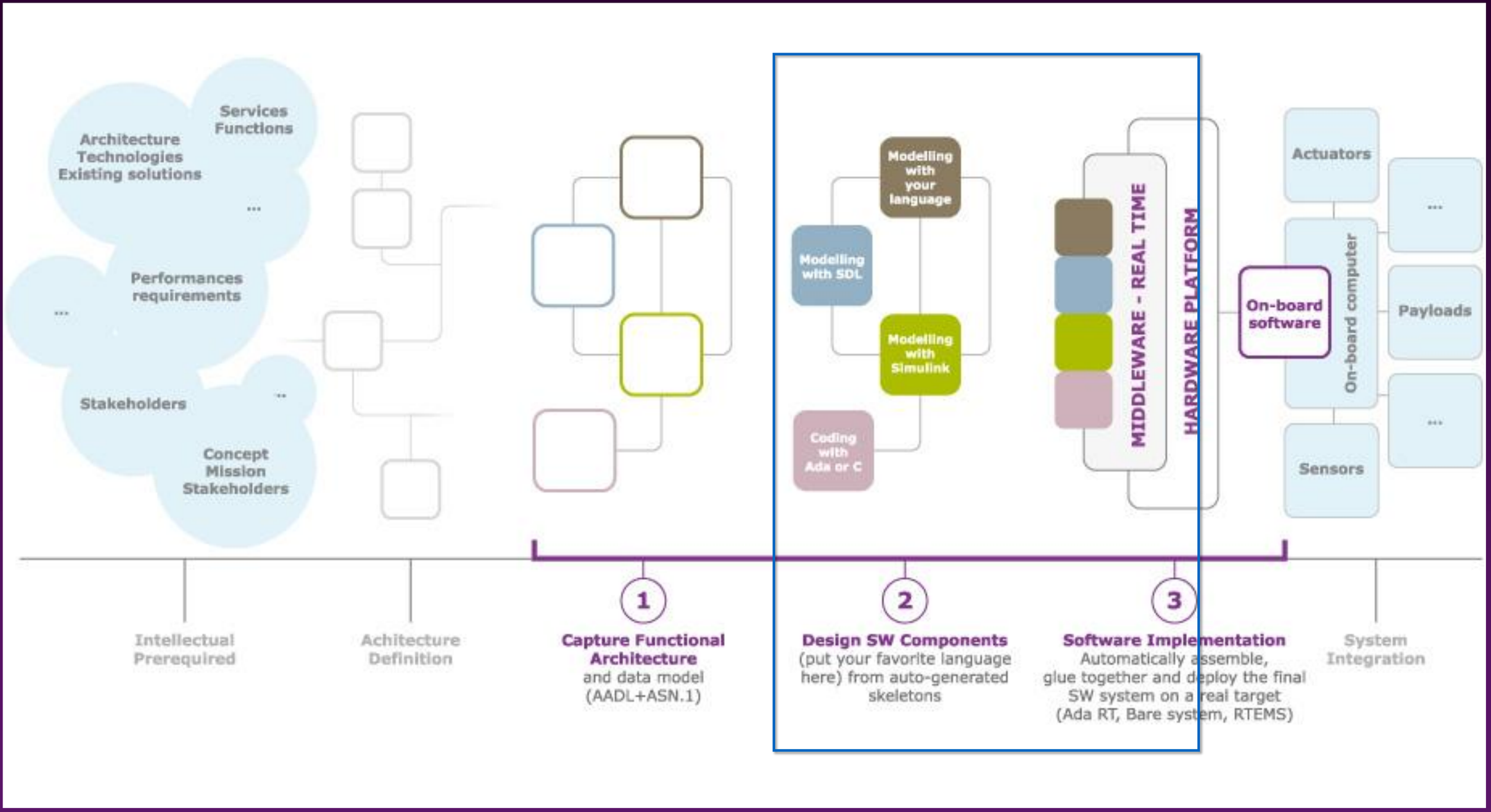
Does not require Matlab after model is exported

10x faster code generation, with no performance penalties on the generated code

KRATES
inseneriburoo • engineering bureau

QGen in TASTE

KRATES
inseneriburoo • engineering bureau

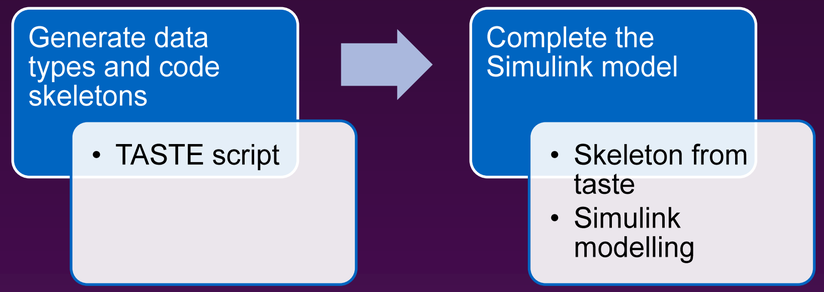

Services
Functions
Architecture
Technologies
Existing solutions
...
Performances
requirements
...
Stakeholders
..
Concept
Mission
Stakeholders
Modelling with your language
Modelling with SDL
Modelling with Simulink
Coding with Ada or C
MIDDLEWARE - REAL TIME
HARDWARE PLATFORM
On-board software
Actuators
...
On-board computer
Payloads
...
Sensors
Intellectual Prerequired
Achitecture Definition
1
Capture Functional Architecture
and data model (AADL+ASN.1)
2
Design SW Components
(put your favorite language here) from auto-generated skeletons
3
Software Implementation
Automatically assemble, glue together and deploy the final SW system on a real target (Ada RT, Bare system, RTEMS)
System Integration

Code generation with Simulink

Generate data types and code skeletons

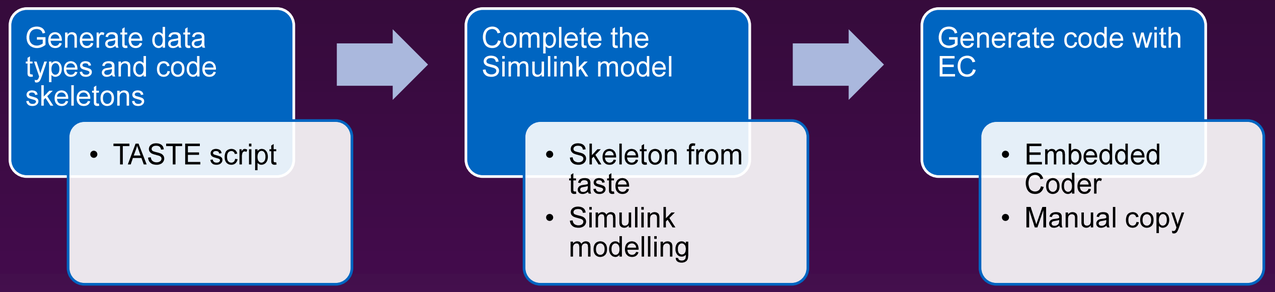
- TASTE script

Code generation with Simulink



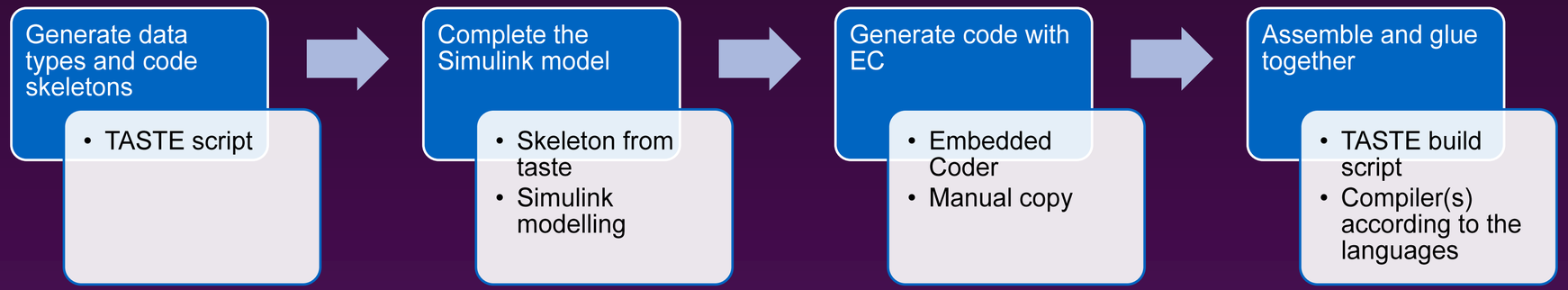
KRATES
inseneriburoo • engineering bureau

Code generation with Simulink



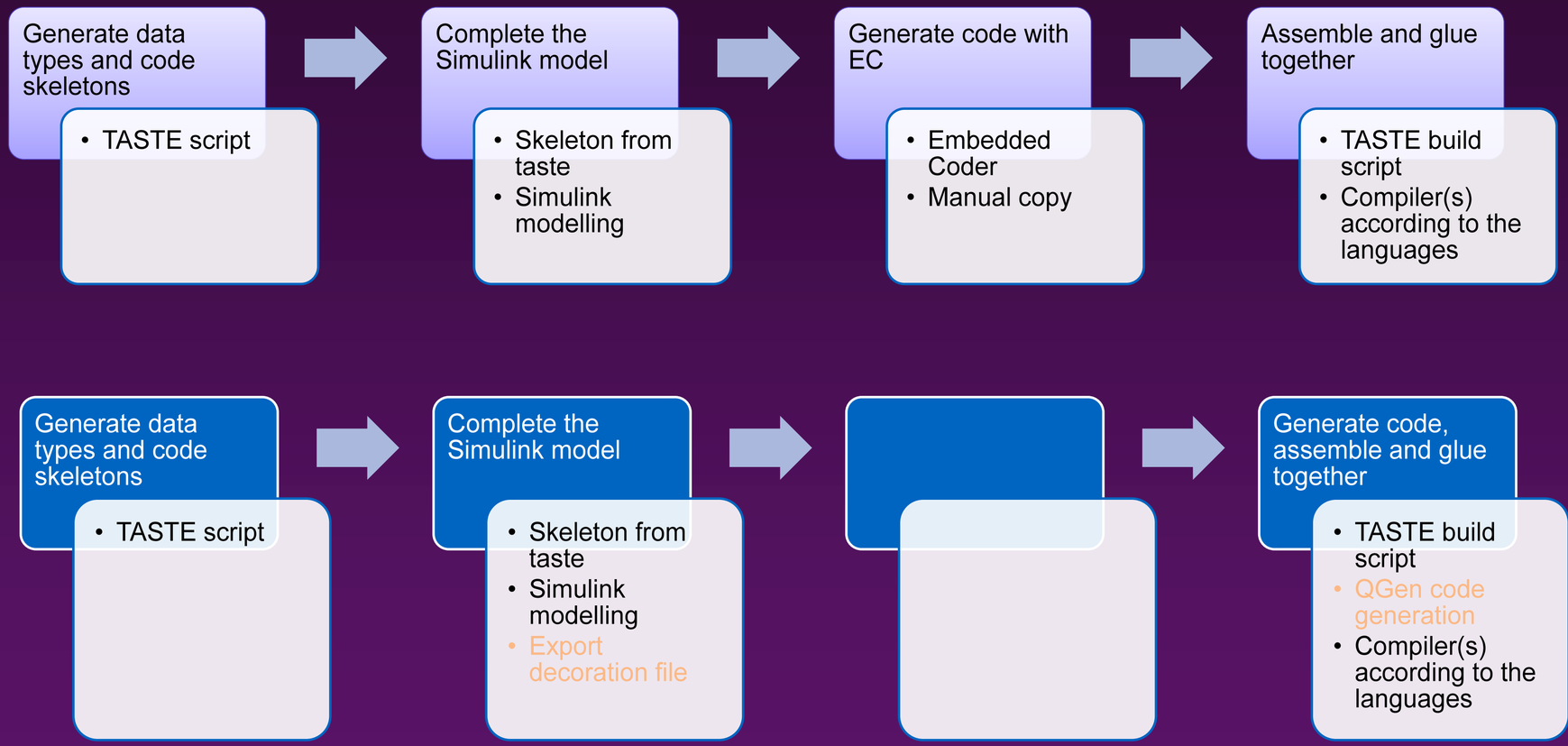
KRATES
inseneriburoo • engineering bureau

Code generation with Simulink



KRATES
inseneribüroo • engineering bureau

Code generation with QGen



KRATES
inseneribüroo • engineering bureau

Interfacing TASTE-Simulink

```
void scheduler_RI_Controller4_C(asn1SccMyReal *IN_sv,
                                asn1SccMyReal *IN_pv,
                                asn1SccMyReal *OUT_mv)
{
  /* Buffer(s) to put the encoded input parameter(s) */
  static char IN_buf_sv[sizeof(asn1SccMyReal)] = {0};
  int size_IN_buf_sv=0;
  static char IN_buf_pv[sizeof(asn1SccMyReal)] = {0};
  int size_IN_buf_pv=0;

  /* Buffer(s) for the output parameter(s) */
  static char OUT_buf_mv[sizeof(asn1SccMyReal)];
  int size_OUT_buf_mv=0;

  /* Encode each input parameter */
  size_IN_buf_sv=Encode_NATIVE_MyReal(IN_buf_sv, sizeof(asn1SccMyReal), IN_sv);
  if (-1 == size_IN_buf_sv) {
    exit (-1);
  }
  size_IN_buf_pv=Encode_NATIVE_MyReal(IN_buf_pv, sizeof(asn1SccMyReal), IN_pv);
+ if (-1 == size_IN_buf_pv) {
  /* Call to VM callback function */
  vm_scheduler_Controller4_C(IN_buf_sv, size_IN_buf_sv,
                             IN_buf_pv, size_IN_buf_pv,
                             OUT_buf_mv, &size_OUT_buf_mv);

  /* Decode each output parameter */
+ if (0 != Decode_NATIVE_MyReal(OUT_mv, OUT_buf_mv, size_OUT_buf_mv)) {
}
```

KRATES
inseneribüroo • engineering bureau

Interfacing TASTE-Simulink

```
void scheduler_RI_Controller4_C(asn1SccMyReal *IN_sv,
                                asn1SccMyReal *IN_pv,
                                asn1SccMyReal *OUT_mv)
{
  /* Buffer(s) to put the encoded input parameter(s) */
  static char IN_buf_sv[sizeof(asn1SccMyReal)] = {0};
  int size_IN_buf_sv=0;
  static char IN_buf_pv[sizeof(asn1SccMyReal)] = {0};
  int size_IN_buf_pv=0;

  /* Buffer(s) for the output parameter(s) */
  static char OUT_buf_mv[sizeof(asn1SccMyReal)];
  int size_OUT_buf_mv=0;

  /*
  siz
  if
  }
  siz
  if
  /* Call to VM callback function */
  vm_scheduler_Controller4_C(IN_buf_sv, size_IN_buf_sv,
                             IN_buf_pv, size_IN_buf_pv,
                             OUT_buf_mv, &size_OUT_buf_mv);

  /* Decode each output parameter */
  if (0 != Decode_NATIVE_MyReal(OUT_mv, OUT_buf_mv, size_OUT_buf_mv)) {
  }
```

```
void pid_c_Controller4_C(void *psv, size_t size_sv,
                         void *ppv, size_t size_pv, void *pmv, size_t *pSize_mv)
{
    Convert_From_native_To_MyReal_In_Controller4_C_Simulink_sv(psv, size_sv);
    Convert_From_native_To_MyReal_In_Controller4_C_Simulink_pv(ppv, size_pv);
    Execute_Controller4_C_Simulink();
    *pSize_mv = Convert_From_MyReal_To_native_In_Controller4_C_Simulink_mv(pmv, 16);
}
```

KRATES
inseneribüroo • engineering bureau

Interfacing TASTE-QGen

```
void scheduler_RI_Controller4_C(asn1SccMyReal *IN_sv,
                                asn1SccMyReal *IN_pv,
                                asn1SccMyReal *OUT_mv)
{
  /* Buffer(s) to put the encoded input parameter(s) */
  static char IN_buf_sv[sizeof(asn1SccMyReal)] = {0};
  int size_IN_buf_sv=0;
  static char IN_buf_pv[sizeof(asn1SccMyReal)] = {0};
  int size_IN_buf_pv=0;

  /* Buffer(s) for the output parameter(s) */
  static char OUT_buf_mv[sizeof(asn1SccMyReal)];
  int size_OUT_buf_mv=0;

  /*
  siz
  if
  }
  siz
  if
  /* Call to VM callback function */
  vm_scheduler_Controller4_C(IN_buf_sv, size_IN_buf_sv,
                             IN_buf_pv, size_IN_buf_pv,
                             OUT_buf_mv, &size_OUT_buf_mv);

  /* Decode each output parameter */
  if (0 != Decode_NATIVE_MyReal(OUT_mv, OUT_buf_mv, size_OUT_buf_mv)) {
}
```

```
void pid_c_Controller4_C(void *psv, size_t size_sv,
                         void *ppv, size_t size_pv, void *pmv, size_t *pSize_mv)
{
    Convert_From_native_To_MyReal_In_Controller4_C_Simulink_sv(psv, size_sv);
    Convert_From_native_To_MyReal_In_Controller4_C_Simulink_pv(ppv, size_pv);
    Execute_Controller4_C_Simulink();
    *pSize_mv = Convert_From_MyReal_To_native_In_Controller4_C_Simulink_mv(pmv, 16);
}
```

```
void scheduler_RI_Controller4_C(asn1SccMyReal *IN_sv,
                                asn1SccMyReal *IN_pv, asn1SccMyReal *OUT_mv)
{

  /* Call QGenc function */
  Controller4_C_comp(*IN_sv, *IN_pv, OUT_mv);

}
```

KRATES
inseneribüroo • engineering bureau

QGen TASTE highlights

Code generation driven by a single build script

Reduce manual steps

Fully repeatable process

KRATES
inseneriburoo • engineering bureau

QGen TASTE highlights

Code generation driven by a single build script

Reduce manual steps

Fully repeatable process

Interfacing with native data types

Reduces the number of required buffers

Cleaner glue code

KRATES
inseneriburoo • engineering bureau

QGen TASTE highlights

Code generation driven by a single build script

Reduce manual steps

Fully repeatable process

Interfacing with native data types

Reduces the number of required buffers

Cleaner glue code

Code generation with verification support

CodePeer and SPARK integration

KRATES
inseneriburoo • engineering bureau

QGen TASTE highlights

Code generation driven by a single build script

Reduce manual steps

Fully repeatable process

Interfacing with native data types

Reduces the number of required buffers

Cleaner glue code

Code generation with verification support

CodePeer and SPARK integration

Extensible to other input languages

Scicos, SDL ...

KRATES
inseneribüroo • engineering bureau

Validation on a case study

KRATES
inseneriburoo • engineering bureau

Integrated Motor Control Mechanisms

- A model developed by S.A.B.C.A.
- Describes a control system with three control loops (current, speed, pace) running periodically with different frequencies
- Complete model describing the SW architecture, control law and a relevant part of the hardware
- Done in the study
  - Extracting the achitectural part and implementing it in TASTE
  - Encapsulating the control loops as atomic subsystems and generating code with QGen
  - Comparing the simulation in TASTE with that in Simulink

KRATES
inseneriburoo • engineering bureau

On-target regression testing

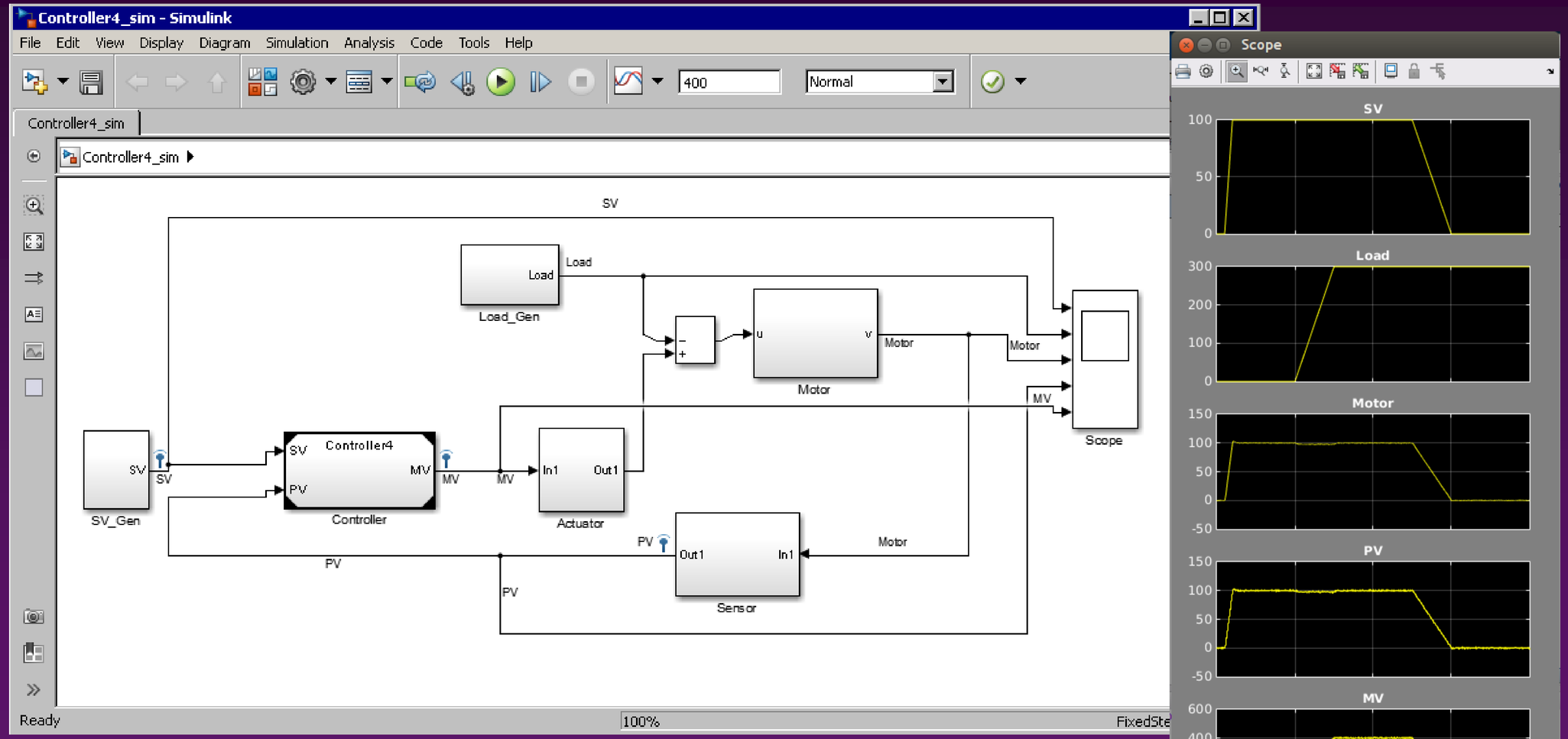
KRATES
inseneriburoo • engineering bureau

Run module tests on target processor without the need of connecting the IO or integrating with other modules

Process

- Validate the model on Simulink by using the Plant model to generate expected IO responses
- Use QGen for generating the embedded code
- Use Qgen_test to encapsulate reference data from simulation and generate test harness
- Add monitoring interface in TASTE to communicate with the test harness
- Run the test in target and view the results in TASTE or save to file

Validate the model in Simulink



Export test vectors

```
>> sim_and_log('Controller4_sim.mdl', '.')
-------------------------------------------
Simulating: Controller4_sim.mdl
Warning: The model 'Controller4_sim' has the 'Configuration Parameters' > 'Data Import/Export'
> 'Signal logging format' parameter set to 'ModelDataLogs'.  The signal logging save format
'ModelDataLogs' will be removed in a future release.  To take advantage of new functionality,
update models that use 'ModelDataLogs' signal logging format to use the 'Dataset' format.  For
help with resolving this and other upgrade issues, use the Simulink Upgrade Advisor.
> In sim_and_log at 18
------------------
// MV;PV;SV;
The log data is stored in file ./Controller4.output.sl.txt !
SimAndLog Done
```

KRATES
inseneribüroo • engineering bureau

Export test vectors

```
>> sim_and_log('Controller4_sim.mdl', '.')
------------------------------------------
Simulating: Controller4_sim.mdl
Warning: The model 'Controller4_sim' has the 'Configuration Parameters' > 'Data Import/Export'
> 'Signal logging format' parameter set to 'ModelDataLogs'.  The signal logging save format
'ModelDataLogs' will be removed in a future release.  To take ac
update models that use 'ModelDataLogs' signal logging format to
help with resolving this and other upgrade issues, use the Simul
> In sim_and_log at 18
------------------
// MV;PV;SV;
The log data is stored in file ./Controller4.output.sl.txt !
SimAndLog Done
```

1 Controller4.output.sl.txt

```
1   //·MV;PV;SV;
2   -7.27929240262787E-01;2.52096706584515E-01;0.00000000000000E+00
3   -5.59439400961325E+00;1.67335115920176E+00;0.00000000000000E+00
4   -3.02339274880996E+00;-7.29981334055796E-01;0.00000000000000E+00
5   -4.24322698916446E-01;7.28317946785802E-01;0.00000000000000E+00
6   -2.39737794470434E+00;-4.65929491077388E-02;0.00000000000000E+00
7   1.06425068631002E+00;-5.02977574667044E-01;0.00000000000000E+00
8   3.47046942976240E+00;-8.53745432721018E-01;0.00000000000000E+00
9   2.30244264513166E+00;-3.38602766076177E-02;0.00000000000000E+00
10  -9.77615117714633E-01;3.24472246198110E-01;0.00000000000000E+00
11  -1.33812687464584E+00;7.71536146867599E-02;0.00000000000000E+00
12  -1.13801291067426E+00;2.36043667897418E-01;0.00000000000000E+00
13  -1.55301201893532E+00;2.05962093268831E-01;0.00000000000000E+00
14  -1.50125526145890E-01;-2.70852361522421E-01;0.00000000000000E+00
15  -1.60764542174370E+00;7.13820901055039E-01;0.00000000000000E+00
16  -8.16046231698702E-01;-5.66093007143453E-01;0.00000000000000E+00
17  6.17283345355210E+00;-1.71360499354709E+00;0.00000000000000E+00
18  8.26182531143153E+00;-1.18099634317271E+00;0.00000000000000E+00
19  -2.08865448571224E-01;1.35780572292104E+00;0.00000000000000E+00
20  -2.97018292522609E+00;-2.33115412630904E-01;0.00000000000000E+00
21  2.72371783811945E+00;-6.67664642796856E-01;0.00000000000000E+00
22  3.70099289673455E+00;-5.28671361027018E-01;0.00000000000000E+00
23  3.06989272447473E+00;-3.92133431442536E-01;0.00000000000000E+00
```

KRATES
inseneribüroo • engineering bureau

Generate test harness

```
assert@assertvm:$./create_sim_harness.py --model demo_model/Controller4.mdl --language ada --outp
ut Controller4_test --clean
Cleaning output directory Controller4_test ...
Copying QGen sim support files ...
Generating test harness for simulation test ...
QGENC_SIM arguments:
['--incremental', 'demo_model/Controller4.mdl', '--output', 'Controller4_test/Controller4/hmc_sim
', '--language', 'ada', '--debug', '--typing', 'demo_model/Controller4_types.txt']
[INFO] :Preprocessing model Controller4
[INFO] :Finished preprocessing model Controller4 ID= 0
[INFO] :Sequencing model
[INFO] :Finished sequencing model Controller4
[INFO] :Generating code model Controller4
[INFO] :Finished code model generation  Controller4
[INFO] :Generating simulation harness
[INFO] :Printing to Ada Controller4
[INFO] :Printing to Ada Controller4 ID= 6
[INFO] :Finished Ada printing Controller4 ID= 6
[INFO] :Finished simulation harness generation Controller4
Generating code for the target model ...
QGENC arguments:
['--clean', 'demo_model/Controller4.mdl', '--output', 'Controller4_test/Controller4/hmc_sim/src',
 '--language', 'ada', '--debug', '--typing', 'demo_model/Controller4_types.txt']
```

KRATES
inseneribüroo • engineering bureau

Generate test harness

```
assert@assertvm:$./create_sim_harness.py --model demo_model/Controller4.mdl --language ada --outp
ut Controller4_test --clean
Cleaning output directory Controller4_test ...
Copying QGen sim support files ...
Generating test harness for simulation test ...
QGENC_SIM arguments:
['--incremental', 'demo_model/Controller4.mdl', '--outpu
', '--language', 'ada', '--debug', '--typing', 'demo_mod
[INFO] :Preprocessing model Controller4
[INFO] :Finished preprocessing model Controller4 ID= 0
[INFO] :Sequencing model
[INFO] :Finished sequencing model Controller4
[INFO] :Generating code model Controller4
[INFO] :Finished code model generation  Controller4
[INFO] :Generating simulation harness
[INFO] :Printing to Ada Controller4
[INFO] :Printing to Ada Controller4 ID= 6
[INFO] :Finished Ada printing Controller4 ID= 6
[INFO] :Finished simulation harness generation Controlle
Generating code for the target model ...
QGENC arguments:
['--clean', 'demo_model/Controller4.mdl', '--output', 'C
 '--language', 'ada', '--debug', '--typing', 'demo model
```

```
procedure Init is
   use type Integer_32;
   use type String;
   Target_Language : constant String := "ada";
   Test_Points : constant Integer_32 := 3;
   Test_Length : constant Integer_32 := 4001;
begin
   Controller4.init;
   HiMoCo_TASTE_Sim_Support.Sim_Test (Target_Language,
                                      Test_Points, Test_Length);
end Init;

procedure Run_Test_Step (I : Integer_32; Test_Ada : Boolean) is
   use type Boolean;
   use type Integer_32;
   use type Long_Float;
   MV : Long_Float;
   PV : Long_Float;
   SV : Long_Float;
begin
   PV := Simulation_Data.Sim_Vector (Integer (I), 1);
   SV := Simulation_Data.Sim_Vector (Integer (I), 2);
   Controller4.comp (SV, PV, MV);
   Simulation_Data.Test_Vector (Integer (I), 0) := MV;
   Simulation_Data.Test_Vector (Integer (I), 1) := PV;
   Simulation_Data.Test_Vector (Integer (I), 2) := SV;
end Run_Test_Step;
end Simulation;
```

KRATES
inseneribüroo • engineering bureau

Generate test harness

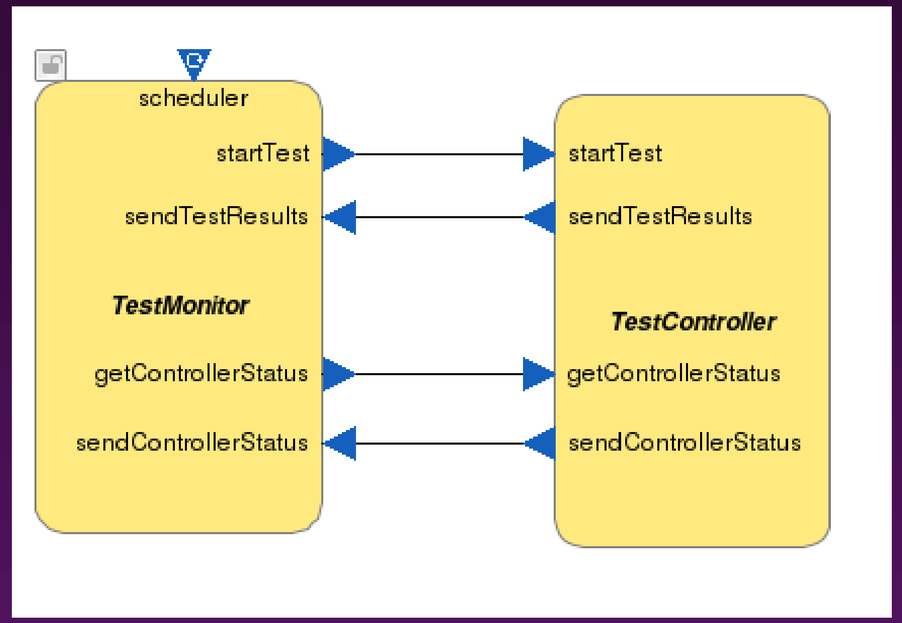
```
assert@assertvm:$./create_sim_harness.py --model demo_model/Controller4.mdl --language ada --outp
ut Controller4_test --clean
Cleaning output directory Con
Copying QGen sim support file
Generating test harness for s
QGENC_SIM arguments:
['--incremental', 'demo_model
', '--language', 'ada', '--de
[INFO] :Preprocessing model C
[INFO] :Finished preprocessin
[INFO] :Sequencing model
[INFO] :Finished sequencing m
[INFO] :Generating code model
[INFO] :Finished code model g
[INFO] :Generating simulation
[INFO] :Printing to Ada Contr
[INFO] :Printing to Ada Contr
[INFO] :Finished Ada printing
[INFO] :Finished simulation h
Generating code for the targe
QGENC arguments:
['--clean', 'demo_model/Contr
 '--language', 'ada', '--debu
```

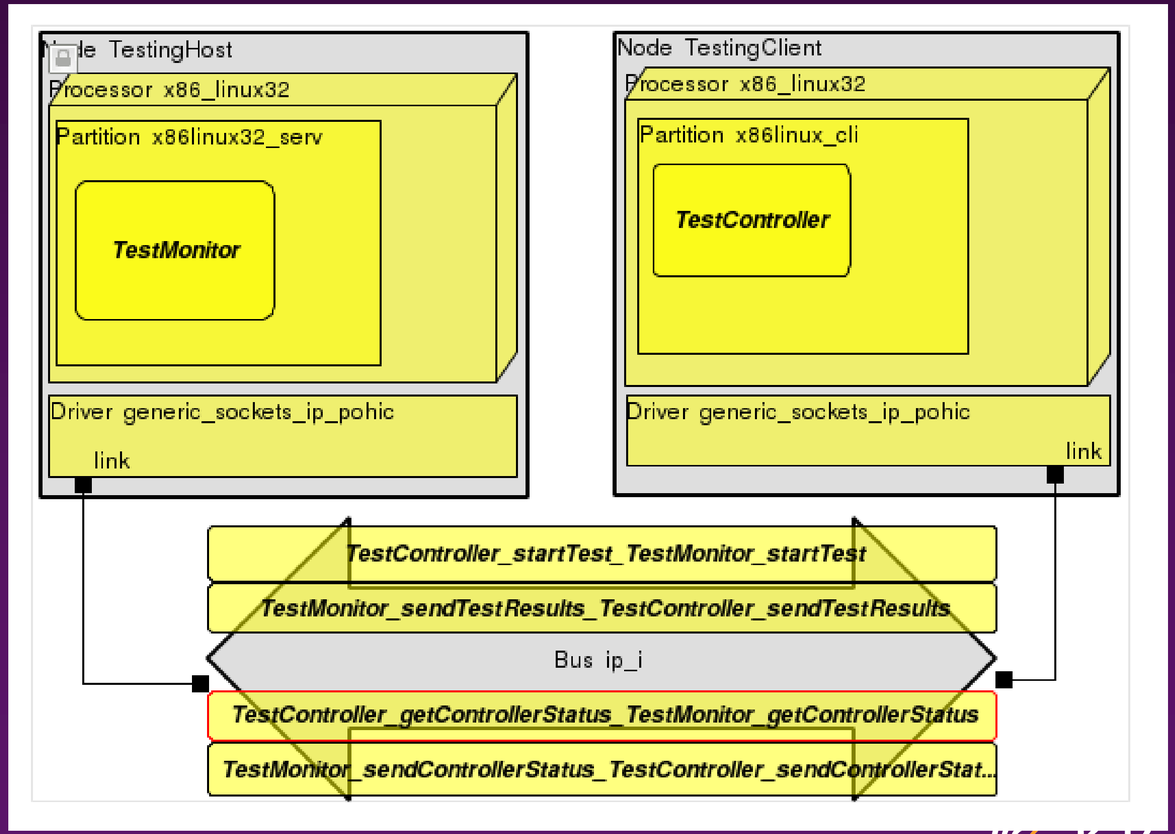
```
procedure Init is
   use type Integer_32;
   use type String;
   Target_Language : constant String := "ada";
   Test_Points : constant Integer_32 := 3;
   Test_Length : constant Integer_32 := 4001;
begin
   Controller4.init;
   HiMoCo_TASTE_Sim_Support.Sim_Test (Target_La
                                      Test_P
end Init;

procedure Run_Test_Step (I : Integer_32; Test_Ac
   use type Boolean;
   use type Integer_32;
   use type Long_Float;
   MV : Long_Float;
   PV : Long_Float;
   SV : Long_Float;
begin
   PV := Simulation_Data.Sim_Vector (Integer (I), 1)
   SV := Simulation_Data.Sim_Vector (Integer (I), 2)
   Controller4.comp (SV, PV, MV);
   Simulation_Data.Test_Vector (Integer (I), 0) := M
   Simulation_Data.Test_Vector (Integer (I), 1) := P
   Simulation_Data.Test_Vector (Integer (I), 2) := S
end Run_Test_Step;
end Simulation;
```

```
-------------
--Run_Test--
-------------

procedure Run_Test (Is_Ada      : Boolean;
                    Test_Length : Integer;
                    Test_Points : Integer)
is

   Start_Time     : Time;
   Sim_Dur        : Duration;
   Avg_Cycle_Time : Duration;

begin
   --Measure the test execution time
   Start_Time := Clock;

   --Main test loop
   for I in 0 .. Test_Length - 1 loop
      Run_Test_Step (Integer_32 (I), Is_Ada);
   end loop;

   Sim_Dur := Clock - Start_Time;
   Avg_Cycle_Time := Sim_Dur / Test_Length;

   Tot_Exec_Time := Sim_Dur;
   Avg_Exec_Time := Avg_Cycle_Time;

   --Compare ada sim results
   Compare_Test_Results (Test_Length,
                         Test_Points);
end Run_Test;
```

KRATES
inseneribüroo • engineering bureau

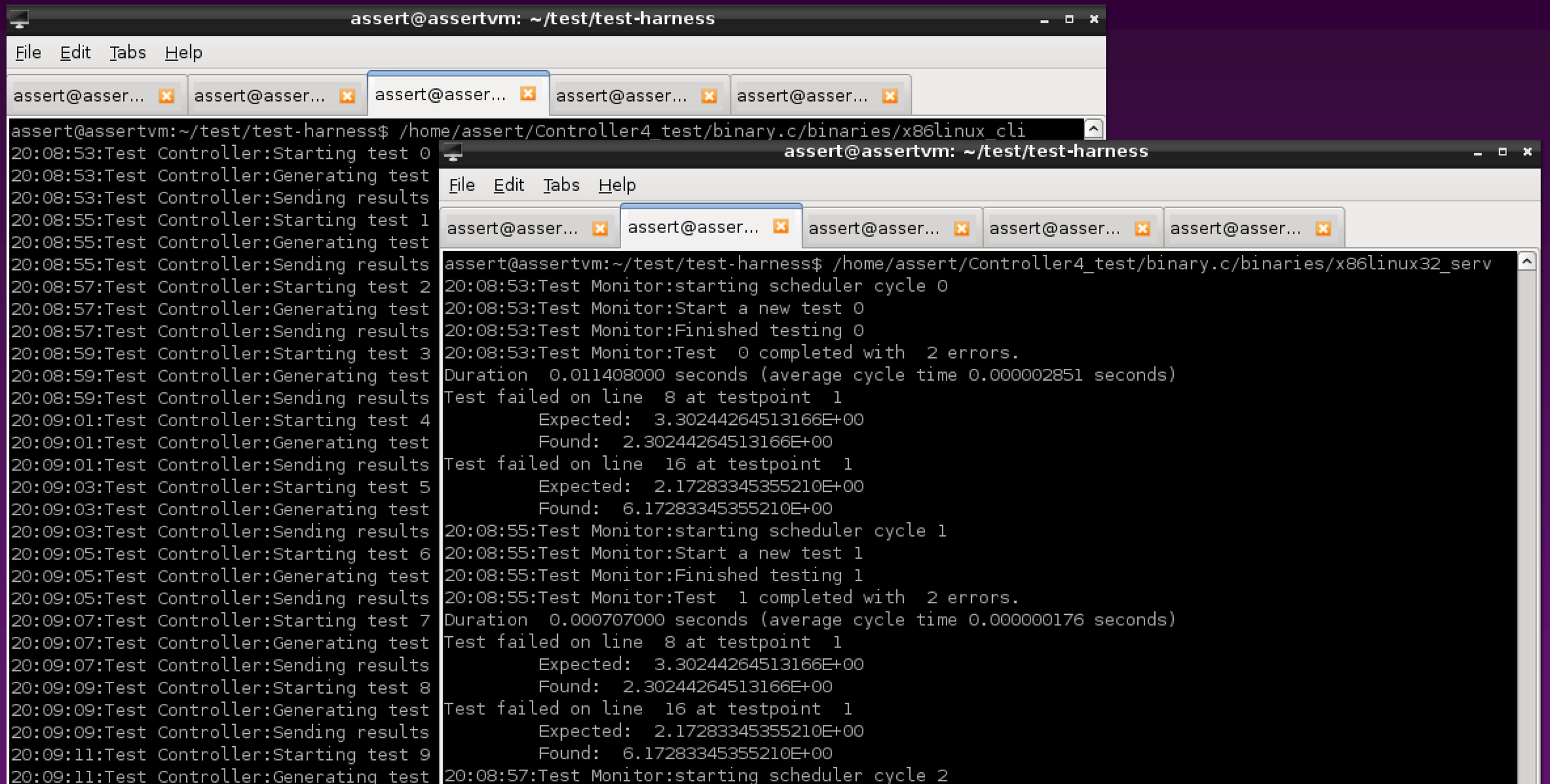
Compose test monitor in TASTE





KRATES
insenieriburoo • engineering bureau

Copy to target and launch



QGen testing support highlights

Simple test vector export

The same format used in the regression testsuite of QGen and for testing the generated code

KRATES
inseneriburoo • engineering bureau

QGen testing support highlights

Simple test vector export

The same format used in the regression testsuite of QGen and for testing the generated code

Unit testing in target environment

Reliable check of numeric precision, calls to native API.

Performance measurement

KRATES
inseneriburoo • engineering bureau

QGen testing support highlights

Simple test vector export

The same format used in the regression testsuite of QGen and for testing the generated code

Unit testing in target environment

Reliable check of numeric precision, calls to native API.

Performance measurement

Monitoring through TASTE

Standard communication routines generated by taste

Possibility to visualise with TASTE GUI

KRATES
inseneriburoo • engineering bureau

QGen qualification

KRATES
inseneribüroo • engineering bureau

QGen and DO-178C / DO-330

DO-178C (Software Considerations in Airborne Systems and Equipment Certification)

DO-330 (Software Tool Qualification Considerations)

- Tool Operational Requirements (TOR) - define the tool's functionality and interface from the perspective of the tool user's software life cycle
- COTS tools such as QGen don't have a single user / software life cycle / TOR

DO-330 11.3 „Qualifying COTS tools“

- Work-sharing between the tool *developer* and *user* for the qualification of the tool
- Generic Developer Tool Operational Requirements (D-TOR) are identified by the *developer*

The *developer*

- develops, verifies and *pre*-validates the tool based on D-TOR

Each *user* must

- develop their own TOR based on D-TOR; ensure that TOR are complete, accurate and correct; ensure that software life cycle needs are met by the tool; ensure that tool operation complies with TOR in the *user environment*

KRATES
inseneriburoo • engineering bureau

QGen and DO-330

Developer activities

- Define Developer Tool Operational Requirements (D-TOR):

  Tool interfaces, supported Simulink blocks and configurations,

  supported Stateflow features, semantics (expected behaviour of generated code)

- Define Tool Requirements (TR):

  Generated code patterns

User activities

- Define TOR based on D-TOR; ensure that TOR are complete, accurate and correct; ensure that software life cycle needs are met by the tool; ensure that tool operation complies with TOR in the *user environment*
- Include compiling and running a subset of tool test outputs (generated code) in the user environment and comparing with the expected simulation outputs (part of D-TOR).
- The activity is minimal in case the D-TOR fully satisfy the *user* needs and the given environment is already pre-validated by the *developer*

KRATES
inseneriburoo • engineering bureau

QGen TQL-1 Qualkit

A modeling standard compliant with DO-331

Verified Tool Operational Requirements (D-TOR) and Tool Requirements (TR) defining the transformations from model to code

A testsuite for validating the D-TOR in user environment

Integration with AdaCore GNATemulator and GNATcoverage allows for Processor-In-Loop unit testing that is as fast and practical as MIL, while gathering structural coverage data on un-instrumented code

Full traceability from each modelling construct to corresponding code element.

=>

KRATES
inseneribüroo • engineering bureau

QGen TQL-1 Qualkit

Certification credit for

- **Verification activities on source code: remove review/analysis of generated code**
- **Verification activities on object code: remove unit testing**

QGen provides TQL1 qualification kit which can be validated by AdaCore when using AdaCore compilers. Otherwise, validation can be done by applicant.

Removal of verification activities on source code, unit testing, structural coverage

KRATES
inseneriburoo • engineering bureau

QGen and ECSS

KRATES
inseneriburoo • engineering bureau

QGen, DO/ED and ECSS

ECSS compliant software development is governed by ECSS-E-ST-40C and ECSS-Q-ST-80C

- Oriented at the whole process of space software development
- Some guidance for ACG *usage*
- Terms OTS, COTS, MOTS are defined, but no specific guidance for *tool qualification*

However, generally the applicable requirements in DO-178C/ED-12C, DO-330/ED-215 and ECSS are compatible

- A mapping between the main requirements of the DO/ED and ECSS standards for ACG tools such as QGen and the tayloring of ECSS-Q-ST-80C for QGen is provided as a result of the current project

KRATES
inseneribüroo • engineering bureau

QGen and DO-178C / DO-330

DO-330 (Tool Qualification Document)

- **Precise identification of certification credit for code generator qualification**
- **Identification of credit w.r.t qualification strategy (TQL1 vs TQL5)**
- **See DO-330, FAQ D.8, in particular scenario 3 on section 1.8.3.2.1**

	Objective		Activity
	Description	**Ref**	**Ref**
1	Source Code complies with low-level requirements.	6.3.4.a	6.3.4
2	Source Code complies with software architecture.	6.3.4.b	6.3.4
3	Source Code is verifiable.	6.3.4.c	6.3.4
4	Source Code conforms to standards.	6.3.4.d	6.3.4
5	Source Code is traceable to low-level requirements.	6.3.4.e	6.3.4

3	Executable Object Code complies with low-level requirements.	6.4.c	6.4.2 6.4.2.1 6.4.3 6.5
4	Executable Object Code is robust with low-level requirements.	6.4.d	6.4.2 6.4.2.2 6.4.3 6.5

KRATES
inseneribüroo • engineering bureau

QGen and ECSS-E-ST-40C / ECSS-Q-ST-80C

- **Coding credits obtained through a qualified ACG**

ECSS-E-ST-40: 5.5.3.1 Coding		
Clause	**Description**	**Certification credit**
5.5.3.1	The supplier shall develop and document the following:	
a.1	the coding of each software unit	Automated: source code automatically generated
a.2	the build procedures to compile and link software units	User‘s responsibility. Can be automated by a custom extension to QGen
ECSS-Q-ST-80: 6.3.4 Coding		
Clause	**Description**	**Certification credit**
6.3.4.1	Coding standards […] shall be specified and observed.	Covered in the scope of ACG qualification
6.3.4.2	The standards shall be consistent with the product quality requirements	Redundant
6.3.4.3	The tools to be used in implementing and checking conformance with coding standards shall be identified […]	Redundant
6.3.4.4	Coding standards shall be reviewed with the customer to ensure that they reflect product quality requirements.	Redundant
6.3.4.5	Use of low level programming languages shall be justified.	Not used
6.3.4.6	a. The supplier shall define measurements, criteria and tools to ensure that the software code meets the quality requirements. b. The code evaluation shall be performed in parallel with the coding process […]	Redundant
6.3.4.7	Synthesis of the code analysis results and corrective actions implemented shall be described in the software product assurance reports.	Redundant
6.3.4.8	The code shall be put under configuration control immediately after successful unit testing.	User‘s responsibility

KRATES
inseneriburoo • engineering bureau

QGen and ECSS-E-ST-40C / ECSS-Q-ST-80C

- **Provided that the verification and validation is performed on the *input model* (LLR) several verification credits can be obtained through a qualified ACG**

ECSS-E-ST-40: 5.8.3.5 Verification of code		
Clause	**Description**	**Certification credit**
5.8.3.5	The supplier shall verify the software code ensuring that:	
a.1	the code is externally consistent with the requirements and design of the software item	Covered in the scope of ACG qualification
a.2	there is internal consistency between software units	Covered in the scope of ACG qualification
a.3	the code is traceable to design and requirements, testable, correct, and in conformity to software requirements and coding standards	Covered in the scope of ACG qualification
a.4	the code that is not traced to the units is justified	Covered in the scope of ACG qualification
a.5	the code implements proper events sequences, consistent interfaces, correct data and control flow, completeness, appropriate allocation of timing and sizing budgets, and error handling	Covered in the scope of ACG qualification
a.6	the code implements safety, security, and other critical requirements correctly as shown by appropriate methods	Covered in the scope of ACG qualification
a.7	the effects of run-time errors are controlled	Not covered. Static verification for run-time errors is performed by QGenv
a.8	there are no memory leaks	Covered in the scope of ACG qualification
a.9	numerical protection mechanisms are implemented	Responsibility of the model designer. Static verification of overflows etc. are performed by QGenv

QGen and ECSS-E-ST-40C / ECSS-Q-ST-80C

- **QGen ensures that the model coverage implies structural coverage of generated code**
- **If the compiler has been validated for the target platform, unit testing of the executable code becomes redundant**

ECSS-E-ST-40: 5.5.3.2 Software unit testing		
Clause	**Description**	**Certification credit**
5.5.3.2.a	The supplier shall develop and document the test procedures and data for testing each software unit.	Covered in the scope of ACG qualification
5.5.3.2.b	The supplier shall test each software unit ensuring that it satisfies its requirements and document the test results.	Covered in the scope of ACG qualification
5.5.3.2.c	The unit test shall exercise: 1. code using boundaries … 2. all the messages and error cases …	Covered in the scope of ACG qualification
ECSS-E-ST-40: 5.8.3.6 Verification of software unit testing (plan and results)		
Clause	**Description**	**Certification credit**
5.8.3.6	The supplier shall verify the unit tests results ensuring that:	
a.1	the unit tests are consistent with detailed design and requirements	Covered in the scope of ACG qualification
a.2	the unit tests are traceable to software requirements, design and code	Covered in the scope of ACG qualification
a.3	software integration and testing are feasible	Covered in the scope of ACG qualification
a.4	operation and maintenance are feasible	Not covered. Responsibility of the model designer
a.5	all activities defined in clause 5.5.3 are performed	Covered in the scope of ACG qualification
a.6	test results conform to expected results	Covered in the scope of ACG qualification
a.7	test results, test logs, test data, test cases and procedures, and test documentation are maintained under configuration management	Covered in the scope of ACG qualification

QGen and ECSS-E-ST-40C / ECSS-Q-ST-80C

- **QGen ensures that the model coverage implies structural coverage of generated code**
- **If the compiler has been validated for the target platform, unit testing of the executable code becomes redundant**

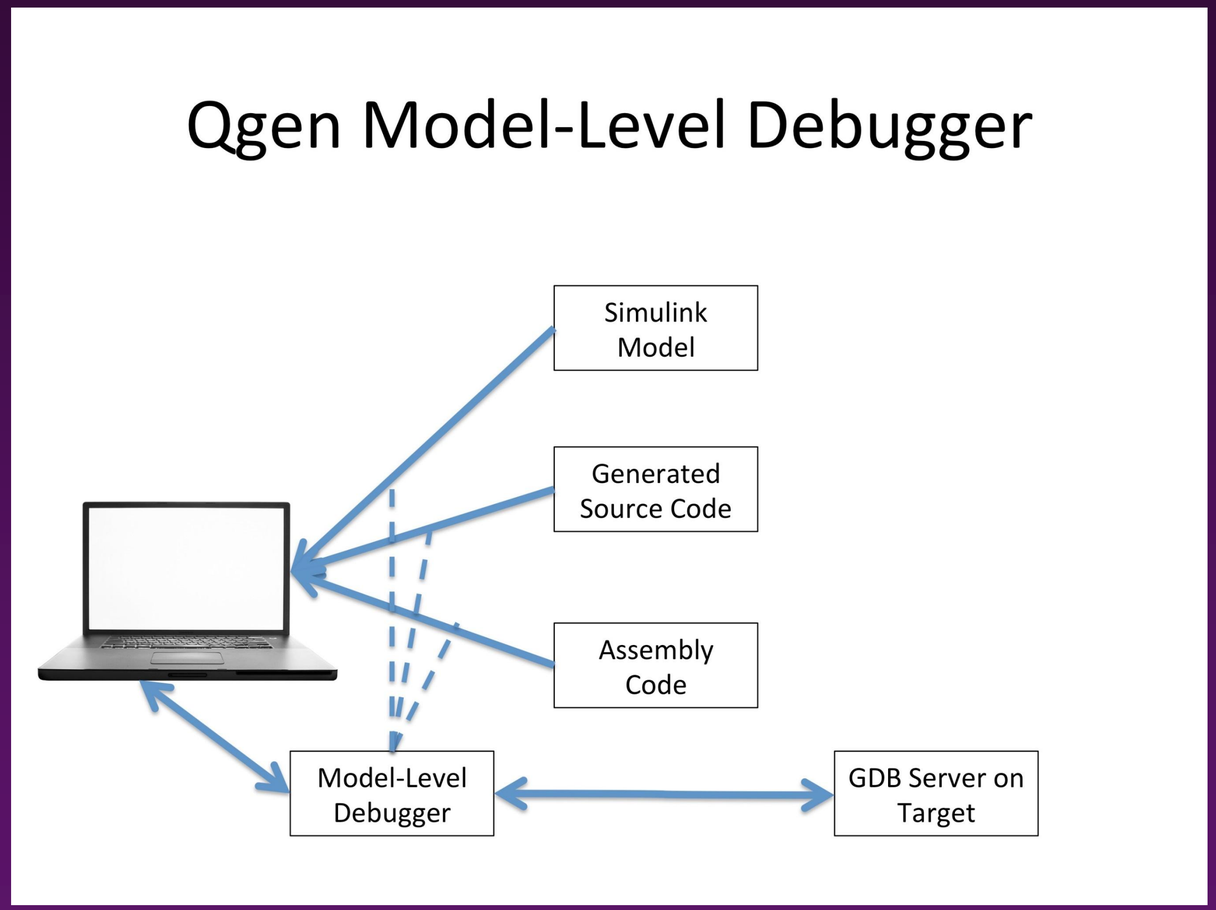
ECSS-E-ST-40: 5.8.3.6 Verification of software unit testing (plan and results)		
Clause	**Description**	**Certification credit**
a.8	normal termination (i.e the test end criteria defined in the unit test plan) is achieved	Covered in the scope of ACG qualification
a.9	abnormal termination of testing process (e.g incorrect major fault, out of time) is reported	Covered in the scope of ACG qualification
a.10	abnormal termination condition is documented in summary section of the unit test report, together with the unfinished testing and any uncorrected faults	Covered in the scope of ACG qualification
ECSS-Q-ST-80 6.3.5 Testing and validation		
Clause	**Description**	**Certification credit**
6.3.5.1	**Testing** shall be performed in accordance with a **strategy** for each testing level (i.e. unit, integration, ..., acceptance), ..	Covered in the scope of ACG qualification for the generated units
6.3.5.2	Based on the criticality of the software, **test coverage** goals for each testing level shall be agreed between the customer and the supplier and their achievement monitored by metrics: ...	Covered in the scope of ACG qualification for the generated units
...	...	...
6.3.5.32	Activities for the validation of the **quality requirements** shall be specified in the definition of the validation specification	Covered in the scope of ACG qualification for the generated units

QGen pipeline

KRATES
inseneribüroo • engineering bureau

Model-Level Debugging

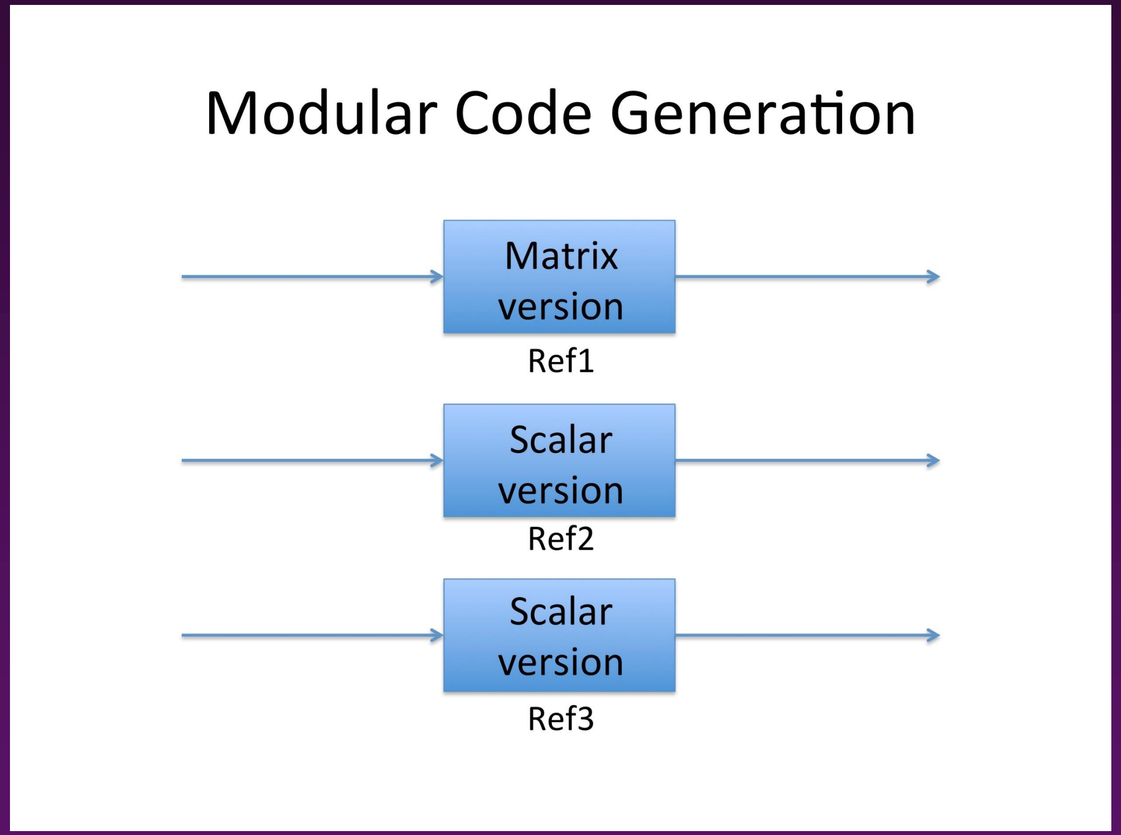
- *Model-Level Breakpoints*
  - Stop on event
  - Stop on signal transition
- *Model-Level Data Display*
  - Trace of Signal over Time
  - Sequence of Events
- *Model-Level Stepping*
  - Step one Time increment
  - Step one Block Computation/Update
- *Correlate Model Blocks with:*
  - Generated Code
  - Assembly Code



KRATES
inseneribüroo • engineering bureau

Modular Code Generation

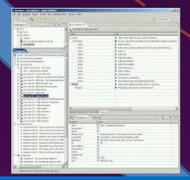
- *Reuse*
  - All Equivalent Uses of Library Block can Share Generated Code
  - e.g. The Two Scalar Version References can share code, even if they have state
- *Stability of Code and Naming*
  - Same Code and Names are Generated Each Time for given reference to Block, even if other parts of Model Change
  - e.g. Implementation of Matrix Version will always have same code and name even if code elsewhere changes



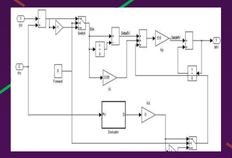
KRATES
inseneriburoo • engineering bureau

QGen + Spark

Requirements analysis



Requirements specification



Mode Specification

Final system conformity

Software verification
Unit conformity

INTRODUCTION OF FORMAL METHODS AT HIGH LEVEL

Write requirements and structure in SysML
Use SPARK to formalize the requirements
Use “satisfy” relationships to associate requir ements with design

High Readability
Model guide line

ENSURING A LEVEL OF EQUIVELANCE

Tier 1 can use high level model with SPARK to test their mode against it and ensure a certain level of equivalence

PRESERVING FORMALISM IN TO SIMULINK

Translate SysML into equivalent Simulink skeleton
Keep the SPARK requirements a s annotations.
Send this model with SPARK em bedded to Tier 1

USING FORMALISM TO ENSURE INTEGRITY AND AID IN TESTING

Translate Simulink into SPARK
Use QGen code generator
QGen will translate SPARK annotations into SPARK contract associated with the generated code.
Send this model with SPARK embedded to Tier 1

KRATES
inseneriburoo • engineering bureau

QGen-TASTE potential tasks

- Code generation for SDL models
  - Export SDL models and adapt QGen StateFlow generator to take into account SDL semantics
    OR
  - Export abstract source code from the current SDL generator and use QGen for converting to Ada/C
- Support for safety requirements in the interface model
  - Define requirements on the interface view (e.g. using VDM)
  - Convert to SPARK contracts using QGen
- Round-trip between top-level architecture in Simulink and TASTE
  - Depending on validation workflow it may make sense to duplicate high-level architecture in Simulink
  - QGen can help to import this high level architecture from Simulink to TASTE or vice versa

KRATES
inseneriburoo • engineering bureau

QGen availability and licensing

- All presented developments are licensed under the GPL license
- QGen is available from AdaCore as a commercial open-source product
  - Please visit http://www.adacore.com/qgen
- The TASTE modifications are integrated into the development branch
- Evaluation version of QGen will be integrated in the TASTE VM

KRATES
inseneribüroo • engineering bureau

KRATES
inseneribüroo • engineering bureau

Thank you!