ARGO

WCET-Aware Parallelization of Model-Based Applications
for Heterogeneous Parallel Systems

H2020-ICT-2015

Project Number: 688131

Deliverable D6.4

D6.4 Test Case Demonstration and Evaluation Report (First Increment)

Editors:	Dr. Umut Durak
Authors:	Dr. Umut Durak, David Mueller, Koray Kasnakli, Dr. Marcus Bednara
Version:	2
Status:	FINAL
Dissemination level:	Public (PU)
Filename:	D6.4_test_case_demo_and_eval_I1_1.02.docx

ARGO Consortium

Karlsruhe Institute of Technology	DE
Scilab Enterprises	FR
Recore Systems B.V.	NL
Université de Rennes I	FR
Technological Educational Institute of Western Greece	GR
AbsInt Angewandte Informatik GmbH	DE
Deutsches Zentrum für Luft- und Raumfahrt	DE
Fraunhofer IIS	DE
emmtrix Technologies GmbH	DE

Document revision history

Version	Based on	Date	Author	Comments / Changes
1v0		04.08.2017	U. Durak, D. Mueller, K. Kasnakli	Final Version
1v01	1v0	19.10.2017	U. Durak, D. Mueller	Final version, proofread for consistency
1v02	1v01	26.10.2017	M. Bednara, K. Kasnakli	Final version, proofread for consistency

Table of Contents

List of Figures

List of Tables

Glossary of Terms

API	Application Programming Interface
CG	Code Generation
CI	Continuous Integration
EGPWS	Enhanced Ground Proximity Warning System
HIL	Hardware-in-the-Loop
IR	Intermediate Representation
HTG	Hierarchical Task Graph
SSA	Static Single Assignment
SIL	Software-in-the-Loop
WCET	Worst Case Execution Time

1. Introduction

1.1 Purpose

This document reports the test and evaluation of the ARGO Toolchain using both of our test cases (Enhanced Ground Proximity Warning System and Polarization Image Processing System) as specified in Deliverable D6.2 “Test Cases and Requirements Specification”. We will specify the demonstration and evaluation phases in detail and report the results of demonstration and evaluation efforts.

This documentation of the demonstration and evaluation is based on the *IEEE Standard for Software and Systems Test Documentation* (IEEE Std 829-2008) [1] which provides clear guidelines for documenting test, demonstration and evaluation efforts. Since this IEEE standard was developed for a wide range of software systems, not all parts of the standard are convenient for our specific case. Therefore, we used the standard to tailor and design the outline of this report.

1.2 Overview

This document is organized as follows: in Section 2 we give the specification of demonstrations and evaluations based on the “Test Design and Phases” that were previously described in ARGO Deliverable D6.3 *Test cases and Design and Implementation* [2]. In Section 3 we report the results of each demonstration and evaluation specified. Section 4 provides an outlook over the next steps planned in both test cases and Section 5 provides a list of references.

The overview of the evaluation could be reported as positive. As of the first increment, where we are standing at the middle of the project schedule, there exists an integrated toolchain that is provided in a Linux container (as a Docker image). While there are various remarks resulting from the evaluation that will be introduced in the body of the report, as of the date the report is written, it is possible to go from a Scilab/Xcos diagram (or Scilab scripts) to parallel code. Along with that, an extensive evaluation infrastructure has been developed with a large number of real life industry scale test cases and demonstrations. This infrastructure is expected to guide the project until the end with end user requirements.

2. Demonstration and Evaluation Specification

2.1 Enhanced Ground Proximity Warning System (DLR)

2.1.1 Overview

The requirements of ARGO EGPWS are documented in D6.2 Test Cases and Requirements Specification [3]. The test and evaluation workflow with ARGO EGPWS is depicted in Figure 1.

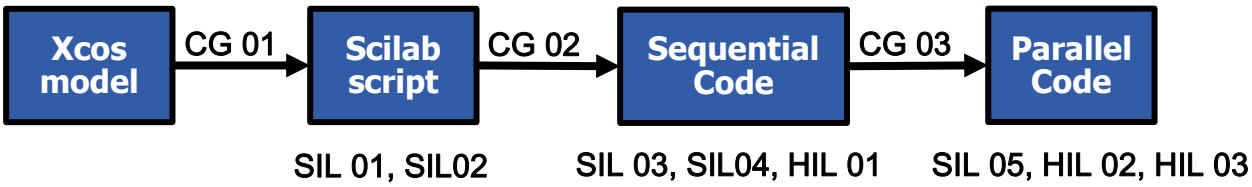


Figure 1: The test and evaluation workflow for the ARGO EGPWS

The description of the items in Figure 1 can be given as follows:

Code Generation (CG) is the generation of target deployable code using the ARGO Toolchain. The code generation is tested and evaluated in three steps using the ARGO Toolchain as follows:

CG 01: Scilab script generation from Xcos models.
CG 02: Sequential C code generation from Scilab script.
CG 03: Parallel C code generation from sequential C code.

Software-in-the-Loop Testing (SIL) is the testing of generated code. It is essentially non real-time and targets at functional verification. SIL testing is being done repetitively, corresponding to the steps of code development as listed below:

SIL 01: Open-loop tests are executed for Scilab scripts generated from Xcos model elements in order to verify that the outcomes of script execution are complying with those of the Xcos model elements.
SIL 02: Closed-loop tests are executed for Scilab scripts generated from the overall Xcos model in order to verify that the outcomes of script execution are complying with those of the overall Xcos model.
SIL 03: Open-loop tests are executed for sequential C code generated from the Scilab scripts that correspond to Xcos model elements in order to verify that the outcomes of sequential C code execution are complying with the Scilab scripts for Xcos model elements.
SIL 04: Closed-loop tests are executed for sequential C code generated from the Scilab scripts of the overall Xcos model elements in order to verify that the outcomes of sequential C code execution are complying with the Scilab scripts for the overall Xcos model.
SIL 05: Closed-loop tests are executed for generated parallel C code of the overall Xcos model in order to verify that the outcomes of parallel C code execution are complying with the sequential C code for the overall Xcos model. This task is planned for increment 2 using both the RECORE FlexaWare SDE and InvasIC API.

Hardware-in-the-Loop Testing (HIL) is the testing of generated C code on a target platform essentially in a real-time setting. It enables the verification of non-functional requirements. This task is planned for increment 2 and will be done in the following three steps:

HIL 01: closed loop tests will be executed for generated sequential C code on a single core target. The plant model is executed on an x86 PC running on a real-time operating system.

HIL 02: closed loop tests will be executed for generated parallel C code on the ARGO RECORE platform. The plant model is executed on an x86 PC running on a real-time operating system.

HIL 03: Piloted test runs will be executed with the generated parallel C code on the ARGO RECORE platform integrated into DLR's AVES.

2.1.2 Code Generation

Code Generation (CG) is the generation of target deployable code using the ARGO Toolchain as described in the following subsections.

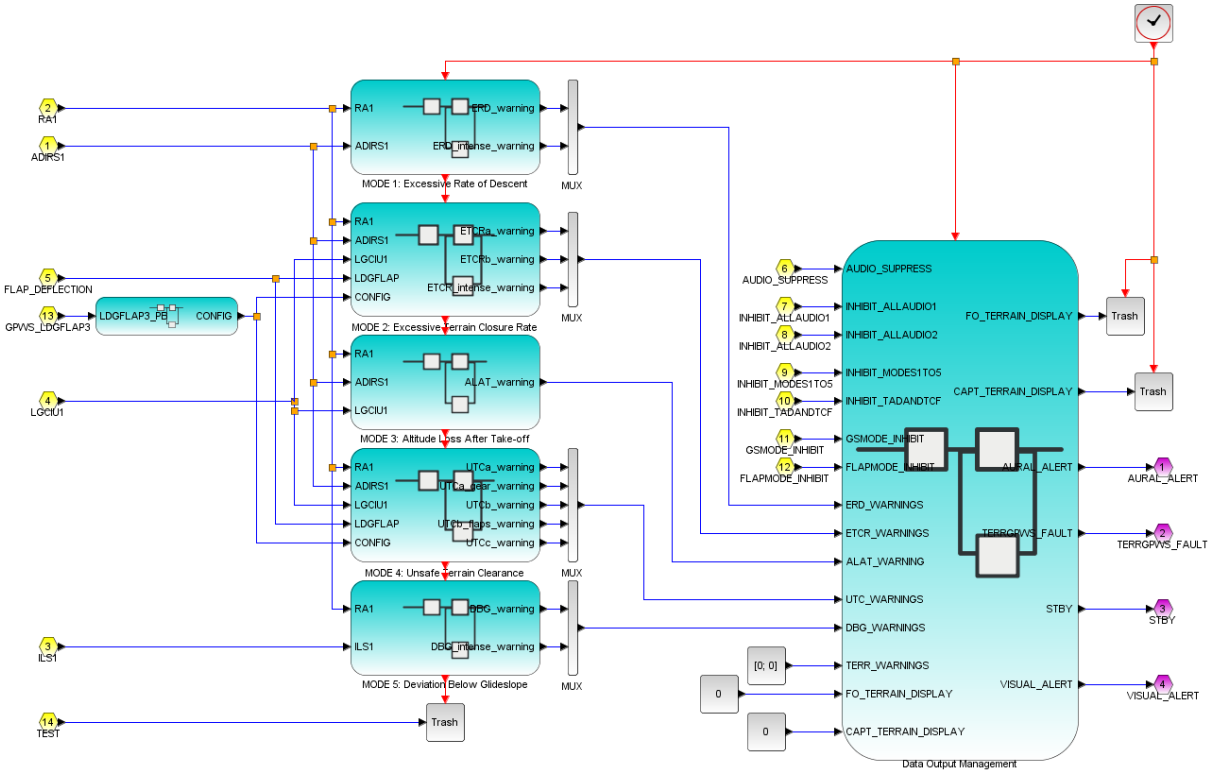


Figure 2: ARGO EGPWS Design Composition

2.1.2.1 Code Generation 01 (CG 01)

Identification: CG 01 Scilab script generation from Xcos models.

Purpose: The purpose of GC 01 is to test the Scilab script generation feature of the ARGO Toolchain. The functionality assigned to the Scilab/Xcos Front-End is the translation of input Xcos models to Scilab scripts. GC 01 addresses the objectives OBJ1, OBJ2 and OBJ5 as follows:

OBJ1: The availability of the ARGO Toolchain will be demonstrated by generating Scilab scripts for supported Xcos block set as described in document D 3.1 Intermediate Representation and Sequential Code Generation.

OBJ2: Subjective assessment will be provided about the comparison of Scilab/Xcos Front-End and MATLAB/Simulink.

OBJ5: The code generation will be executed for the comprehensive time-critical use case ARGO EGPWS

Approach: The overall ARGO EGPWS model and individual model elements that are subject to code generation are depicted in Figure 2. In this step Scilab scripts will be generated for single Xcos model elements. This is currently carried out by the Scilab/Xcos Front-End (Figure 3).

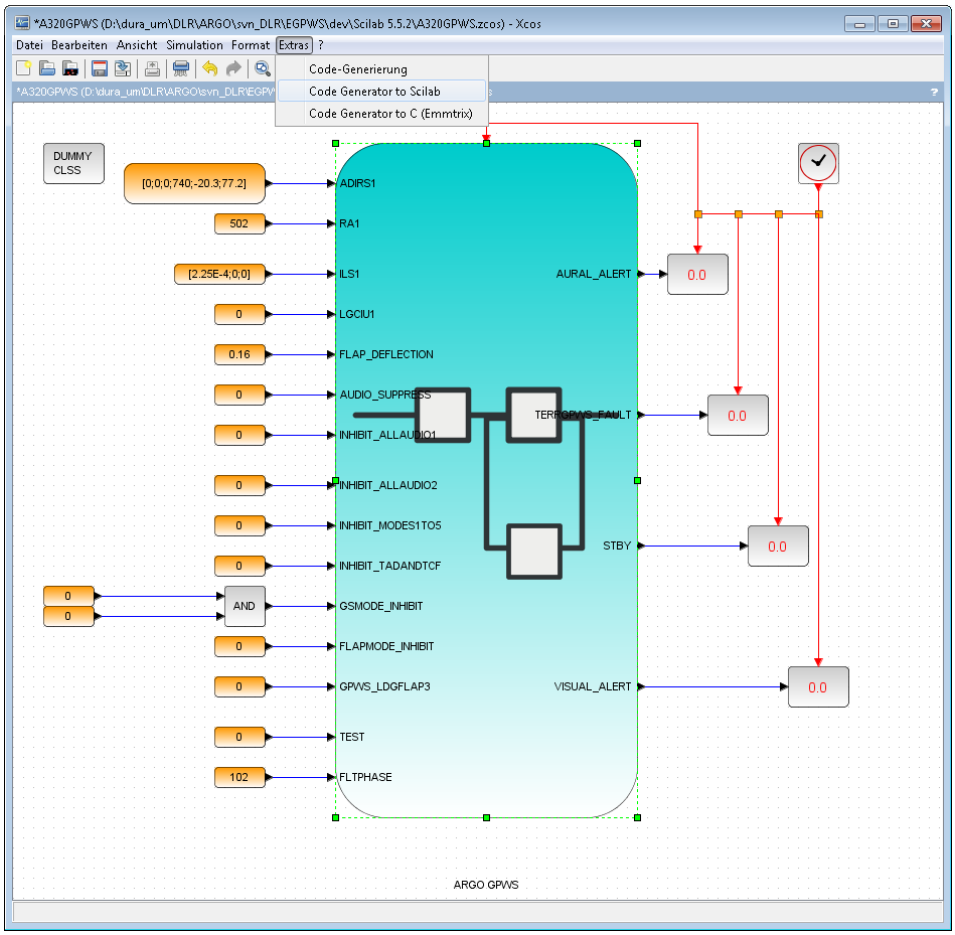


Figure 3: Scilab Script Generation from Xcos models

The Pass/Fail Criteria of this step is the generation of Scilab scripts without any error. We expect to a folder with the name *generated*, containing two Scilab files, namely *ARGO_GPWS.sci* and *ARGO_GPWS_scenario.sce*. The plausibility test is conducted by running the *ARGO_GPWS_scenario.sce* which executes the *ARGO_GPWS.sci* with random inputs. One successful execution is enough to claim the success of this step.

The following steps were followed to prepare the setup for the test case:

- Scilab 5.5.2 was installed.
- Xcos Code Generator was installed in Scilab using *atomsInstall*.

2.1.2.2 Code Generation 02 (CG 02)

Identification: CG 02 Sequential C-code generation from Scilab script.

Purpose: The purpose of GC 02 is to test the sequential C-code generation feature of the ARGO Toolchain. The functionality assigned to the ARGO Toolchain is transforming input Scilab script to sequential C-code. GC 02 addresses the objectives OBJ1, OBJ2 and OBJ5 as follows:

OBJ1: The availability of the ARGO Toolchain will be demonstrated by conducting sequential C-code generation for the supported Xcos block set described in document D3.1 *Intermediate Representation and Sequential Code Generation*.

OBJ2: Subjective assessment will be provided about the comparison of the ARGO Toolchain and MATLAB/Simulink.

OBJ5: The code generation will be executed for the comprehensive time-critical use case ARGO EGPWS.

Approach: The below Scilab scripts are subject to code generation:

Overall ARGO EGPWS:

ARGO_GPWS_scenario.sce

ARGO_GPWS.sci

Mode 1: Excessive Rate of Descent:

MODE_1_Excessive_Rate_of_Descent_scenario.sce
MODE_1_Excessive_Rate_of_Descent.sci

Mode 2: Excessive Terrain Closure Rate:

MODE_2_Excessive_Terrain_Closure_Rate_scenario.sce

MODE_2_Excessive_Terrain_Closure_Rate.sci

Mode 3: Altitude Loss After Take-off:

MODE_3_Altitude_Loss_After_Takeoff_scenario.sce

MODE_3_Altitude_Loss_After_Takeoff.sci

Mode 4: Unsafe Terrain Clearance:

MODE_4_Unsafe_Terrain_Clearance_scenario.sce

MODE_4_Unsafe_Terrain_Clearance.sci

Mode 5: Deviation Below Glideslope:

MODE_5_Deviation_Below_Glideslope_scenario.sce

MODE_5_Deviation_Below_Glideslope.sci

Data Output Management:

Data_Output_Management_scenario.sce

Data_Output_Management.sci

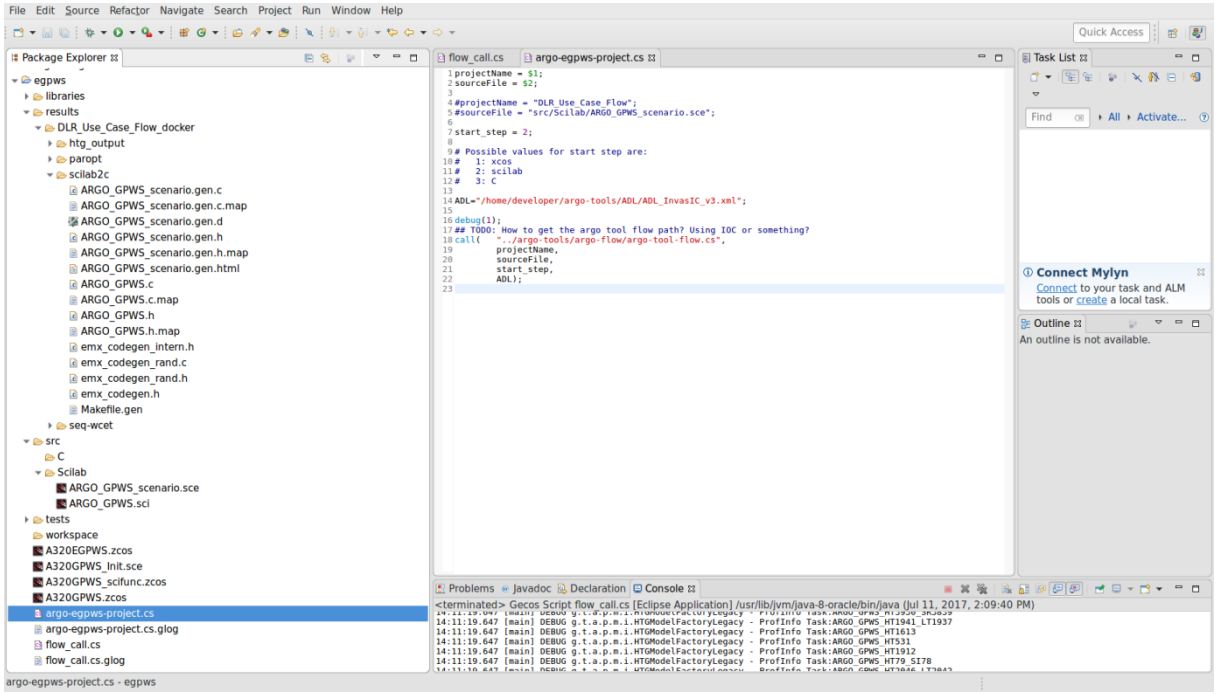


Figure 4: ARGO Toolchain

Sequential C code generation is carried out using the ARGO Toolchain (Figure 4). Compiler scripts are utilized for the automation of the process. Figure 5 presents two compiler scripts, *flow_call.cs* and *argo-egpws-project.cs* that set the project parameters and call the *argo-tool-flow.cs for* the code generation. *argo-tool-flow.cs* can execute the whole process from Xcos model to parallel code. Here within this test we start with step 2 (Scilab script) and only consider the outputs that are created for sequential code generation.

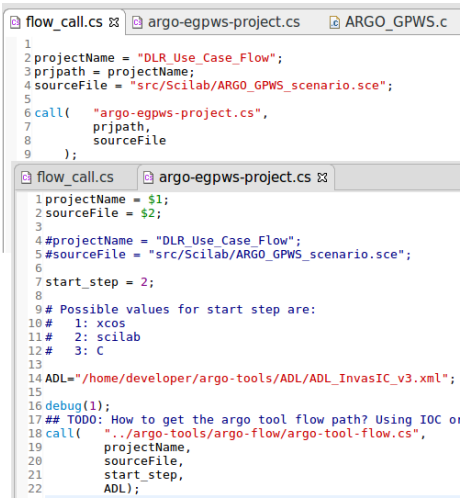


Figure 5: Compiler Scripts

The Pass/Fail Criteria of this step is the generation of sequential C code that compiles without any error.

2.1.2.3 Code Generation 03 (CG 03)

Identification: CG 03 Parallel C-code generation from sequential C-code.

Purpose: The purpose of GC 03 is to test the parallel C-code generation feature of ARGO Toolchain. The functionality assigned to the ARGO Toolchain is transforming input sequential C-code to parallel C-code. GC 03 addresses the objectives OBJ1, OBJ2 and OBJ5 as follows:

OBJ1: The availability of the ARGO toolchain will be demonstrated by conducting parallel C code generation for the supported Xcos block set described in document D 3.1 *Intermediate Representation and Sequential Code Generation*.

OBJ2: Subjective assessment will be provided about the comparison of the ARGO Toolchain and MATLAB/Simulink.

OBJ5: The code generation will be executed for the comprehensive time-critical use case ARGO EGPWS.

Approach: The below Scilab scripts are subject to code generation:

Overall ARGO EGPWS:

ARGO_GPWS_scenario.h

ARGO_GPWS_scenario.c

ARGO_GPWS.h

ARGO_GPWS.c

The sequential C-code generation is carried out using the ARGO Toolchain (Figure 3). Compiler scripts are utilized for the automation of the process. We used the compiler scripts from CG 02 that are presented in Figure 5, namely *flow_call.cs* and *argo-egpws-project.cs*. They set the project parameters and call the *argo-tool-flow.cs* for the code generation. *argo-tool-flow.cs* is meant to execute the whole process from Xcos model to parallel code. Here within this test we start with step 3 (Sequential C-Code) and check the outputs that will be created for the parallel C-code.

The Pass/Fail Criteria of this step is the generation of parallel C code that compiles without any error.

2.1.3 Software-in-the-Loop Testing

Software-in-the-Loop Testing (SIL) is executing the generated code by putting it in a test loop. It is essentially non real-time and targets functional verification. SIL testing is being done repetitively, corresponding to the steps of code development as listed below:

2.1.3.1 Software-in-the-Loop Testing 01 (SIL 01)

Identification: SIL 01 Open-loop testing for Scilab scripts generated from Xcos model elements

Purpose:

Unit tests are executed for Scilab scripts generated from Xcos model elements in order to verify that the outputs of the scripts are complying with those of the Xcos model elements. SIL 01 addresses the objectives OBJ1 and OBJ5 as follows:

OBJ1: The functionality of the ARGO Toolchain will be demonstrated by conducting Software-in-the-loop tests on the Scilab scripts that it generates. SIL 01 will verify the correct code generation for Scilab Scripts within the ARGO Toolchain.

OBJ5: This test case demonstrates the industry standard testing approaches with the ARGO Toolchain for the comprehensive time-critical use case ARGO EGPWS.

Approach: The below Scilab scripts are subject to testing:

Mode 1: Excessive Rate of Descent:

MODE_1_Excessive_Rate_of_Descent.sci

Mode 2: Excessive Terrain Closure Rate:

MODE_2_Excessive_Terrain_Closure_Rate.sci

Mode 3: Altitude Loss After Take-off:

MODE_3_Altitude_Loss_After_Takeoff.sci

Mode 4: Unsafe Terrain Clearance:

MODE_4_Unsafe_Terrain_Clearance.sci

Mode 5: Deviation Below Glideslope:

MODE_5_Deviation_Below_Glideslope.sci

Data Output Management:

Data_Output_Management.sci

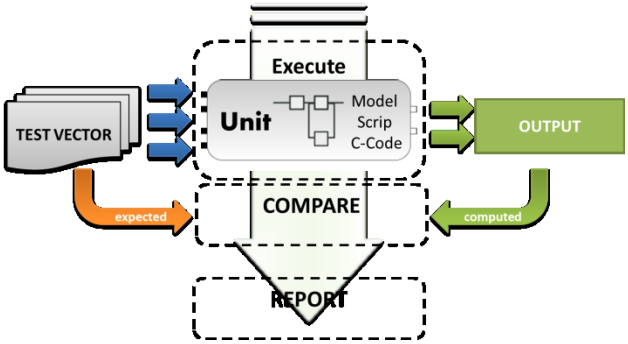


Figure 6: Open-loop Unit Testing

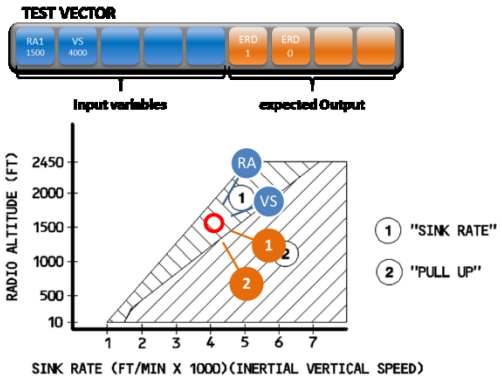


Figure 7: Test Vectors Construction

The approach used in SIL 01 can be presented as open-loop unit testing in a black-box fashion (Figure 6). The test vectors and expected outputs are constructed using the requirements specifications (Figure 7). The interfaces of the components of the ARGO EGPWS model are executed extensively regarding the range coverage (Figure 8).

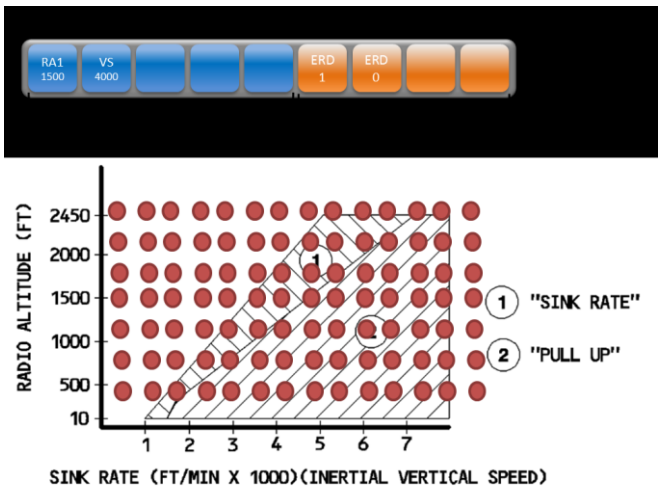


Figure 8: Test Input Selection

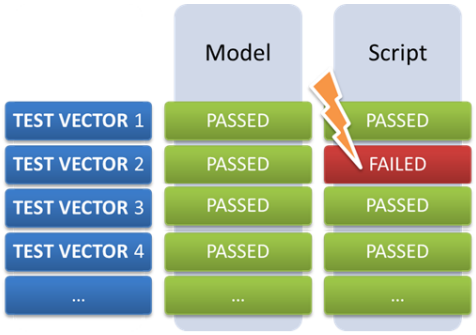


Figure 9: Pass/Fail Criteria

The Pass/Fail Criteria of this step is the conformance of the script outputs with the expected outputs which are aligned with the source Scilab/Xcos model output (Figure 9).

2.1.3.2 Software-in-the-Loop Testing 02 (SIL 02)

Identification: SIL 02 Closed-loop testing for Scilab scripts generated from the overall ARGO EGPWS Scilab/Xcos model.

Purpose:

Scenario tests are executed for Scilab scripts generated from the overall ARGO EGPWS Scilab/Xcos model in order to verify that the outputs of the scripts are complying with those of the Xcos model. SIL 02 addresses the objectives OBJ1 and OBJ5 as follows:

OBJ1: The functionality of the ARGO Toolchain will be demonstrated by conducting Software-in-the-loop tests on the Scilab scripts that it generates. SIL 02 will verify the correct code generation for Scilab scripts within the ARGO Toolchain.

OBJ5: This test case demonstrates the industry standard testing approaches with ARGO Toolchain for the comprehensive time-critical use case ARGO EGPWS.

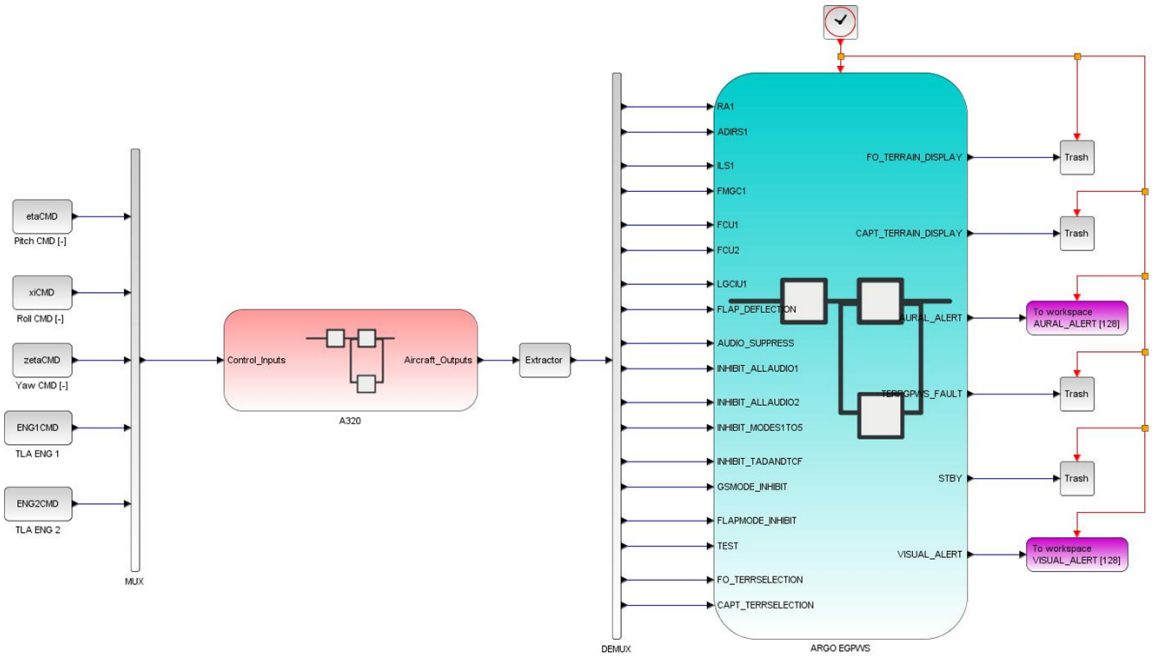


Figure 10: ARGO EGPWS Close-loop Test Model

Approach: This is the Scilab script that is subject to testing:

ARGO_GPWS.sci

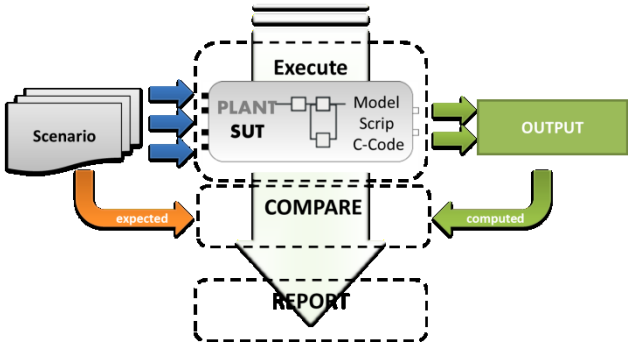


Figure 11: Close-loop Scenario Testing

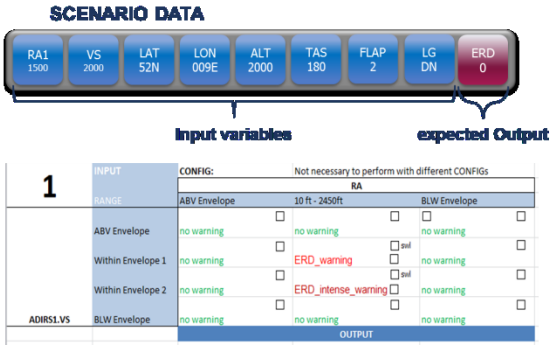


Figure 12: Test Vectors Construction

The approach used in SIL 02 can be presented as closed-loop scenario testing in a black-box fashion (Figure 11). Closed-loop testing reintegrates the generated script/code back into the Xcos schema and executes the test scenario with a plant and the system under test (SUT) together (Figure 10). The plant in the SIL 02 case is the aircraft (Airbus A320) and the SUT is the ARGO EGPWS. A Scilab Function Block (scifunc_block_m) of Xcos is utilized to integrate the auto generated Scilab script back into the model schema (Figure 13).

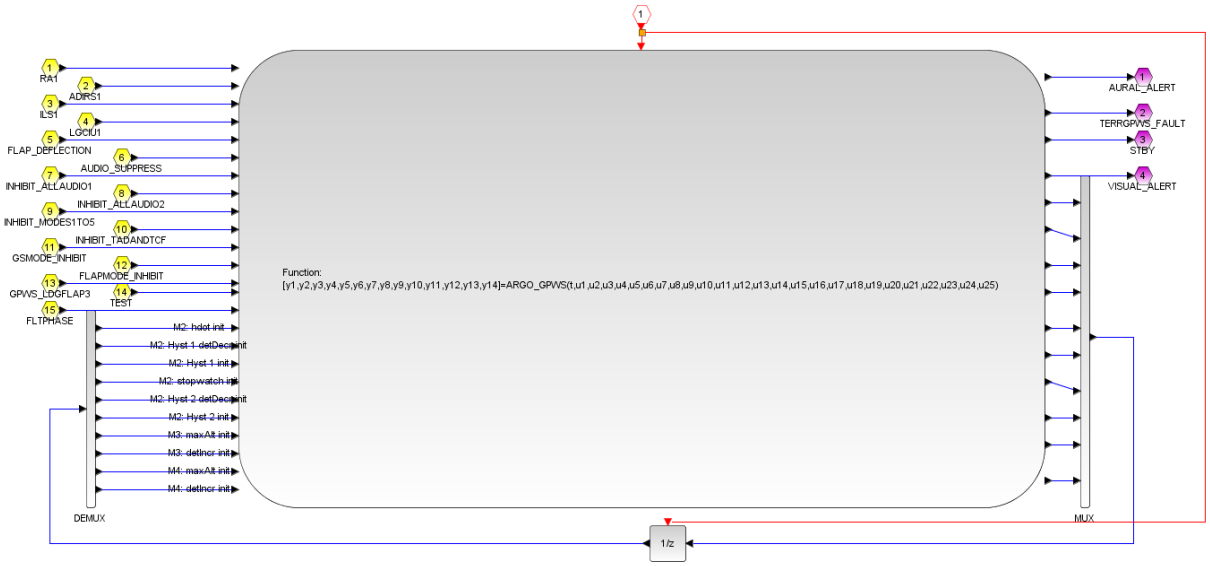


Figure 13: SIL 02 ARGO EGPWS Scilab Function Block

The scenarios that contain the initial state and the course of events during the test execution are applied to the test model; the model outputs are compared to the expected outputs and the results are reported. An excerpt from a sample scenario is given in Figure 14. Scenarios are constructed using the requirement specifications in order to cover the 5 modes of the ARGO EGPWS (Figure 15). The inputs are classified regarding the decision trees of each mode. All possible output values of modes are addressed by the designed scenarios.

```
//Flight in 3000ft at V_TAS = 200kn and VS +2000 ft/min
scenario.aircraft.flightstate.position.latitude = 10*deg2rad; //in °, East is positive
scenario.aircraft.flightstate.position.longitude = 52*deg2rad; //in °, North is positive
scenario.aircraft.flightstate.position.altitude = 3000*ft2m; //in ft

scenario.aircraft.flightstate.attitude.phi = 0; //in rad
scenario.aircraft.flightstate.attitude.theta = 0.204407070913782; //in rad
scenario.aircraft.flightstate.attitude.psi = 0; //in rad

scenario.aircraft.flightstate.angularvelocity.p_K_f = 0*deg2rad; //in °/s
scenario.aircraft.flightstate.angularvelocity.q_K_f = 0*deg2rad; //in °/s
scenario.aircraft.flightstate.angularvelocity.r_K_f = 0*deg2rad; //in °/s

//scenario.aircraft.flightstate.translationalvelocity.V_TAS = u_K_f * cos theta]
scenario.aircraft.flightstate.translationalvelocity.u_K_f = 102.3108373093248; //in m/s
scenario.aircraft.flightstate.translationalvelocity.v_K_f = 0; //in m/s
scenario.aircraft.flightstate.translationalvelocity.w_K_f = 10.882697876220250; //in m/s
```

Figure 14: Sample Excerpt from Scenario

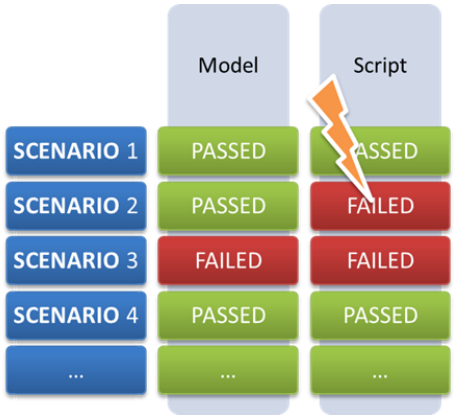


Figure 15: Pass/Fail Criteria

The Pass/Fail Criteria of this step is the conformance of the script outputs with outputs of the source Scilab/Xcos model. As presented in Figure 15, if the source Scilab/Xcos model is passing the test in a particular scenario, we are expecting the script to also pass.

2.1.3.3 Software-in-the-Loop Testing 03 (SIL 03)

Identification: SIL 03 Open-loop testing for sequential C-code generated from Scilab scripts for Xcos model elements

Purpose:

Unit tests are executed for sequential C-code generated from Scilab scripts for Xcos model elements in order to verify that the outcomes of sequential C-code are complying with the Xcos model elements. SIL 03 addresses objectives OBJ1 and OBJ5 as follows:

OBJ1: The functionality of the ARGO Toolchain will be demonstrated with conducting Software-in-the-loop tests on the generated sequential C-code. SIL 03 will verify the correct code generation of ARGO Toolchain for sequential C-code.

OBJ5: This test case demonstrates the industry standard testing approaches with ARGO Toolchain for the comprehensive time-critical use case ARGO EGPWS

Approach: The below C-files are subject to testing:

Mode 1: Excessive Rate of Descent:

MODE_1_Excessive_Rate_of_Descent.c

Mode 2: Excessive Terrain Closure Rate:

MODE_2_Excessive_Terrain_Closure_Rate.c

Mode 3: Altitude Loss After Take-off:

MODE_3_Altitude_Loss_After_Takeoff.c

Mode 4: Unsafe Terrain Clearance:

MODE_4_Unsafe_Terrain_Clearance.c

Mode 5: Deviation Below Glideslope:

MODE_5_Deviation_Below_Glideslope.c

Data Output Management:

Data_Output_Management.c

The approach used in SIL 03 can be presented as open-loop unit testing in a black-box fashion (Figure 6). The test vectors and expected results are constructed using the requirements specifications (Figure 7). The interfaces of the components of the EGPWS model are exercised extensively regarding the range coverage (Figure 8).

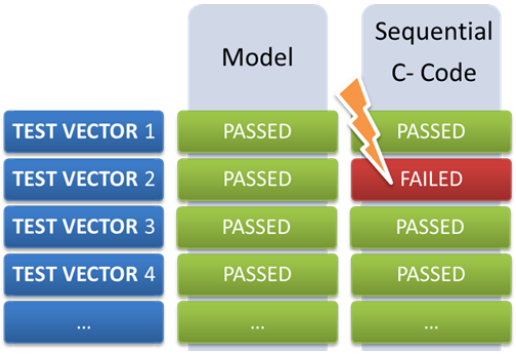


Figure 16: Pass/Fail Criteria

The Pass/Fail Criteria of this step is the conformance of the sequential C-code outputs with the expected outputs which are aligned with the source Scilab/Xcos model output (Figure 9).

2.1.3.4 Software-in-the-Loop Testing 04 (SIL 04)

Identification: SIL 04 Closed-loop testing for sequential C-code generated from Scilab scripts for the overall EGPWS Scilab/Xcos model

Purpose:

Scenario tests are executed for sequential C-code generated from Scilab scripts for the overall EGPWS Scilab/Xcos model in order to verify that the outputs of the sequential C-code are complying with those of the Xcos model. SIL 04 addresses the objectives OBJ1 and OBJ5 as follows:

OBJ1: The functionality of the ARGO Toolchain will be demonstrated by conducting Software-in-the-loop tests on the generated sequential C-code. SIL 04 will verify the correct code generation of sequential C-code within the ARGO Toolchain.

OBJ5: This test case demonstrates the industry standard testing approaches with ARGO Toolchain for the comprehensive time-critical use case ARGO EGPWS.

Approach: This is the C-file that is subject to testing:

ARGO_GPWS.c

The approach used in SIL 04 can be presented as closed-loop scenario testing in a black-box fashion (Figure 11). Closed-loop testing reintegrates the generated script/code back into the Xcos schema and executes the test scenario with a plant and the SUT (Figure 10). The plant in the SIL 04 case is the aircraft (Airbus A320) and the system under test is the ARGO EGPWS.

There is a number of ways for reintegrating the generated script/code back into the Xcos schema. We have chosen to compile the sequential C-code as a separate process using the real-time simulation architecture 2Simulate [4] of DLR. Thereafter, the executable that is used in SIL 04 will later be used in flight simulator integration studies. Scilab/Xcos UDP Blocks are utilized to integrate the separate ARGO GPWS process back into the model schema (Figure 17).

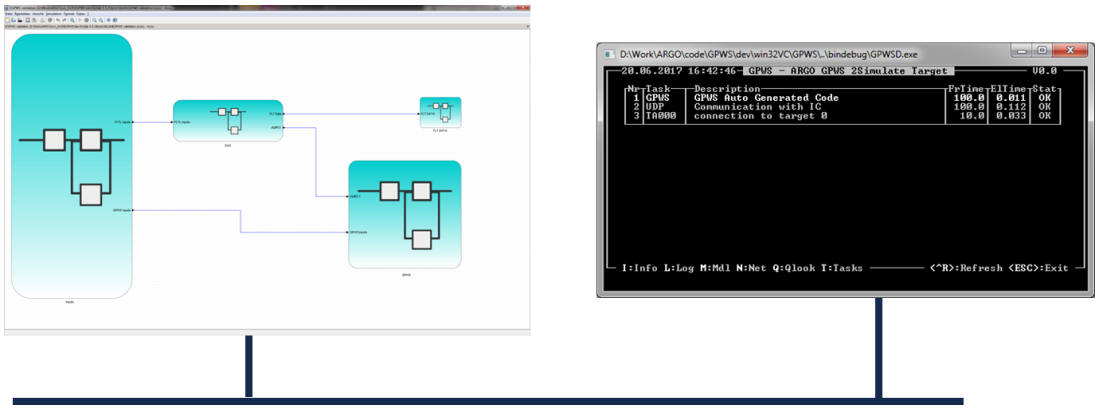


Figure 17: SIL 04 ARGO EQPWS and Scilab/Xcos Integration

The scenarios that contain the initial state and the course of events during the test execution are applied to the test model; the model outputs are compared to expected outputs and the results are reported. An excerpt from a sample scenario is given in Figure 14. Scenarios are constructed using the requirement specifications in order to cover the 5 modes of ARGO EGPWS (Figure 15). The inputs are classified regarding the decision trees of each mode. All possible output values of modes are addressed by the designed scenarios.

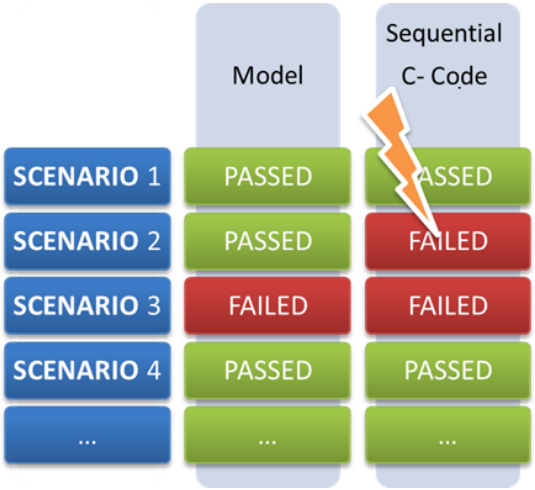


Figure 18: SIL 04 Pass/Sail Criteria

The Pass/Fail Criteria of this step is the conformance of the script outputs with the outputs of the source Scilab/Xcos model. As presented in Figure 18, if the source Scilab/Xcos model is passing the test in a particular scenario, we are expecting the C-code to pass as well.

2.2 Polarization Image Processing System (Fraunhofer IIS)

Referring to D6.3 *Test cases and Design and Implementation* [2], phase 2b is the concern of this documentation, which corresponds to Software-in-the-loop testing. Model-in-the-loop and Hardware-in-the-loop tests are planned for the second increment of the corresponding deliverable. Please note that the code generation is considered as a submodule of Software-in-the-loop testing in the IIS use-case instead of separate modules as in the DLR use-case.

Table 2 in D6.3 shows Test Case Phases vs. Measurements [2]. The objectives that have to be achieved in this increment are given as OBJ1 and OBJ5 and the first part of OBJ2.

2.2.1 Testing Specifications

The objects to be tested are divided into the following two categories.

Unit Tests: The macros **paROI()**, **paGOCorrection()**, **paDenoise()**, **paInterpolation()**, **paStokes()**, **paAomp()**, **paDolp()** based on D6.3 *Test cases and Design and Implementation* [2] with minor changes (refer to section 2.2.2.2) are tested considering the principle of least effort. For that purpose, the main Scilab script **argo.sci** and function **testCaseDemoPolkaFlow()** are modified into **argo_*macroname*.sci** and **tCDPF_*macroname*()** respectively. The modifications consist of replacing the real data with some predefined matrix and commenting out all macros other than the one under test. An exception to this rule is the macro **paROI()**, which crops the input matrix into the desired size. For this increment, the TCP/IP interface is left out. In order to execute the unit tests, a small compiler script **unit_test_call.cs** is written.

Flow Test: The processing pipeline described in D6.3 *Test cases and Design and Implementation* [2] is the concatenation of the macro blocks mentioned in Unit tests. The main Scilab script **argo_predef.sci** and function **tCDPF_predef()** are again the modified

versions, which feed a predefined matrix into the pipeline and leave out the TCP/IP interface. In order to execute the flow test, a small compiler script **flow_call.cs** is written.

The main purpose of this increment will be the correct code generation - parallel as well as sequential. The resulting matrices from unit and flow tests computed by Scilab, generated sequential code and parallel code respectively are compared against each other visually for Scilab output and via the built-in function **CompareOutputs()** inside the GeCoS framework for sequential and parallel outputs.

Testing for WCET-awareness and for TCP/IP interface is planned to be tested in the second increment of this deliverable.

Expected outputs are binaries of generated sequential and parallel C-codes and the intermediate files to generate those binaries, which are going to be described in detail in the next section.

2.2.1.1 Approach

All tests are done in the Eclipse environment of the ARGO Toolchain which is integrated into a docker image. To be able to test the components, we have to call the corresponding units from within the GeCoS framework based on the Eclipse environment inside this image.

Following are the necessary steps for an end-user to generate the resulting sequential and parallel codes.

For Unit Tests, **unit_test_call.cs** should be executed, which defines the project name and the relevant output folder. The same should be done with **flow_call.cs** script for Flow Test.

2.2.1.2 Item pass/fail criteria

An object passes the test if the execution of the sequential or parallel code does not trigger an error and if the sequential and parallel code provide identical results.

2.2.2 Overview of the Intermediate Steps

Here is a general overview of the intermediate steps for the ARGO Toolchain to generate sequential and parallel code and its executables. For detailed information please refer to documents D5.1 *Interface Specification* [5] and D3.2 Algorithms for cross layer programming [6]. Greyed-out titles are not subject of this increment.

Scilab to C via Emmtrix Code Generator

Code generation with WCET pragmas and end-user constraints

Intermediate Representation (IR) and Hierarchical Task Graph (HTG) Generation

Generation of GeCoS IR, Static Single Assignment (SSA) and HTG

Code Transformations

Predictability enhancement optimization to expose more parallelism

Core-level code-snippet WCET Estimation

WCET calculations at different granularities using aiT WCET analyzer

Parallelization / Optimize Program Schedule

Optimal task and data mapping

Parallel Program IR

Extended GeCoS IR with channel-based communication and synchronization

System-Level WCET

Final Core-level WCET Estimation

Architecture Specific Post-Optimizer

Parallel Code Generation

Parallel C code based on IR of parallel program

Sequential / Parallel Code Comparison

2.2.2.1 Inputs, outputs, and special requirements

Please refer to Section 3.2.2. part h) for the resulting output folder from the toolchain and its explanations and correspondences to the following steps of the ARGO-Flow.

- **Scilab to C via emmtrix Code Generator**

  Inputs: Scilab source code of unit tests and flow test with end-user constraints

  Outputs: Sequential platform independent C code based on C99 with code annotations as comments or pragmas

  Special Requirements: -

- **Intermediate Representation (IR) and Hierarchical Task Graph (HTG) Generation**

  Inputs: Annotated C code

  Outputs: IR, SSA and HTG with annotations

  Special Requirements: -

- **Code Transformations**

  Inputs: HTG with annotations

  Outputs: optimized HTG with loop bounds and data array size tags for validity for scratchpad memory mapping

  Special Requirements: -

- **Core-level code-snippet WCET Estimation**

  Inputs: HTG with loop bounds and data array size tags for validity for scratchpad memory mapping

  Architecture ADL, to identify target architecture for aiT config file generation

  Outputs: For all tasks (hierarchical and leaf) 2 outputs corresponding to 2 scenarios, assuming code and data in local scratchpad memory or in external DRAM

  Special Requirements: aiT license file

- **Parallelization / Optimize Program Schedule**

  Inputs: HTG with WCET times for each leaf tasks, End-user constraints, ADL description with communication timing information

  Outputs: Complete map of each leaf task nodes in HTG on available cores, complete scheduling info of task nodes in HTG, Preliminary decision of data allocation for each variable to a specific memory hierarch

  Special Requirements: -

- **Parallel Program IR**

  Inputs: Annotated HTG from mapping and scheduling

  Outputs: 1st version of parallel IR with abstract communication between tasks

  Special Requirements: -

- **Parallel Code Generation**

  Inputs: in-memory IR of parallel program

  Outputs: set of C source and header files containing the platform optimized parallel program.

  Special Requirements: -

- **Sequential / Parallel Code Comparison**

  Inputs: Application executable

  Outputs: Outcome of the test as succeeded or failed

  Special Requirements: -

2.2.2.2 User Constraints for Parallelization

There are mainly two emmtrix functions for the IIS-use-case constraints for the generated code to be data parallel. For more information, please refer the documentation *emmtrix Code Generator Reference Guide* [7]

```
//EMX?: emx_var_split(data,sizex,sizey,…);
```

The function above is used for splitting the data array into sub-tiles in corresponding dimensions, in order to be able to distribute those tiles among cores. For that purpose the function below is used before each C-loop, where the data-distribution among cores is required.

```
//EMX?: emx_perf_loopfission();
```

The user should split the data according to the number of cores which are available on the target hardware, by inserting these functions to the desired locations at the Scilab source code

The split data arrays are then propagated internally to and from the functions such that the memory tiles of the scratchpad are fixed with tiles of corresponding data over function calls, with the requirement that the size of the scratchpad is big enough.

3. Demonstration and Evaluation Results

3.1 Enhanced Ground Proximity Warning System (DLR)

3.1.1 Code Generation

3.1.1.1 Code Generation 01 (CG 01)

The test is executed in two levels. The first level targeted the model elements that have been presented in Figure 2, whereas the second level is applied to the overall model.

Scilab scripts are successfully generated for the following model elements and the overall model:

Mode 1: Excessive Rate of Descent

Mode 2: Excessive Terrain Closure Rate

Mode 3: Altitude Loss After Take-off

Mode 4: Unsafe Terrain Clearance

Mode 5: Deviation Below Glideslope

Data Output Management

The achievement of OBJ1 and OBJ5 is demonstrated by the successful generation of the Scilab scripts. 30 supported Xcos blocks are described in D3.1 *Intermediate Representation and Sequential Code Generation*. In iteration 1 of the ARGO EGPWS 14 of them are used (Table 1). The list is further enhanced by 4 more blocks that are required later in model development stage (Table 2).

Table 1: Support Scilab /Xcos Block Utilization in ARGO EGPWS

Scilab Xcos Blocks	Status	Scilab Xcos Blocks	Status
ABS		INTRPLBLK_f	Tested
BIGSOM_f		LOGICAL_OP	Tested
CLOCK_c	Tested	NRMSOM_f	
CONST_m	Tested	OUT_f	Tested
CONVERT		POWBKL_f	
DEMUX		PRODUCT	
DERIV		RELATIONALOP	Tested
DOLLAR_f	Tested	SATURATION	Tested

EXTRACTOR	Tested	SINBLK_f	
FROM		SQRT	
FROMWSB		SUMMATION	Tested
GAINBLK_f	Tested	SWITCH2_m	Tested
generic_block3		SUPER_f	Tested
GOTO		TANBLK_f	
IN_f	Tested	TrigFun	

Table 2: Newly Added Blocks to the Supported List

Scilab Xcos Blocks	**Status**	**Scilab Xcos Blocks**	**Status**
INTRP2BLK_f	Tested	MUX	Tested
MAXMIN	Tested	TIME_f	Tested

For OBJ2, the subjective assessment about the comparison of Scilab/Xcos Front-End and MATLAB/Simulink would be as follows: The current code generation front-end that is being provided by Scilab/Xcos is simple to use and straight forward. It provides the comfortable automation in code generation like MATLAB/Simulink. However, it is to be mentioned that the maturity and the feature set of COTS Simulink Coder exceed the limits of what is provided within the scope of the ARGO project. An example that Scilab/Xcos Front-End fails to provide is diagnostic features which will create warnings for possible problems.

3.1.1.2 Code Generation 02 (CG 02)

In this step, we successfully generated sequential C-code which is located in a folder named *results**scilab2c* (Figure 19). The file set includes C files that correspond to *x_scenario.sce* and *x.sci* and all the dependencies with a make file. Figure 19 depicts the output C-files for the overall ARGO GPWS Scilab script.

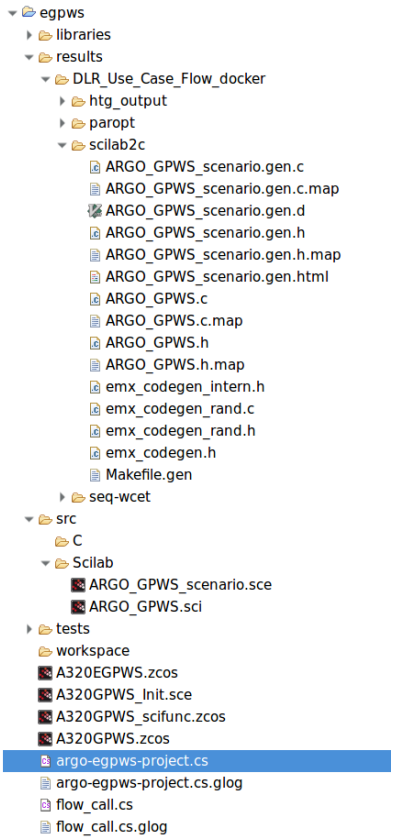


Figure 19: ARGO Toolchain Project Structure

The following steps were followed to prepare the setup for the test case:

- The Docker image that contains the ARGO Toolchain is set up.
- The projects are prepared for each code generation case.

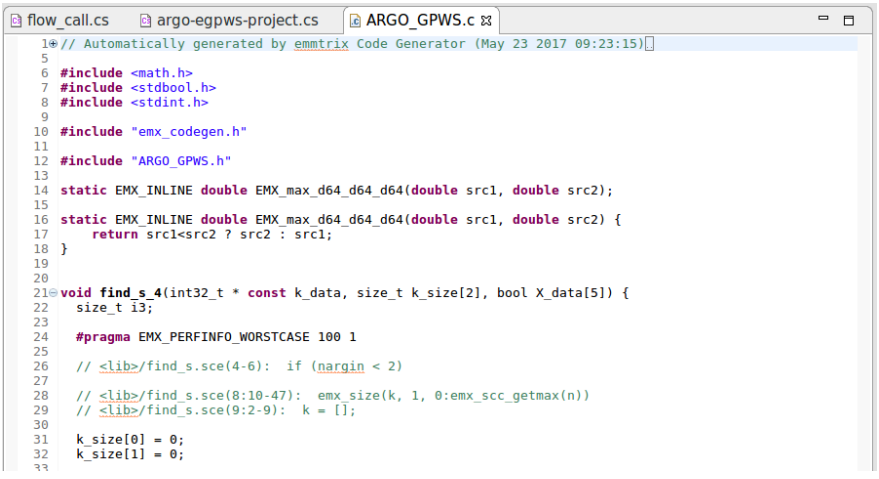


Figure 20: ARGO GPWS Code Excerpt

When the compiler scripts are executed, sequential C-code is successfully generated. A sample excerpt is given in Figure 20. The automation script compiled the auto generated file and executed it with random inputs.

The achievement of OBJ1 and OBJ5 is demonstrated by the successful generation of the sequential C code. For OBJ2, the subjective assessment about the comparison of ARGO Toolchain and MATLAB/Simulink would be as follows: Currently, the ARGO Toolchain is under development. The snapshot that was executed in this test was stable and successfully generated code, but the current user experience and feature set is not yet comparable to Simulink Coder or other COTS code generation tools.

3.1.1.3 Code Generation 03 (CG 03)

In this step, we successfully generated parallel C-code for a 4-core target architecture. One of the cores is assigned to data processing and the code is optimized for the other 3 cores.

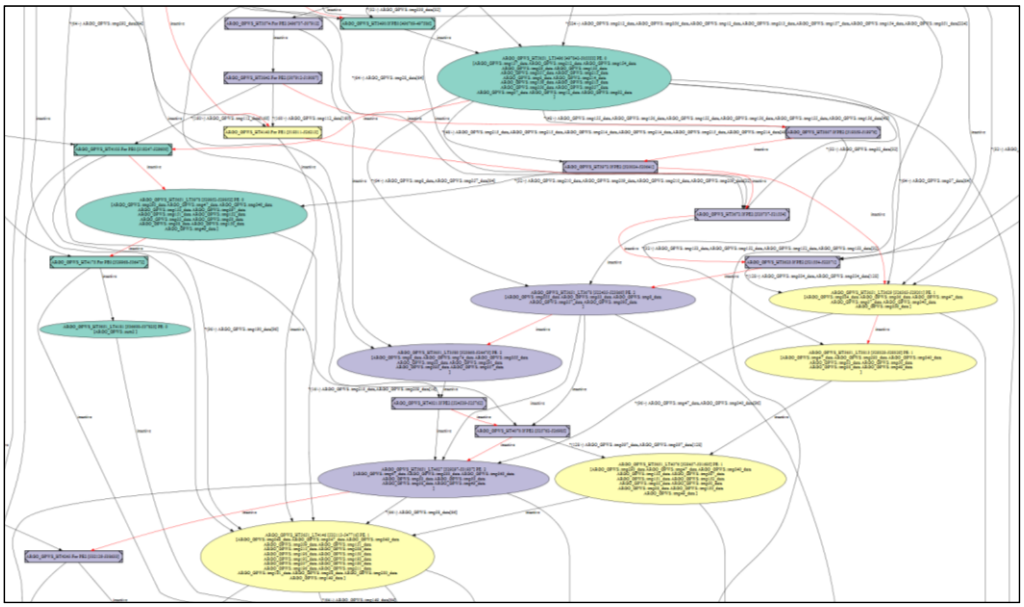


Figure 21: HTG Excerpt from Optimized Parallel ARGO GPWS Code

The following steps were followed to prepare the setup for the test case:

- The Docker image that contains the ARGO Toolchain is set up.
- The projects are prepared for each code generation case.

When the compiler scripts are executed, parallel C-code is successfully generated. The generated code files were located in a folder named *results\codegen\pout*. The Hierarchical Task Graph (HTG) files for the optimized parallel C-code are presented under results\paropt\dotsol. An excerpt from the HTG of optimized parallel C-code for ARGO EGPWS is given in Figure 21. Three different colours designate three processors.

The generated parallel C-code was compiled using InvasIC API on PC platform (in the ARGO Docker container) and the executable is executed with random inputs.

The achievement of OBJ1 and OBJ5 is demonstrated by the successful generation of the parallel C code. For OBJ2, the subjective assessment about the comparison of ARGO

Toolchain and MATLAB/Simulink would be as follows: Currently, the ARGO Toolchain is under development. Parallel code generation is the core focus of the development effort. The snapshot that was exercised in this test was just stable. Successful parallel code generation was only possible with heavy involvement of the developers. The current user experience is not yet comparable to Simulink Coder or other COTS code generation tools.

3.1.2 Software-in-the-Loop Testing

3.1.2.1 Software-in-the-Loop Testing 01 (SIL 01)

The tests are executed using a test harness developed by DLR. The harness executes the Scilab scripts with the selected test inputs, compares their outputs with the expected outputs and generates test reports. A sample test report is provided in Figure 22.

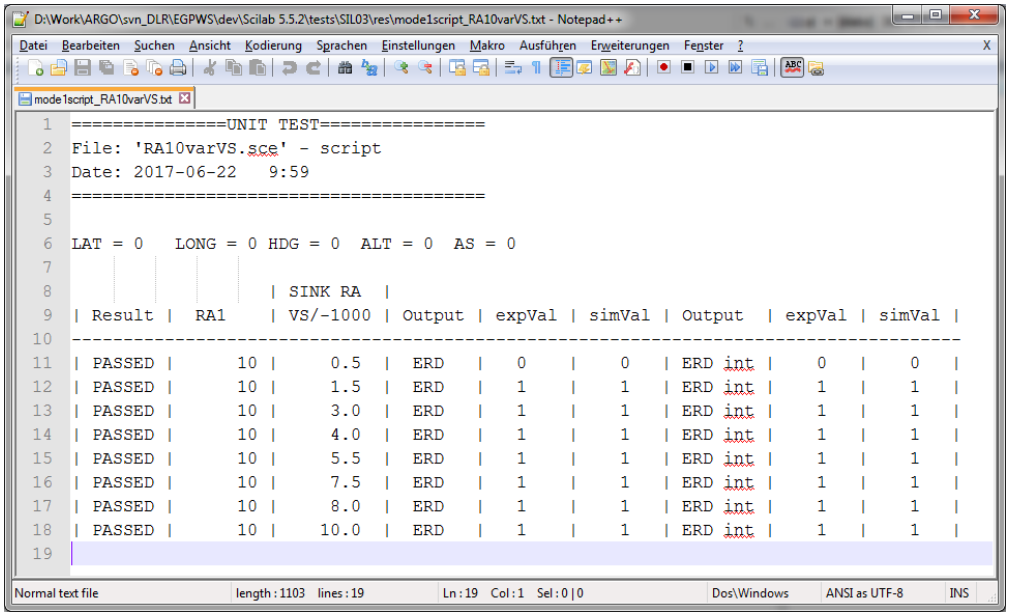


Figure 22: Sample SIL 01 Test Report

In total, 358 test cases are executed for the following 6 scripts under test with the following distribution:

Mode 1: Excessive Rate of Descent: 67 Test Cases

Mode 2: Excessive Terrain Closure Rate: 14 Test Cases

Mode 3: Altitude Loss After Take-off: 157 Test Cases

Mode 4: Unsafe Terrain Clearance: 16 Test Cases

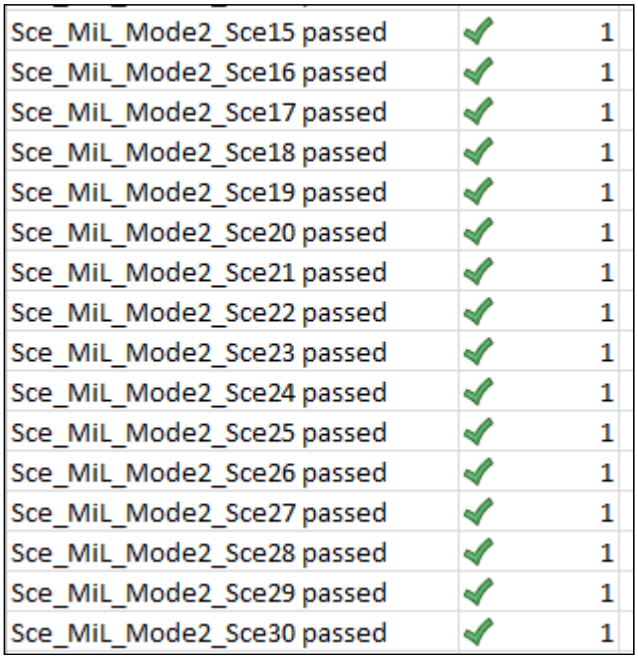
Mode 5: Deviation Below Glideslope: 90 Test Cases

Data Output Management: 14 Test Cases

All test cases were rated successful. The achievement of OBJ1 and OBJ5 is demonstrated by verifying that the Scilab scripts generated from Scilab/Xcos blocks that are listed in Table 1 and Table 2 are functioning properly in the comprehensive time-critical use case ARGO EGPWS.

3.1.2.2 Software-in-the-Loop Testing 02 (SIL 02)

The tests are executed using a test harness developed by DLR. The harness executes the scenarios, compares the EGPWS outputs in these scenarios with the expected ones and generates test reports. An excerpt from a test report is provided in Figure 23.



Sce_MiL_Mode2_Sce15 passed	✓	1
Sce_MiL_Mode2_Sce16 passed	✓	1
Sce_MiL_Mode2_Sce17 passed	✓	1
Sce_MiL_Mode2_Sce18 passed	✓	1
Sce_MiL_Mode2_Sce19 passed	✓	1
Sce_MiL_Mode2_Sce20 passed	✓	1
Sce_MiL_Mode2_Sce21 passed	✓	1
Sce_MiL_Mode2_Sce22 passed	✓	1
Sce_MiL_Mode2_Sce23 passed	✓	1
Sce_MiL_Mode2_Sce24 passed	✓	1
Sce_MiL_Mode2_Sce25 passed	✓	1
Sce_MiL_Mode2_Sce26 passed	✓	1
Sce_MiL_Mode2_Sce27 passed	✓	1
Sce_MiL_Mode2_Sce28 passed	✓	1
Sce_MiL_Mode2_Sce29 passed	✓	1
Sce_MiL_Mode2_Sce30 passed	✓	1

Figure 23: An Excerpt from a Sample SIL 02 Test Report

In total, 1061 test cases are executed regarding the requirements of 5 modes with the following distribution:

Mode 1: Excessive Rate of Descent: 13 Test Cases

Mode 2: Excessive Terrain Closure Rate: 252 Test Cases

Mode 3: Altitude Loss After Take-off: 9 Test Cases

Mode 4: Unsafe Terrain Clearance: 759 Test Cases

Mode 5: Deviation Below Glideslope: 28 Test Cases

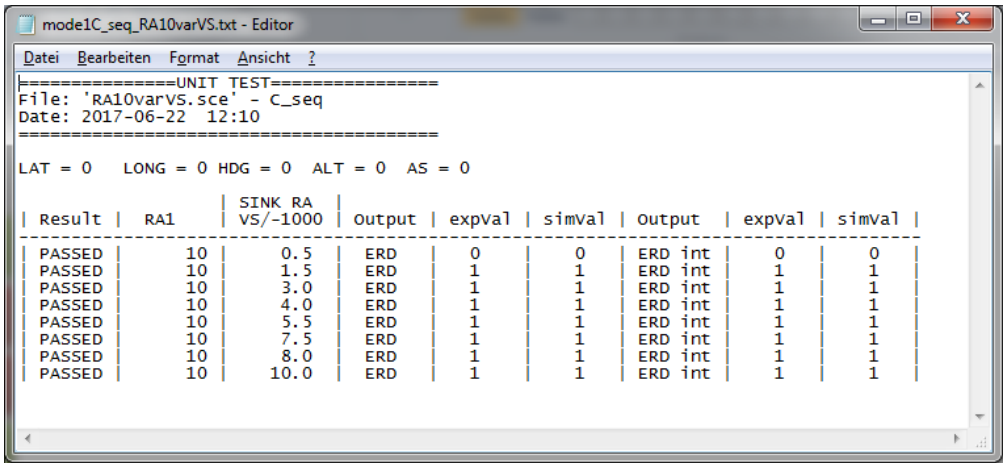
When these 1061 test cases are applied to the ARGO EGPWS Scilab/Xcos model (Model-in-the-Loop testing), currently 770 of them are rated successful. When the same tests are applied to the auto-generated Scilab scripts, the failing test cases are conformant with the ones failing with the Xcos EGPWS model. So the Xcos model and the auto-generated Scilab script are giving the same outputs in all cases.

It is important to note that the debugging and bug fixing of Xcos EGPWS model for the failing test cases is in progress.

The achievement of OBJ1 and OBJ5 is demonstrated by verifying that the Scilab scripts generated for the Scilab/Xcos blocks that are listed in the Table 1 and Table 2 are functioning properly in the comprehensive time-critical use case ARGO EGPWS.

3.1.2.3 Software-in-the-Loop Testing 03 (SIL 03)

The tests are executed using a test harness developed by DLR. The harness executes the sequential C-files with the selected test inputs, compares their outputs with the expected outputs and generates test reports. A sample test report is provided in Figure 24.


mode1C_seq_RA10varVS.txt - Editor

Datei Bearbeiten Format Ansicht ?

```
==============UNIT TEST================
File: 'RA10varVS.sce' - C_seq
Date: 2017-06-22  12:10
=======================================

LAT = 0   LONG = 0 HDG = 0  ALT = 0  AS = 0

                  | SINK RA  |
| Result |  RA1   | VS/-1000 | Output | expVal | simVal | Output  | expVal | simVal |
-------------------------------------------------------------------------------------
| PASSED |     10 |      0.5 |  ERD   |    0   |    0   | ERD int |    0   |    0   |
| PASSED |     10 |      1.5 |  ERD   |    1   |    1   | ERD int |    1   |    1   |
| PASSED |     10 |      3.0 |  ERD   |    1   |    1   | ERD int |    1   |    1   |
| PASSED |     10 |      4.0 |  ERD   |    1   |    1   | ERD int |    1   |    1   |
| PASSED |     10 |      5.5 |  ERD   |    1   |    1   | ERD int |    1   |    1   |
| PASSED |     10 |      7.5 |  ERD   |    1   |    1   | ERD int |    1   |    1   |
| PASSED |     10 |      8.0 |  ERD   |    1   |    1   | ERD int |    1   |    1   |
| PASSED |     10 |     10.0 |  ERD   |    1   |    1   | ERD int |    1   |    1   |
```

Figure 24: Sample SIL 03 Test Report

In accordance with SIL 01, 358 test cases are executed for the following 6 scripts under test with the following distribution:

Mode 1: Excessive Rate of Descent: 67 Test Cases

Mode 2: Excessive Terrain Closure Rate: 14 Test Cases

Mode 3: Altitude Loss After Take-off: 157 Test Cases

Mode 4: Unsafe Terrain Clearance: 16 Test Cases

Mode 5: Deviation Below Glideslope: 90 Test Cases

Data Output Management: 14 Test Cases

All test cases were rated successful. The achievement of OBJ1 and OBJ5 is demonstrated by verifying that the auto-generated sequential C-code generated from the Scilab/Xcos blocks that are listed in the Table 1 and Table 2 is functioning properly in the comprehensive time-critical use case ARGO EGPWS.

3.1.2.4 Software-in-the-Loop Testing 04 (SIL 04)

The tests are executed using a test harness developed by DLR. The harness executes the scenarios, compares the EGPWS outputs in these scenarios with the expected ones and generates test reports.

In accordance with SIL 02, 1061 test cases are executed regarding the requirements of the 5 modes with the following distribution:

Mode 1: Excessive Rate of Descent: 13 Test Cases

Mode 2: Excessive Terrain Closure Rate: 252 Test Cases

Mode 3: Altitude Loss After Take-off: 9 Test Cases

Mode 4: Unsafe Terrain Clearance: 759 Test Cases

Mode 5: Deviation Below Glideslope: 28 Test Cases

As mentioned in SIL 02, when these 1061 test cases are applied to the ARGO EGPWS Scilab/Xcos model (Model-in-the-Loop testing), currently 770 of them are rated successful. When the same tests are applied to auto-generated sequential-C code, the failing test cases are conformant with the ones failing with Xcos EGPWS model. So the Xcos model and the auto-generated sequential-C code are giving the same outputs in all cases.

It is important to note that the debugging and bug fixing of the Xcos EGPWS model for the failing cases is in progress.

The achievement of OBJ1 and OBJ5 is demonstrated by verifying that the auto-generated sequential C-code generated from the Scilab/Xcos blocks that are listed in the Table 1 and Table 2 is functioning properly in the comprehensive time-critical use case ARGO EGPWS.

3.2 Polarization Image Processing System (Fraunhofer IIS)

3.2.1 Overview of the Test Results

For the unit tests, each of the following modules has been tested separately. In order to keep the testing simple, a constant matrix or tensor of appropriate dimension has been provided as input data, which allows for easy testing the numerical correctness of the results

The predefined matrix replacing the real input image data is generated as follows:

```
polTestPattern = [ 90, 135;
                   45,   0 ];
F = uint16(repmat(polTestPattern,244,324));
```

The predefined matrices for the used Gain/Offset Correction are as follows:

```
polTestPattern = [ 90, 135;
                   45,   0 ];
GainFrame = double(repmat((polTestPattern./1.0)',240,320));
OffsetFrame = double(repmat((polTestPattern./8192.0)',240,320));
```

The chosen pattern is easy to analyze for the validation of the outputs from Scilab and generated sequential and parallel C codes.

Following the principle of least effort, and in order to comply with the original dimensionalities, the input and output data arrays of the macros of unit tests are replicated to an array of appropriate dimension, if necessary.

The number of tiles for data parallelization of an array is chosen as 4. The tests are compiled for the target platform InvasIC emulator. The target contains 3 cores for data processing and 1 core for data communication. The program could still be parallelized.

The objective of the unit tests is to verify the functional correctness of the generated C code (parallel and sequential), while the aspects related to WCET awareness or performance are not considered in this stage of the project.

For each module, three unit tests have been conducted:

- A test of the original Scilab model (which is actually not a test but provides us with results considered as functionally correct, i.e., the *ground truth*). The Scilab model is executed on a PC platform (Linux or Windows)
- A test of the sequential C code generated by Emmtrix software tools inside the toolchain. The sequential C code is compiled for and executed on a PC platform
- A test of the parallel C code generated by the ARGO parallelizer. The parallel C code is compiled for and executed on the InvasIC target platform emulator.

The outputs of the three tests have been compared against each other. If they are identical, the test is considered as passed, **which is the case for all tested modules**.

In the following paragraphs, we give the details of the unit tests for each module. In order to preserve the output arguments ***aomp*** and ***dolp*** of macros **tCDPF*_macroname*(),** the generated output matrix is either replicated to a size of 640x960 if it is of size 640x480, or arguments ***aomp*** and ***dolp*** are increased in size in case of larger output arrays of size 640x480x3 or 640x480x4. (e.g. paInterpolation generates a tensor of dimensions 640x480x4, which is stacked into the output arguments (***aomp*** & ***dolp***) pairwise.)

Since the dimensionalities in Flow Test matches the original code, there was no need for such manipulations .

The figures in Section 3.2.2 show the parallelization of units for the Unit Tests and of the processing pipeline of the whole code for the Flow Test. The parallelization degree can be seen in the number of different colors used for rectangular nodes. Note that those in the graph for the Flow in Figure 32 correspond to blocks followed one by another representing our whole pipeline.

For a better view of the graphs, please refer to the directory structure explanation in the next section, to find the locations of these graphs. If you take a closer look, you can see black and red arrows differentiating between tasks with real data dependencies and without any dependencies, but still scheduled sequentially, because of lack of resources. This might mean that there is room for more parallelization, but not necessarily. There are tasks that are connected with an arrow, bearing the description “inactive”. This means that no communication takes place between tasks, suggesting that these tasks are on the same core.

3.2.2 Detailed Test Results

a) **paROI(): Succeeded**
   Input data: Constant 648x488 matrix
   Output data: Constant 640x480 matrix

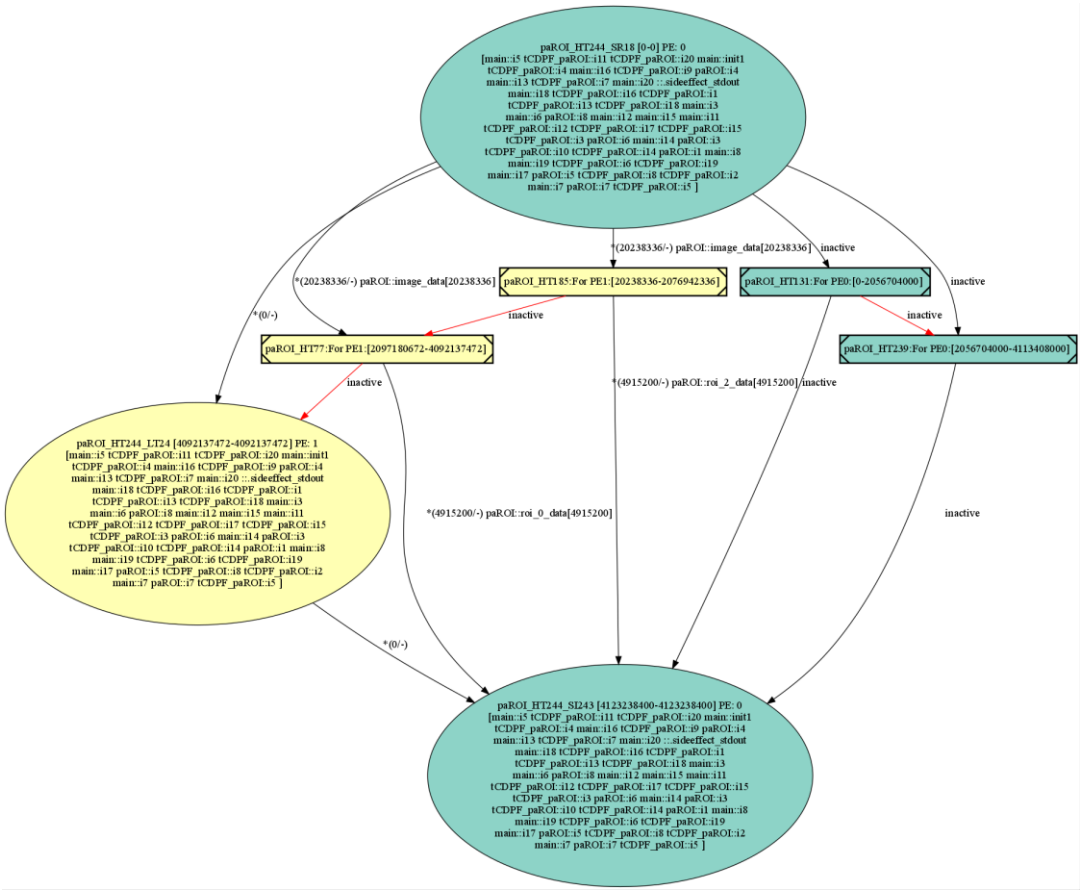


Figure 25: Parallelization for paROI unit

b) paGOCorrection():Succeeded
Input data: Constant 640x480 matrix
Output data: Constant 640x480 matrix

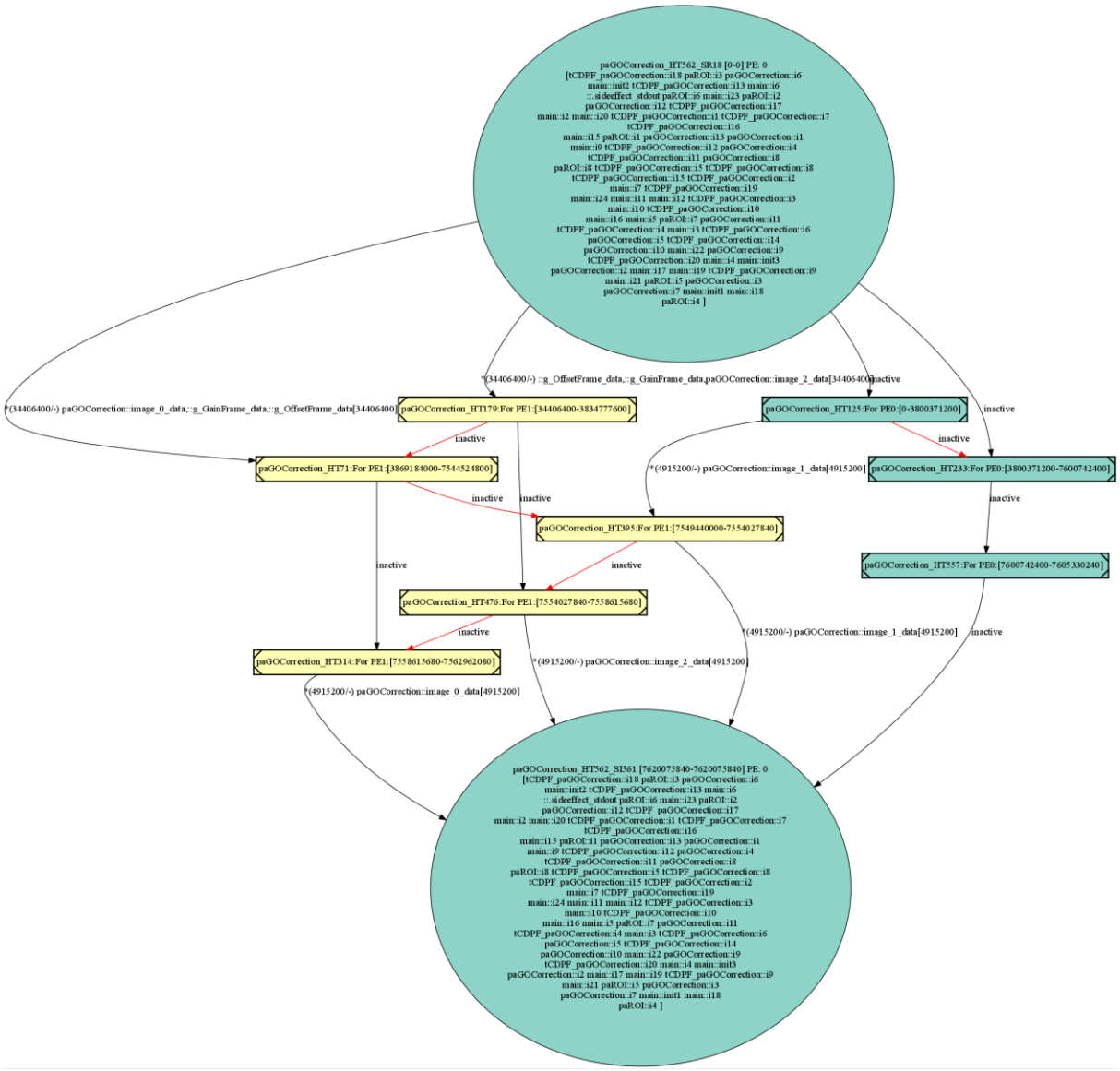


Figure 26: Parallelization for paGOCorrection unit

c) paDenoise():Succeeded
Input data: Constant 640x480 matrix
Output data: Constant 640x480 matrix

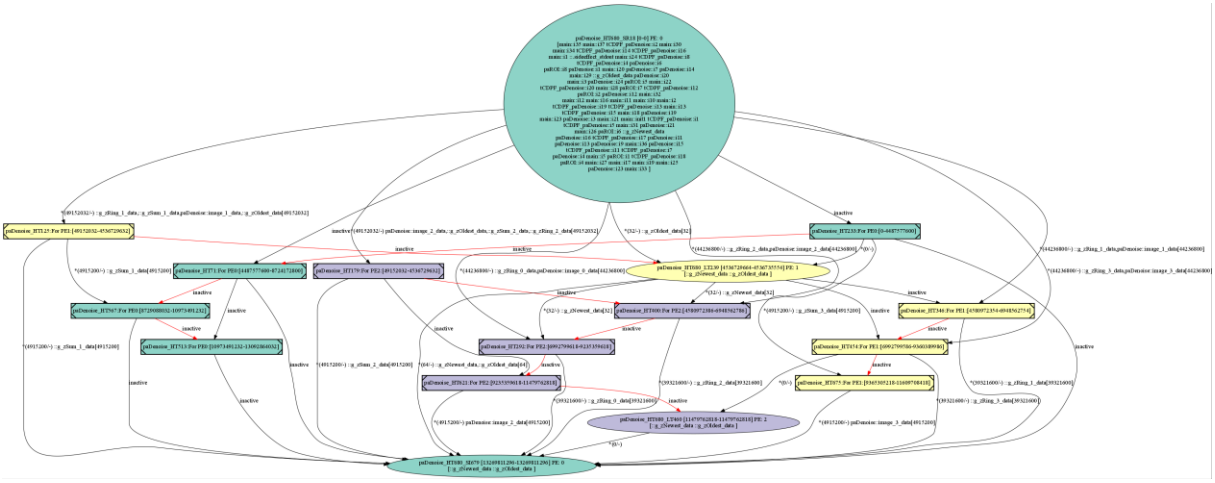


Figure 27: Parallelization for paDenoise unit

d) paInterpolation():Succeeded
Input data: Constant 640x480 matrix
Output data: Constant 640x1920 matrix

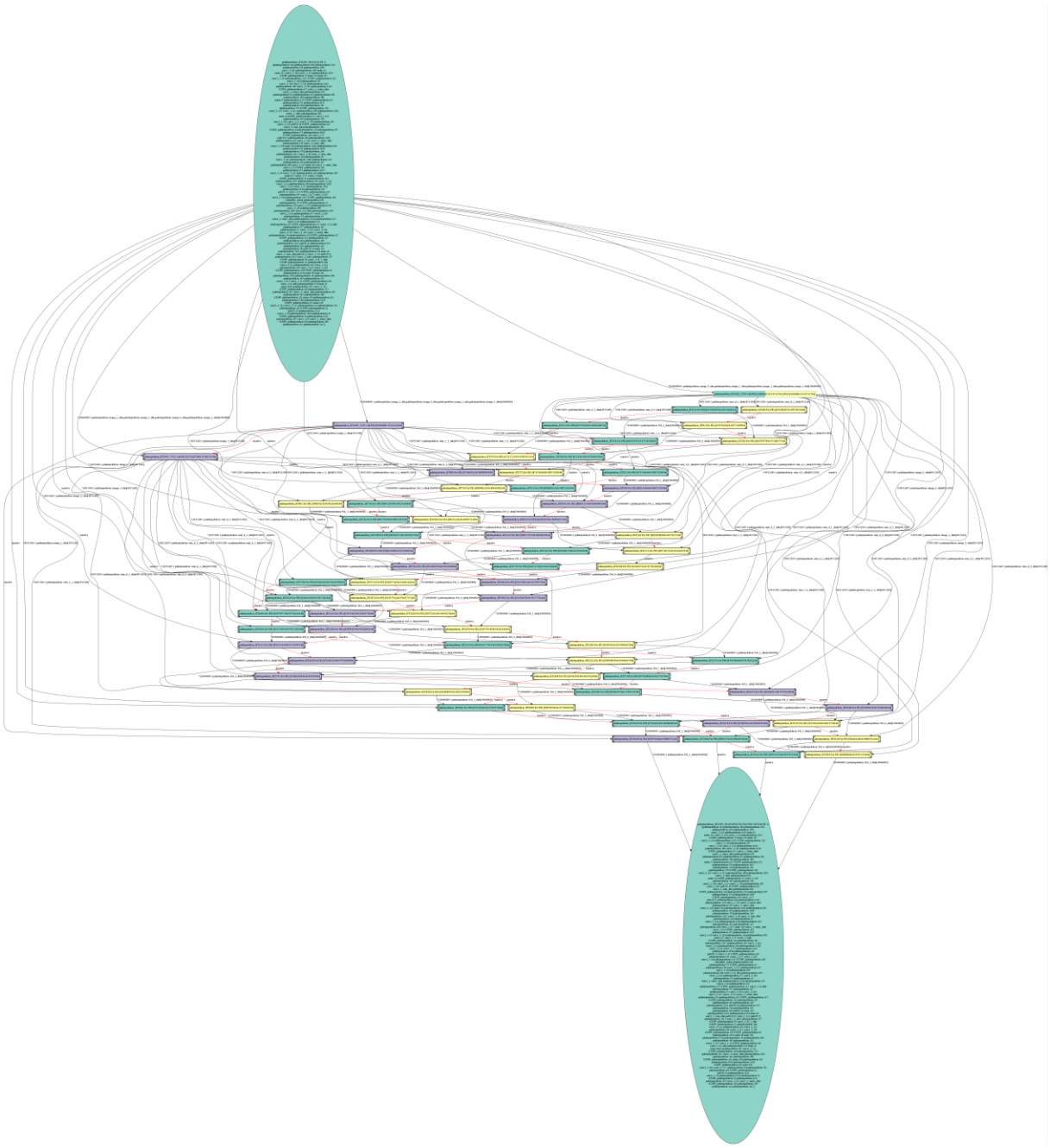


Figure 28: Parallelization for paInterpolation unit

e) **paStokes():Succeeded**
Input data: Constant 640x1920 matrix
Output data: Constant 640x1440 matrix

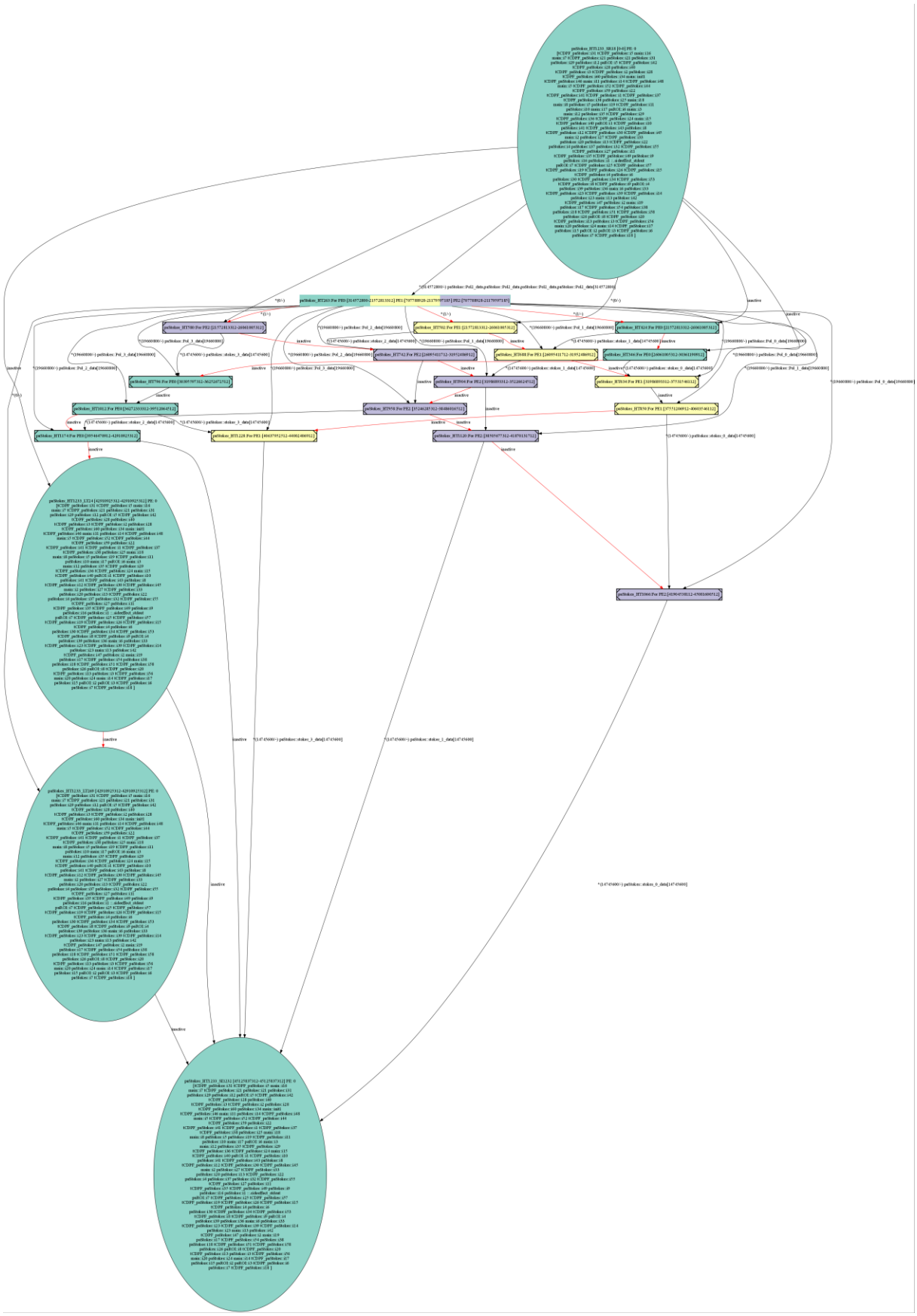


Figure 29: Parallelization for paStokes unit

f) paAomp():Succeeded
Input data: Constant 640x1440 matrix
Output data: Constant 640x480 matrix

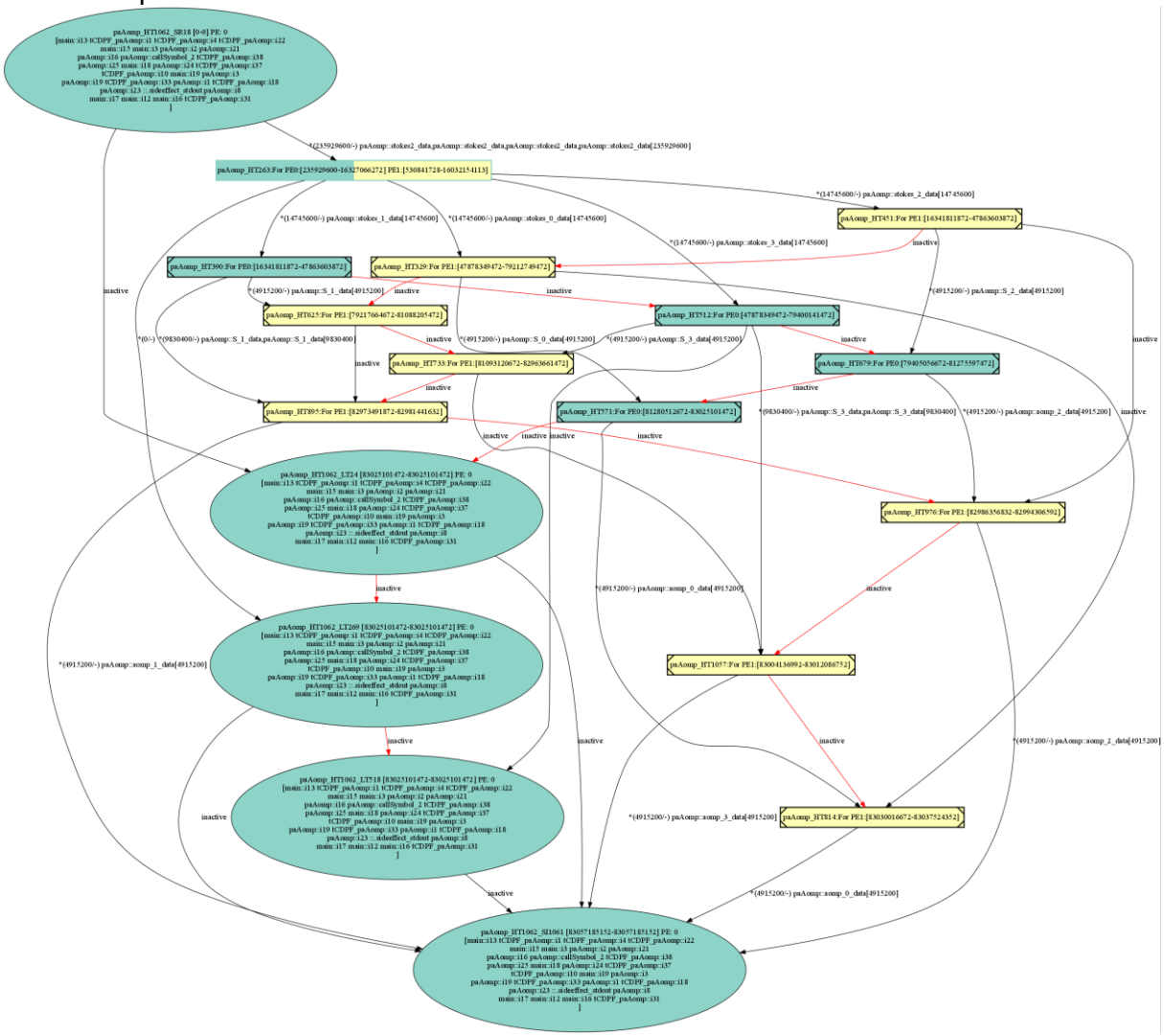


Figure 30: Parallelization for paAomp unit

g) paDolp():Succeeded
Input data: Constant 640x1440 matrix
Output data: Constant 640x480 matrix

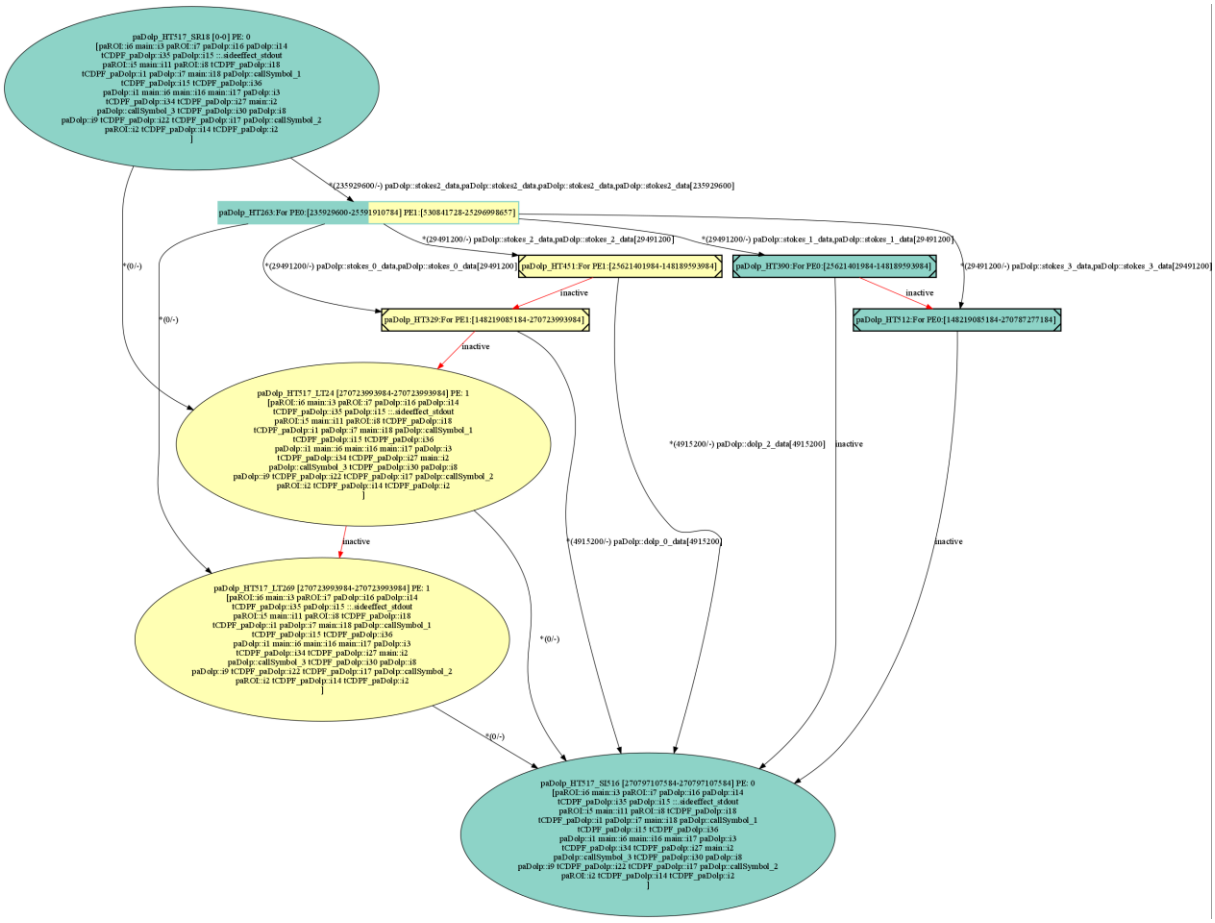


Figure 31: Parallelization for paDolp unit

h) Flow Test: Succeeded
Input data: Constant 648x488 matrix
Output data: Constant 640x480 matrix

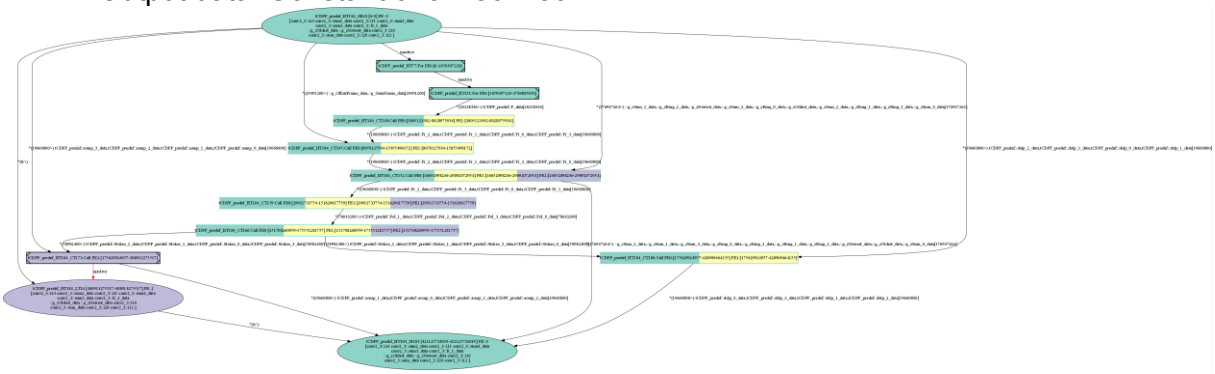


Figure 32: Parallelization for Flow

The log file for each test can be found in the corresponding directory of the test. All the necessary output files mentioned in Section 2.2.2.1 are generated. To elaborate this further, we have to look at the structure of this directory:

```
results.
├──IIS_Flow_Test_predef
```

```
│   ├── codegen
│   │   ├── pout
│   │   │   └── obj
│   │   └── reordered
│   │       └── obj
│   ├── eval
│   │   └── build
│   │       ├── parallel
│   │       ├── reordered
│   │       └── sequential
│   ├── htg_output
│   │   ├── dotty_htg
│   │   ├── dot_ssa
│   │   └── dot_ssa_htg
│   ├── paropt
│   │   ├── dots
│   │   ├── dotsol
│   │   └── dot_before_paropt
│   ├── scilab2c
│   │   └── obj
│   └── seq-wcet
│       ├── ais
│       ├── codegen
│       ├── configure
│       ├── dotty_ANNOTATED
│       │   ├── _CDFG
│       │   └── _HTG
│       └── XTC
│           ├── XML_Reports_EM
│           └── XML_Reports_SPM
└── IIS_Unit_Tests_predef
    ├── macroname1
    │   ├── codegen
    │   │   ├── pout
    │   │   │   └── obj
    │   │   └── reordered
    │   │       └── obj
    │   ├── eval
    │   │   └── build
    │   │       ├── parallel
    │   │       ├── reordered
    │   │       └── sequential
    │   ├── htg_output
    │   │   ├── dotty_htg
    │   │   ├── dot_ssa
    │   │   ├── dot_ssa_htg
    │   │   └── IIS_Use_Case_Unit_Tests
    │   ├── paropt
    │   │   ├── dots
    │   │   ├── dotsol
    │   │   └── dot_before_paropt
    │   ├── scilab2c
    │   │   └── obj
    │   └── seq-wcet
    │       ├── ais
```

```
│       ├──codegen
│       ├──configure
│       ├──dotty_ANNOTATED
│       │   ├──_CDFG
│       │   └──_HTG
│       ├──IIS_Use_Case_Unit_Tests
│       └──XTC
│           ├──XML_Reports_EM
│           └──XML_Reports_SPM
├── macroname2
```

Since Flow Test is a standalone project, it does not have subprojects, as in the case of Unit Tests, which are subdivided to units (macros). Each project folder has generated console log files in its main directory.

scilab2c directory consists of generated .c and .h files from the first step of Section 2.2.2.1

htg_output directory consists of generated HTG graphs with annotations for IR & SSA, which correspond to the second step of section 2.2.2.1

seq-wcet folder corresponds to the fourth step of section 2.2.2.1 and consists of HTG graphs after the Code Transformation step, the sequential C code for core-level code-snippet WCET estimation and output files of the aiT analysis.

paropt folder corresponds to the fifth and sixth step of Section 2.2.2.1 and consists of HTG graphs for parallel program and parallel program IR

codegen folder consists of sequential and parallel program C source files.

eval folder has the generated executables and the resulting output values in corresponding log files for sequential, reordered (again sequential but adapted to toolchain) and parallel (for InvasIC emulator) programs.

The objectives of the workpackage for this increment can be found in D6.3 *Test Case Phases vs. Measurements* [2]. For the IIS use-case, the evaluation of the objectives is as follows:

OBJ1: It was easy to integrate our Scilab code into the ARGO Toolchain and to retrieve some initial results.Our phase 2b “Software-in-the-loop” tests are conducted successfully.

OBJ2: Objective 2 is partially achieved for this increment. The second part of this objective will be considered in iteration 2 of this document.

For the first part of the objective, we can also divide our evaluation into two.

The initial assessment for the development time can be found in D6.2 *Test Cases and Requirements Specification* [3]. This assessment suggests a development time reduction from 8 months to approximately 2.5 months. Until now, we have effectively invested about 7 weeks for the integration of our code into the ARGO Toolchain. The waiting periods for required licenses and other delays caused by administrative issues and other projects are discarded by that assessment.

Compared to our initial assessment this value might look much less, but please note that the Xcos part of the phase 2a “Model-in-the-loop” should be added to this time afterwards, which is planned for the second increment of this document, because of the unsuitability to the IIS use-case.

OBJ5: Here again, we will discard the phase 2a “Model-in-the-loop” and phase 3 “Hardware-in-the-loop” parts of the objective and stage them to the second increment of this document.

Apart from those, our Unit Tests and Flow Test representing our whole pipeline have successfully run on the InvasIC platform emulator.

4. Conclusion and Future Work

4.1 Evaluation Summary

In this first increment we successfully conducted an evaluation of the ARGO Toolchain. It was concluded with successful code generation and extensive Software-in-the-loop testing (more than 1000 test cases).

Regarding the presented use cases, the number of tasks currently included is limited and the consortium is to extend it within the development framework of the ARGO Toolchain. The developed evaluation infrastructure will be readily available for further development of the ARGO Toolchain and further expanded and enhanced for the second increment where Hardware-in-the-loop testing and further evaluation will be performed.

4.2 Future Work

4.2.1 Enhanced Ground Proximity Warning System

The next step for the DLR use-case is to develop a better way to reintroduce sequential and parallel C-code into Scilab/Xcos schemas for a streamlined test flow for SIL 04 and SIL 05. Next, the tests will be integrated to the ARGO CI in order to check the ARGO Toolchain in an automated manner. While it is still too early to really evaluate the user experience, the preliminary results show that the toolchain can be (easily) integrated and generates correct code.

Afterwards, WCET constraints will be considered and user intervention scenarios in parallelization will be exercised.

4.2.2 Polarization Image Processing System

The next step for the IIS use-case is to include a TCP/IP interface and eventually the whole flow to the toolchain environment and complete the tests for it. Afterwards, WCET constraints will be considered. Furthermore, the model is planned to be specified in XCOS instead textual Scilab code and will be extended by at least one more computation intensive processing step.

5. References

[1] *IEEE Standard for Software and System Test Documentation," in IEEE Std 829-2008 , vol., no., pp.1-150, July 18 2008 doi: 10.1109/IEEESTD.2008.4578383*

[2] *ARGO Deliverable D6.3 Test cases and Design and Implementation*, the ARGO Consortium, Umut Durak (DLR), David Mueller (DLR), Koray Kasnakli, Dr. Imen Fassi, Dr. Isabelle Puaut, Dr. Panayiotis Alefragis, Dr. Marcus Bednara, Version 1.05, March 2017.

[3] *ARGO Deliverable D6.2 Test Cases and Requirements Specification*, the ARGO Consortium, Umut Durak (DLR), David Mueller (DLR), Marcus Bednara (Fraunhofer IIS), Version 1.00, June 2016.

[4] Jürgen Gotschlich, Torsten Gerlach and Umut Durak. 2014. 2Simulate: A distributed real-time simulation framework, ASIM STS/GMMS Workshop, Reutlingen, Germany.

[5] *ARGO Deliverable D5.1 Interface Specification*, the ARGO Consortium, George Goulas, Panayiotis Alefragkis (TWG), Steven Derrien, Isabelle Puaut (UR), Simon Reder, Harald Bucher (KIT), Reinhold Heckmann (Absint), Version 1.00, June 2016.

[6] *ARGO Deliverable D3.2 Algorithms for cross layer programming,* the ARGO Consortium, Panayiotis Alefragkis (TWG), TWG, KIT, Scilab, UR1, Version 0.07, July 2017.

[7] *emmtrix Code Generator Reference Guide,* emmtrix Technologies GmbH, April 2017