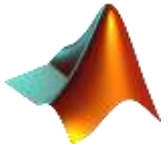
GEEDS - Group Electronics Expertise and Development Services



An Automated Generic Workflow for Code Coverage with Embedded Coder

Mohammad Raouf

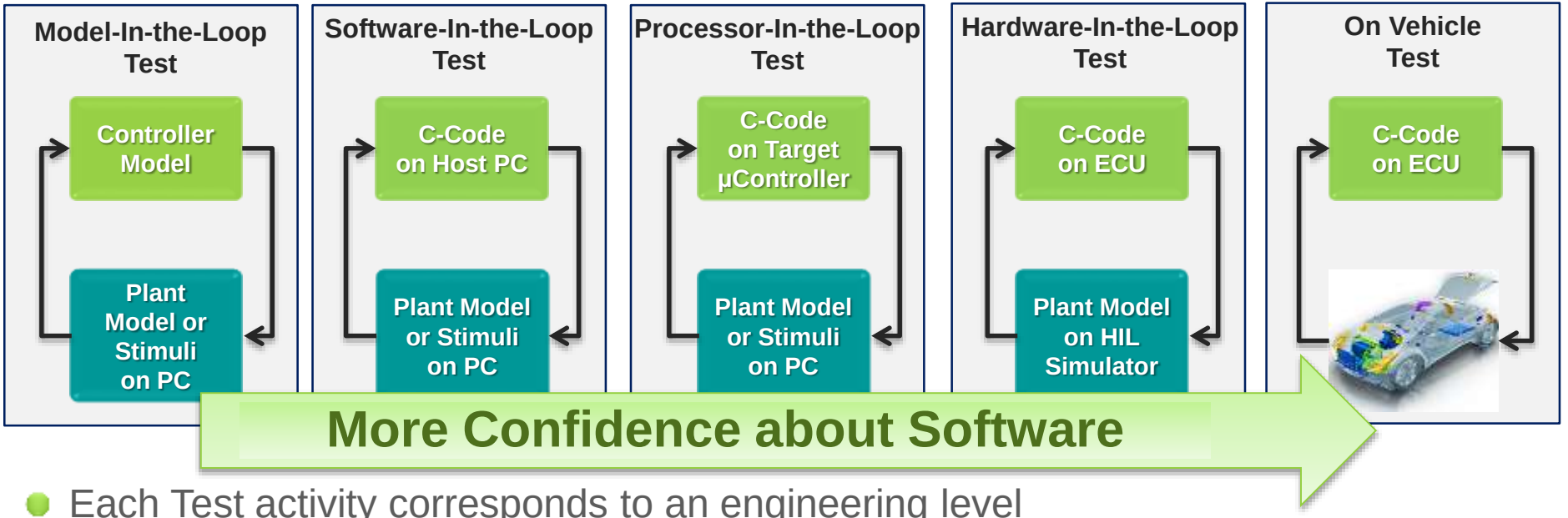
Mathworks Automotive Conference 2016
Plymouth, Michigan, *May12th 2016*

May 2016

Agenda

Agenda

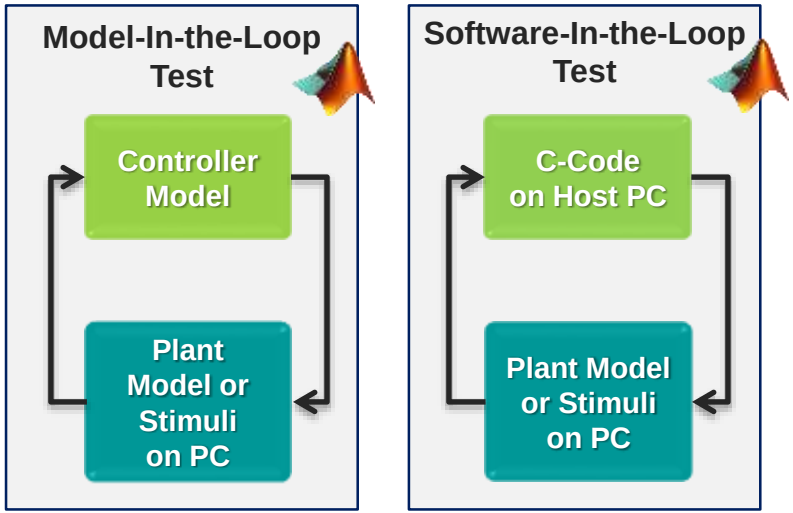
Testing in Model-Based Design



- Each Test activity corresponds to an engineering level
- At each level, there is a possibility that a test case fails
- At each level, both test results and coverage are needed

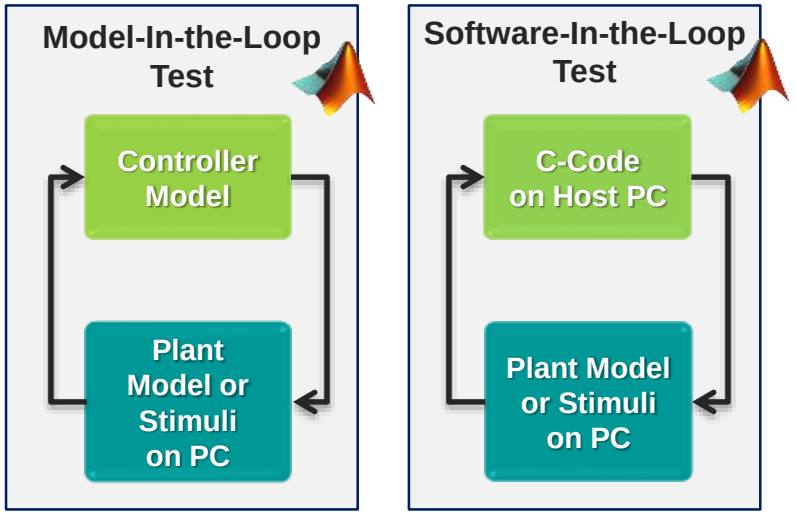
Focus on MIL & SIL testing

- Model-In-the-Loop Test
  - Objective: verify correctness of model functionality vs. either requirements or reference model.
  - Prepare and execute test cases for the model under test
    - Format: Mat file, Signal Builder, Custom Textual templates,…
- Software-In-the-Loop Test
  - Objective: verify correctness of functionality of generated code vs. either requirements or back-to-back with MIL.
  - Re-use same test cases used in MIL



Main Outcomes from MIL & SIL testing

- Model-In-the-Loop
  - MIL execution results
  - Model coverage report
    - Make sure that all possible paths in the model are covered
    - Depending on projects ISO level, coverage up to MC/DC level can be needed
    - Tool: V&V toolbox
- Software-In-the-Loop
  - SIL execution results
  - Code coverage Report
    - Make sure that all possible paths in the code are covered
    - Depending on projects ISO level, coverage up to MC/DC level can be needed
    - Tool:
      - V&V toolbox supported starting ***MATLAB R2016a***
      - External tool (same as used for manual coding)



Do we need Code-Coverage?

INTERNATIONAL STANDARD **ISO 26262-6**

Part 6: **Product development at the software level**

9.4.6 The test environment for software unit testing shall correspond as closely as possible to the target environment. If the software unit testing is not carried out in the target environment, the differences in the source and object code, and the differences between the test environment and the target environment, shall be analysed in order to specify additional tests in the target environment during the subsequent test phases.

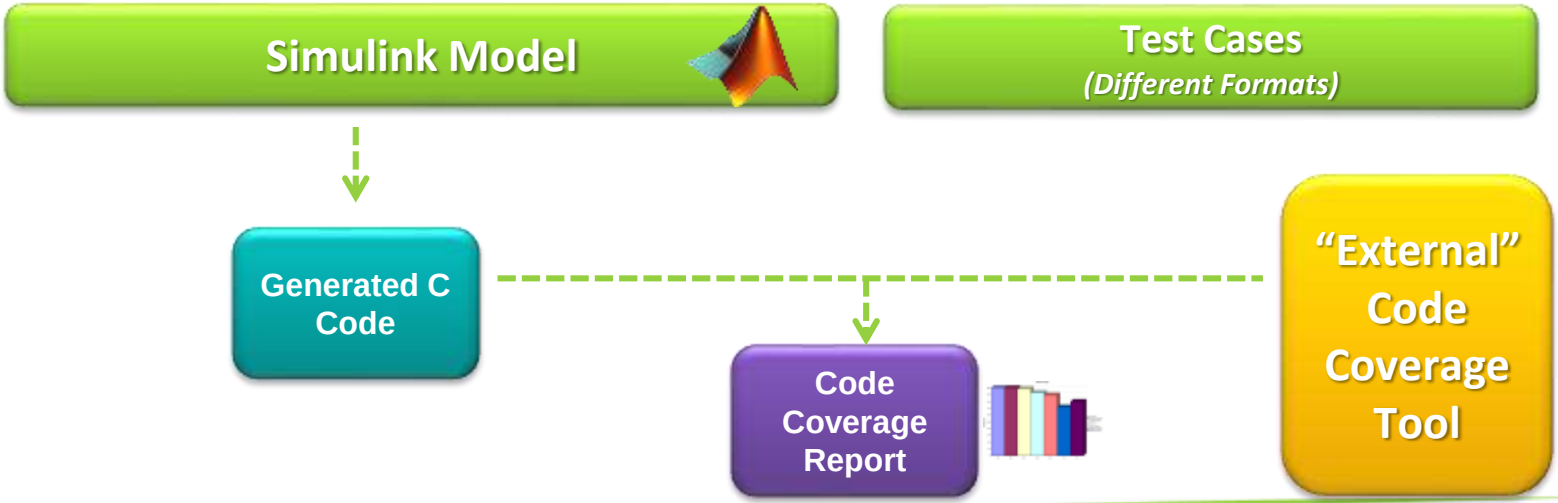
- **Yes**, we still need Code-Coverage reports in addition to Model-Coverage reports, at least for the following reasons:
  - As per the ISO-26262, the testing environment shall correspond as closely as possible to the target environment. Then, executing on code is closer than the model to target environment.
  - Model coverage and code coverage are completely separate, due to various code generation options and settings.

Agenda

Valeo Code Coverage Objective

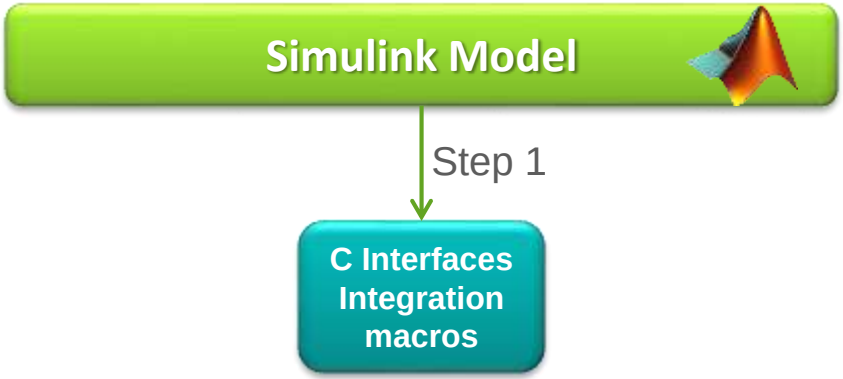
- Objective: using a Simulink model and Test Cases, create a **compilable stand-alone** software package to be used with an external tool **available at the organization,** and so to get the **code coverage reports**.

- The Valeo workflow consists of 6 steps that will be explained.



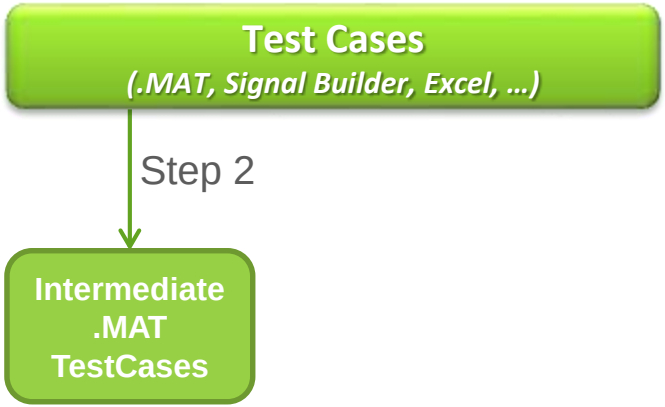
Step 1: Integration with Embedded Coder code

- In order to compile TestCases and Embedded Coder auto-generated code, we generate a wrappers of C-macros.
  - Macros for parsing signals names, offsets and resolutions.
  - Example: macro for mapping the signals name (as used in the test cases) to the Embedded Coder generated name
    - **#define Script_Var(signal) "MODEL_NAME"_U.signal**



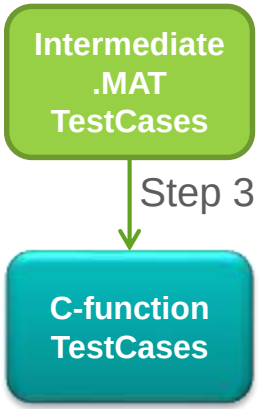
Step 2: Transform Test Cases into MAT Format

- Test cases could be in one of different formats:
  - ".MAT" file
  - Signal Builder
  - Customized template: Excel sheets
  - ...etc.
- We standardize the format by transforming different formats of test cases into a single *TimeSeries MAT* format (per each test case).



Step 3: Transforming MAT into C

- Since we need a standalone software package, we transform the test cases standardized format into C code.
- Key points:
  - Each test case is transformed into a separate C-function.
  - A "**Counter**" variable is introduced
  - Each C-function is composed of a single Switch-Case, controlled by "**Counter**"
  - At each "counter" value, the inputs are updated with the corresponding values in the stimulus, taking into account:
    - Resolution, and Offset
    - Integration C-macros are used
  - Flag is set to indicate end-of-test after last input



Example

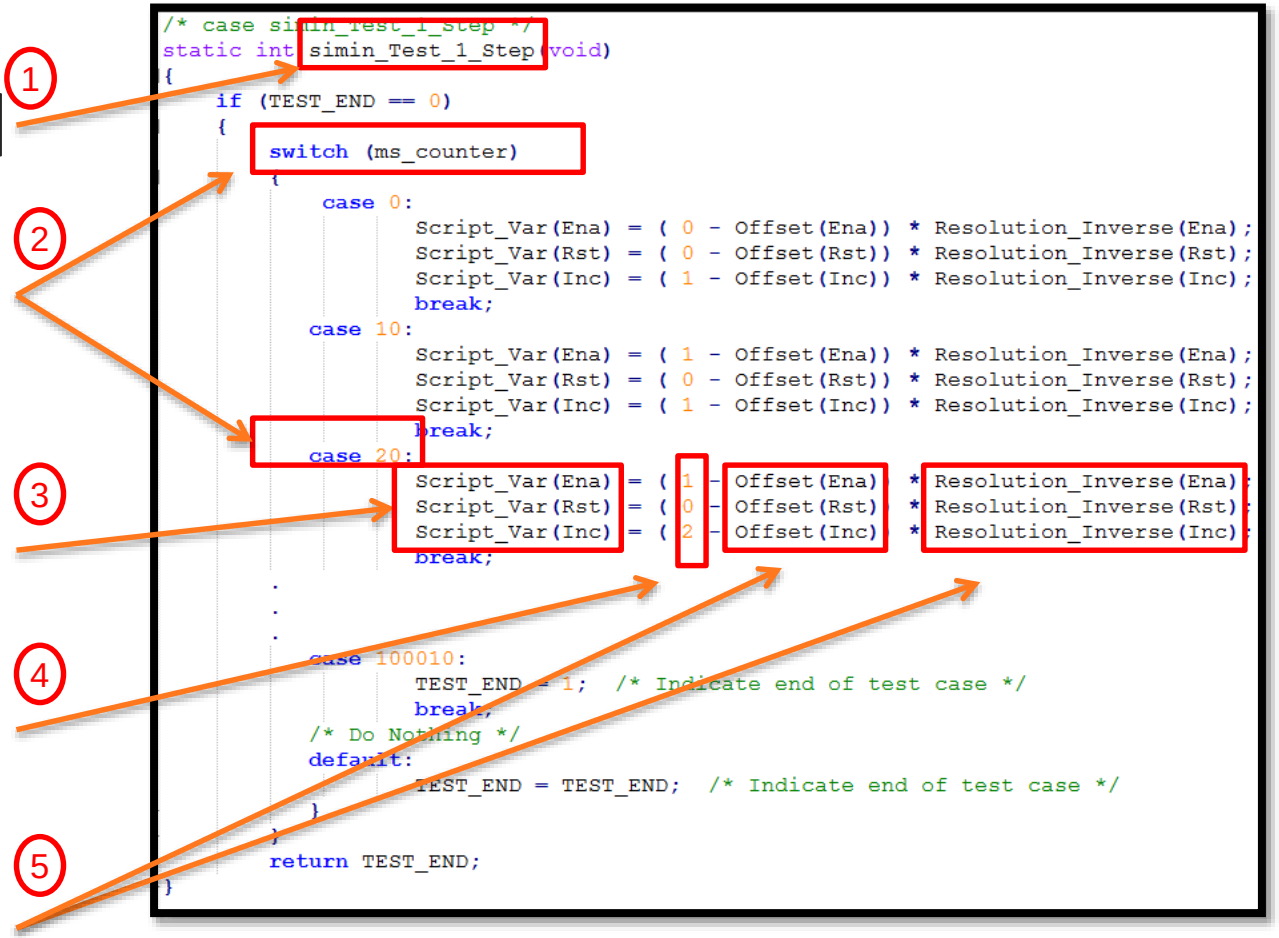
C-function per each test case

Single Switch-Case controlled by “Counter” variable

C-macros for mapping signal name (in test case) to Embedded Coder generated C-code

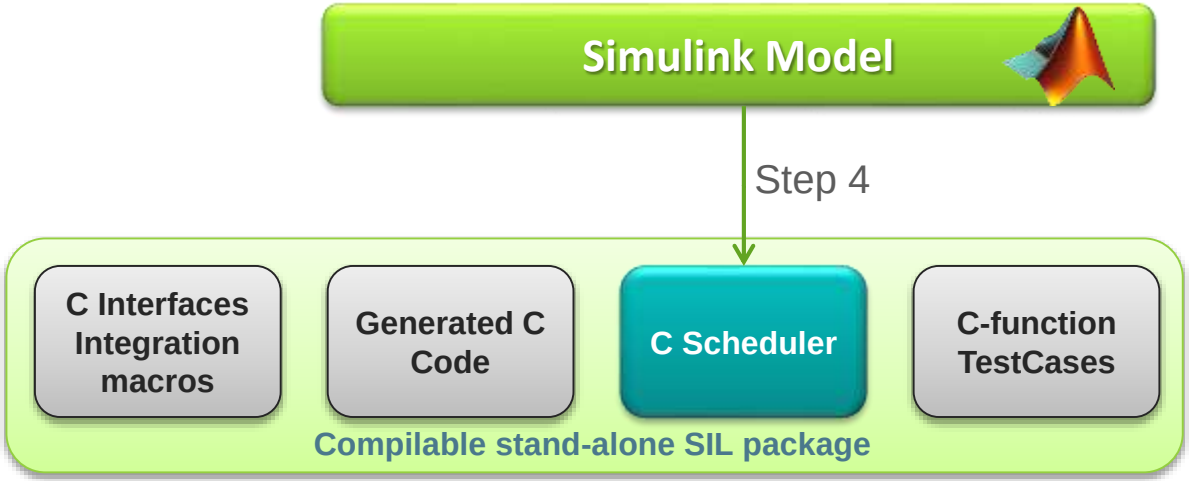
For each input, put the corresponding Signal value from the test case

C-macros for putting the corresponding Offset & Resolution for each input



Step 4: Preparing Scheduler

- Based on the Simulink model, we create a scheduler file, that:
  - Implements the timing counter
  - Update the stimuli before each task execution
  - Organizes the generic execution of test cases



Example: Customized Scheduler

"**Task_PERIOD**" which is extracted automatically from the model "sample time".

"**ms_counter**" is the main counter used for driving the absolute time for stimuli in each test case

"**TEST_END**" indicates the end of execution of the selected test case

Created "**SIL_Main**" to be called from the External Code Coverage Tool task
(Refer to step 6 for more info)

"**test_id**" identifies which test case to be executed

```
//...
int Task_PERIOD = 4.0;  /* Base rate in msec */
int ms_counter = 0;
int TEST_END = 0;
//...
void SIL_main(int test_id)
{
  if (FIRST_TIME == 1) {
    FIRST_TIME= 0;

    /* Initialize model */
    Subsystem_initialize();
    Init_Locals();
  }

  while (TEST_END == 0) {
    if (ms_counter% 1 == 0) {
      TEST_END = SIL_STIMULATE(test_id);
    }

    if (ms_counter% Task_PERIOD == 0) {
      rt_OneStep();
    }

    ms_counter++;
  }

  /* Terminate the model */
  Subsystem_terminate();
}
//...
```

Example: Customized Scheduler

Calling the created C-function "**SIL_STIMULATE**" for the respective "**test_id**" each 1ms.

4

Calling "**rt_OneStep**" each "**Task_PERIOD**"

5

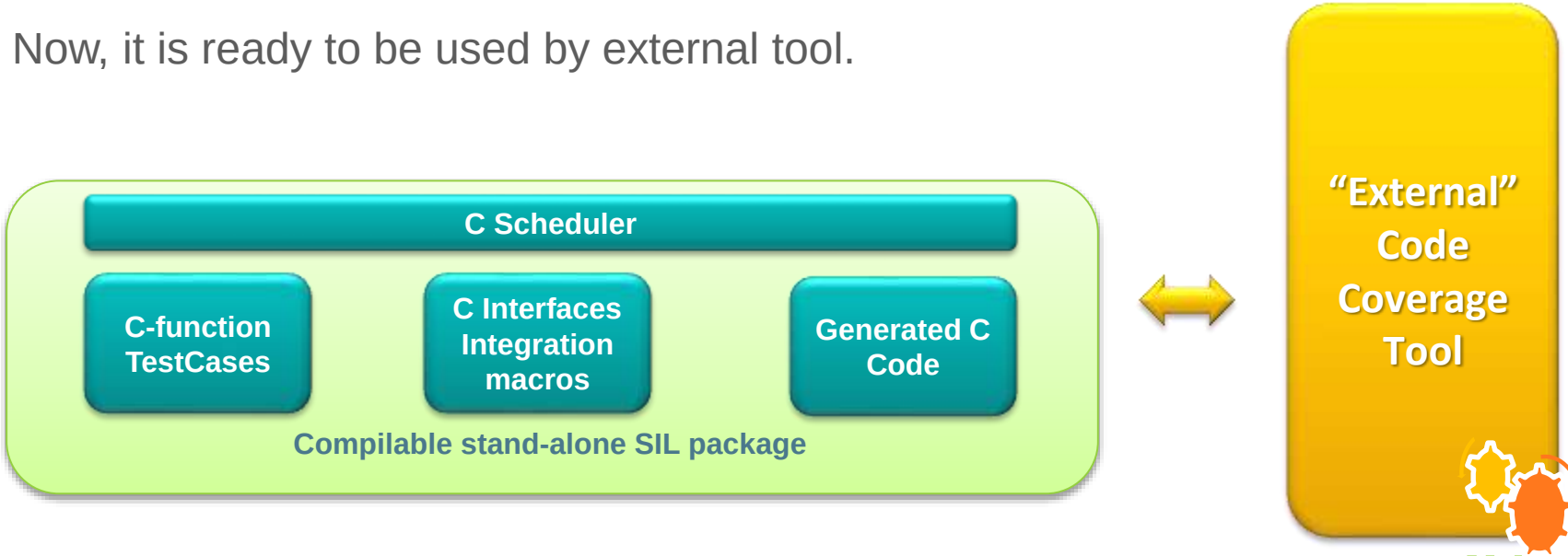
Incrementing the timing counter at the end of the task

6

```
//...
int Task_PERIOD = 4.0;  /* Base rate in msec */
int ms_counter = 0;
int TEST_END = 0;
//...
void SIL_main(int test_id)
{
  if (FIRST_TIME == 1) {
    FIRST_TIME= 0;

    /* Initialize model */
    Subsystem_initialize();
    Init_Locals();
  }

  while (TEST_END == 0) {
    if (ms_counter% 1 == 0) {
      TEST_END = SIL_STIMULATE(test_id);
    }

    if (ms_counter% Task_PERIOD == 0) {
      rt_OneStep();
    }

    ms_counter++;
  }

  /* Terminate the model */
  Subsystem_terminate();
}
//...
```

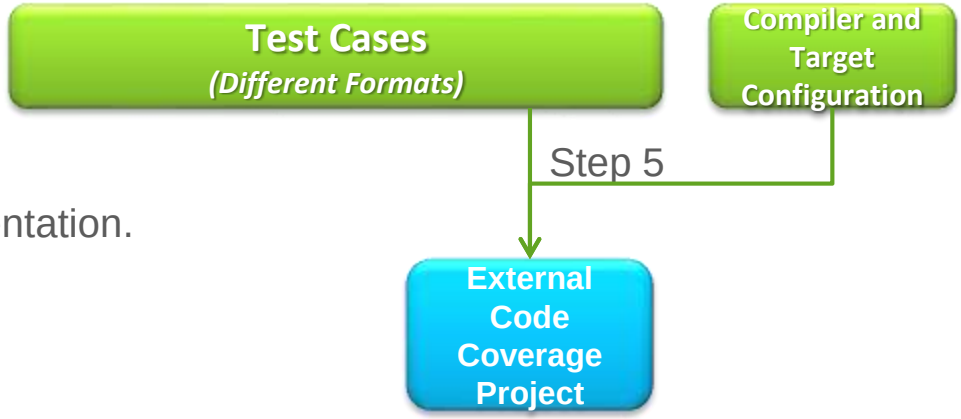
Objective Accomplished after step 4

- Now, we have a **compilable** software SIL package that is decomposed of both the Embedded Coder generated code and the SIL test cases. It is also **stand-alone** cause we have our scheduler file integrated.
- Now, it is ready to be used by external tool.



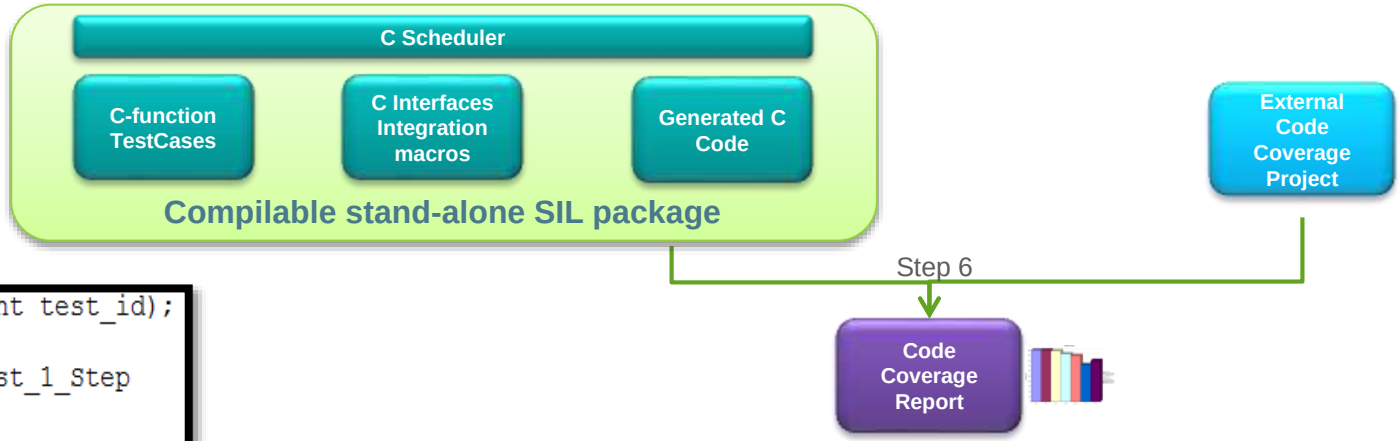
Step 5: External Code Coverage Project Creation

- Normally, any Code-Coverage tool requires a target file in order to proceed. This file specifies different options for compiler/linker and for the target.
- This required target file format highly depends on the external code coverage tool.
- For our workflow, we should have this file prepared before proceeding.
  - This step is out of scope of this presentation.



Step 6: Code-Coverage Execution

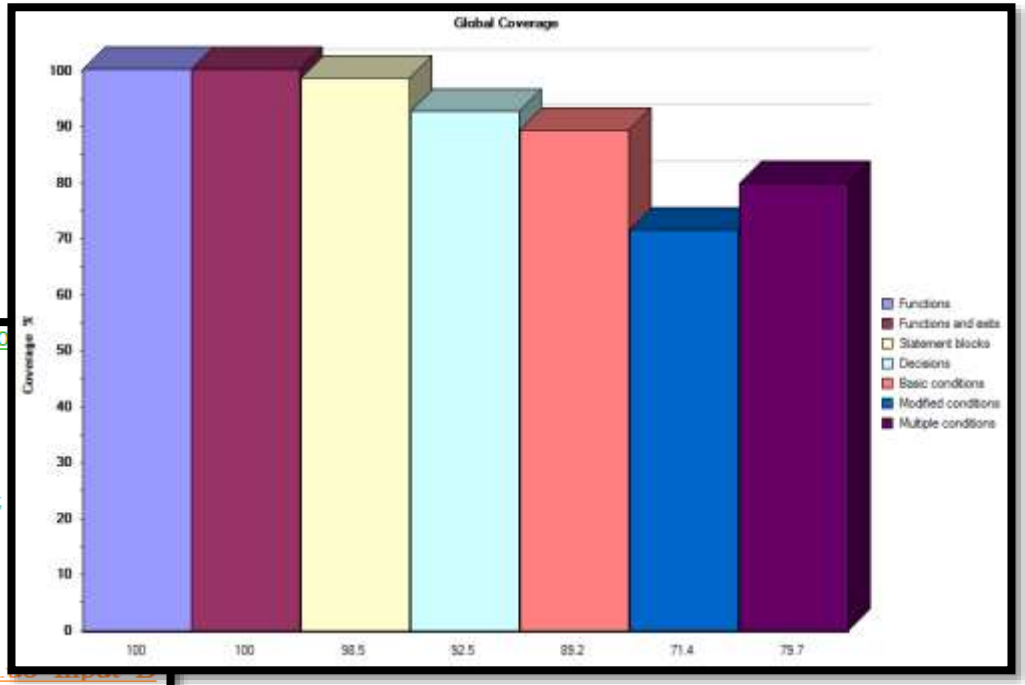
- Now, last step is to execute this SIL package on any of the Code-Coverage tools available at the organization and get the code coverage report.




```
#extern void SIL_main(int test_id);

 -- Test case: simin_Test_1_Step
 TEST  1
 FAMILY Nominal
     ELEMENT
     #SIL_main( 1 );
     END ELEMENT
 END TEST -- TEST  1
```

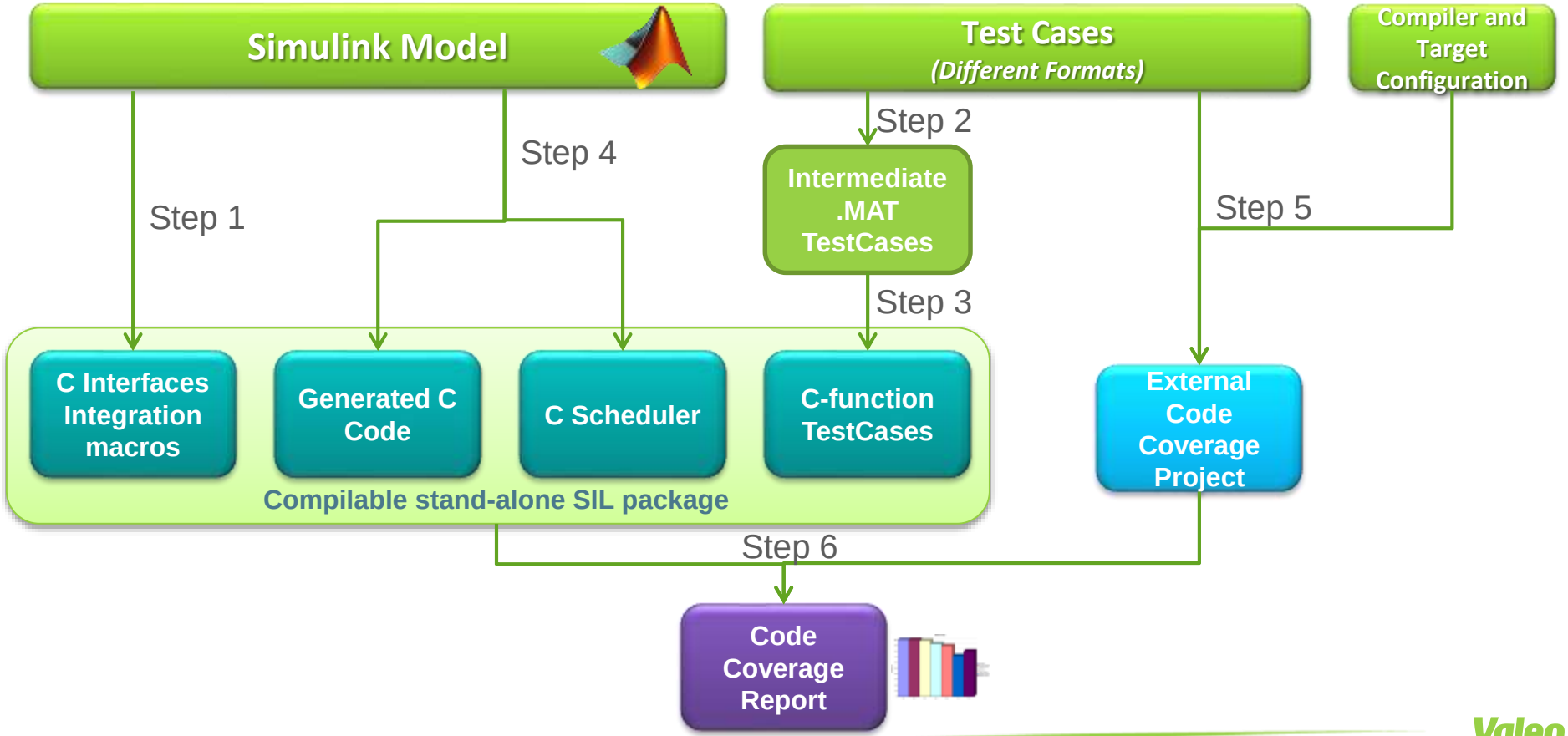
Example: Code Coverage Report

- This workflow is applied with one of our external tools that provide code coverage, and the code coverage report is shown.



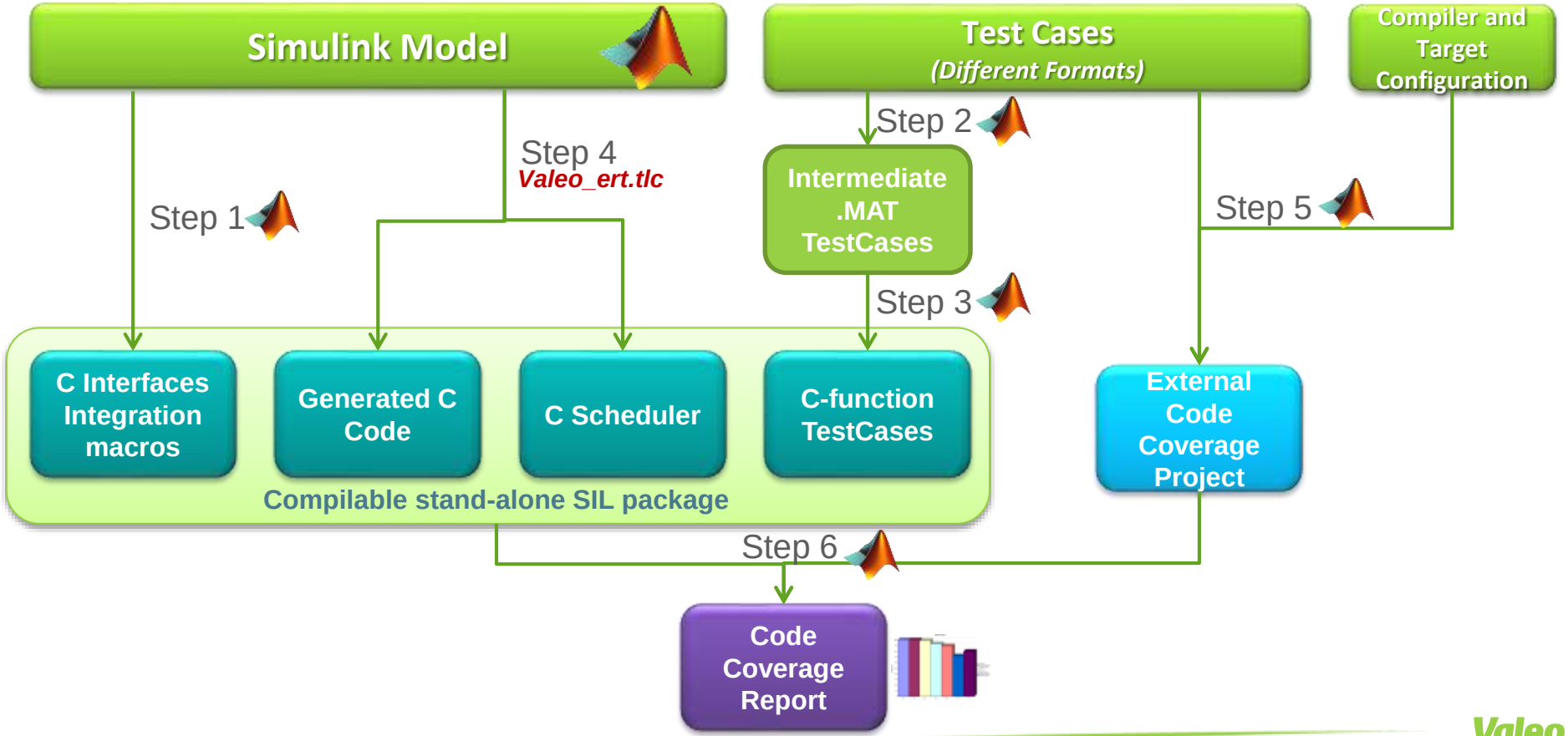

```
if (Minimum_Chart_DW.is_active_c3_Minimum_Chart == 0
basic boolean conditions:
 True
 False
) {
  /* Entry: Minimum_Chart/Minimum_Chart */
  Minimum_Chart_DW.is_active_c3_Minimum_Chart = 1U;

  /* Entry Internal: Minimum_Chart/Minimum_Chart */
  /* Transition: '<S2>:22' */
 } else {
  /* During 'Minimum': '<S2>:1' */
  /* Transition: '<S2>:21' */
  if (Minimum_Chart_U.u8_Input_A <= Minimum_Chart_U.
basic boolean conditions:
 True
 False
) {
    tmp = Minimum_Chart_U.u8_Input_A;
  } else {
    tmp = Minimum_Chart_U.u8_Input_B;
  }
```

Valeo Code Coverage Workflow: Final

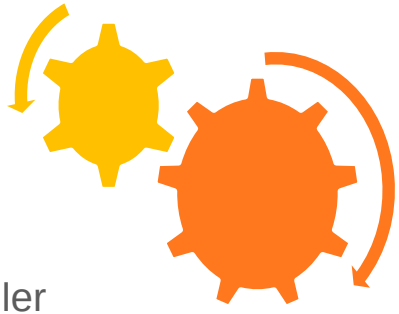


Agenda

Valeo Code Coverage Workflow: fully automated



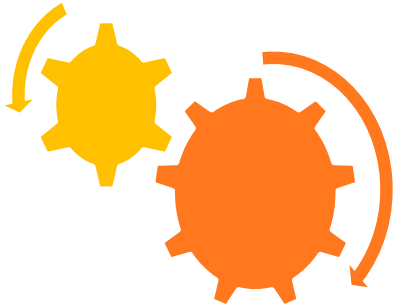
Automation



- In General, MATLAB script is used for the whole automation
- In order to automate the creation of the scheduler:
  - We used our customized “tlc” (based on “ert.tlc”) to generate the scheduler
  - While the code generated from the model is exactly the same as using “ert.tlc”
- Code-Coverage tool execution:
  - MATLAB script is used for automating the creation and customization of the project, such as:
    - Code-coverage level: Statement coverage,..., MC/DC.
    - Selecting the target file to execute with
  - Command lines are used for executing the external code coverage tool project

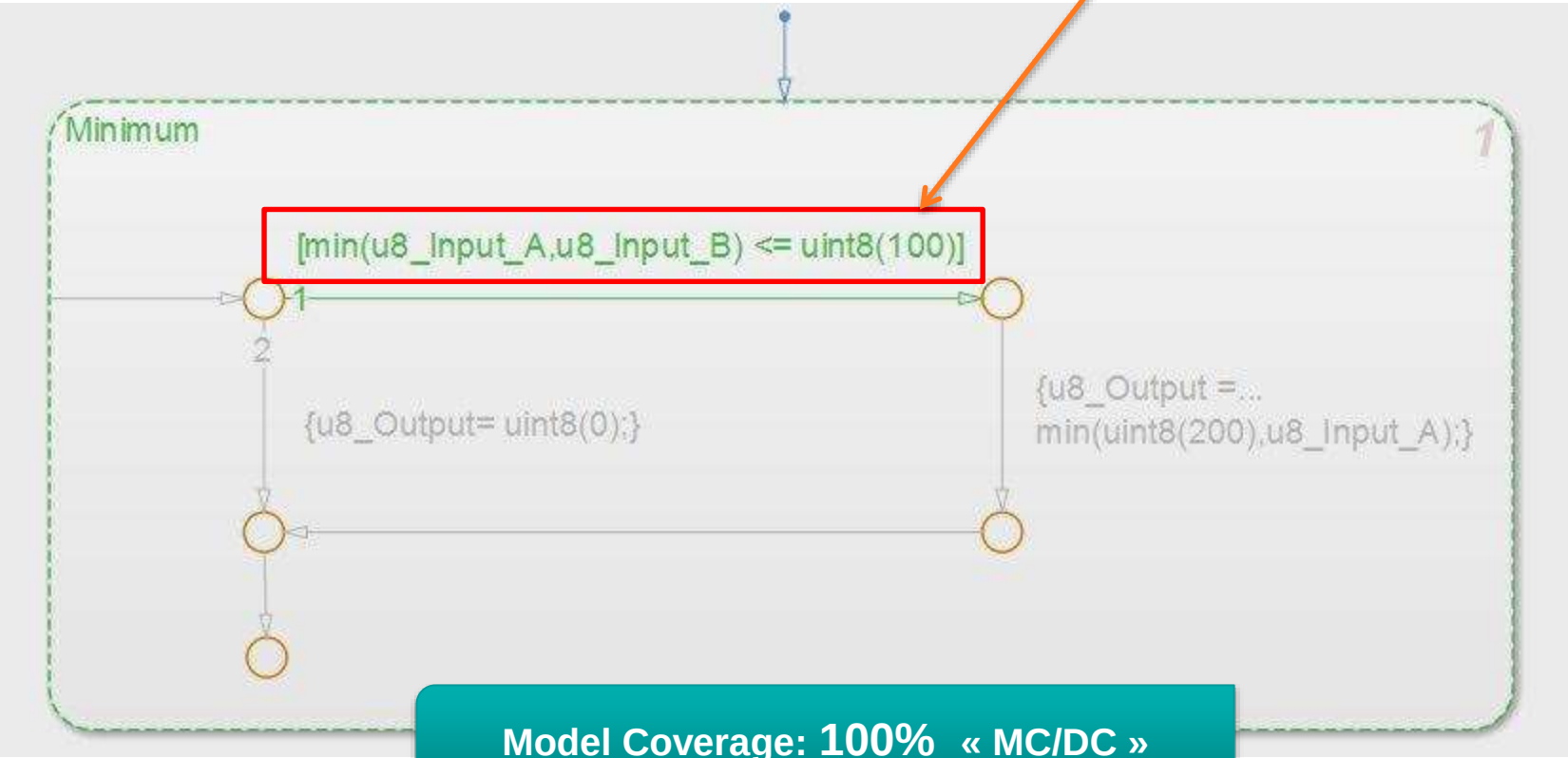
Workflow Deployment

- Workflow is already integrated into our Valeo MBD test-suite
- This workflow is already applied in some of our production projects, mainly safety critical ones.
  - To be generalized on all projects.



Example #1

[min (u8_Input_A, u8_Input_B) <= uint8(100)]

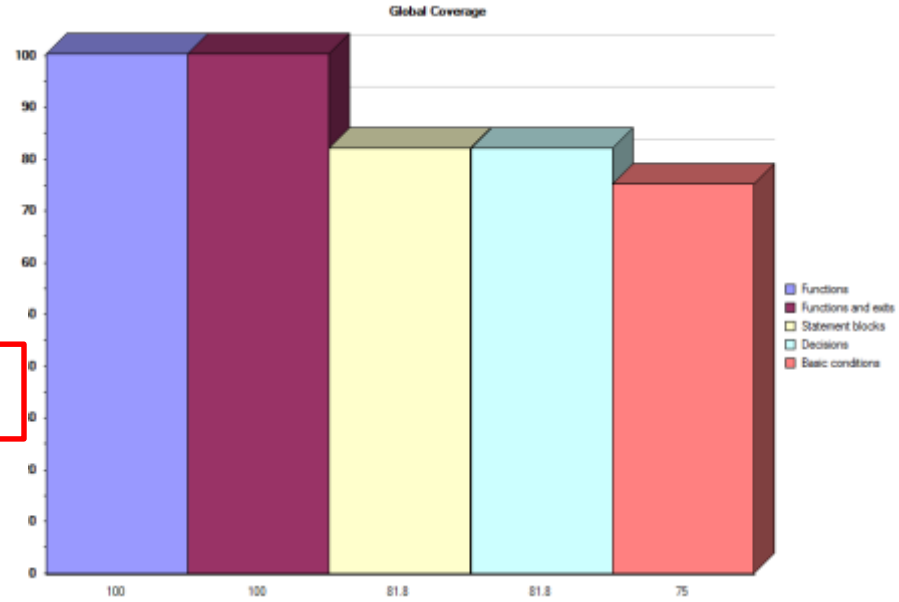


Model Coverage: 100% « MC/DC »

Example #1 (cont.)

- Missing test case detected at code level.

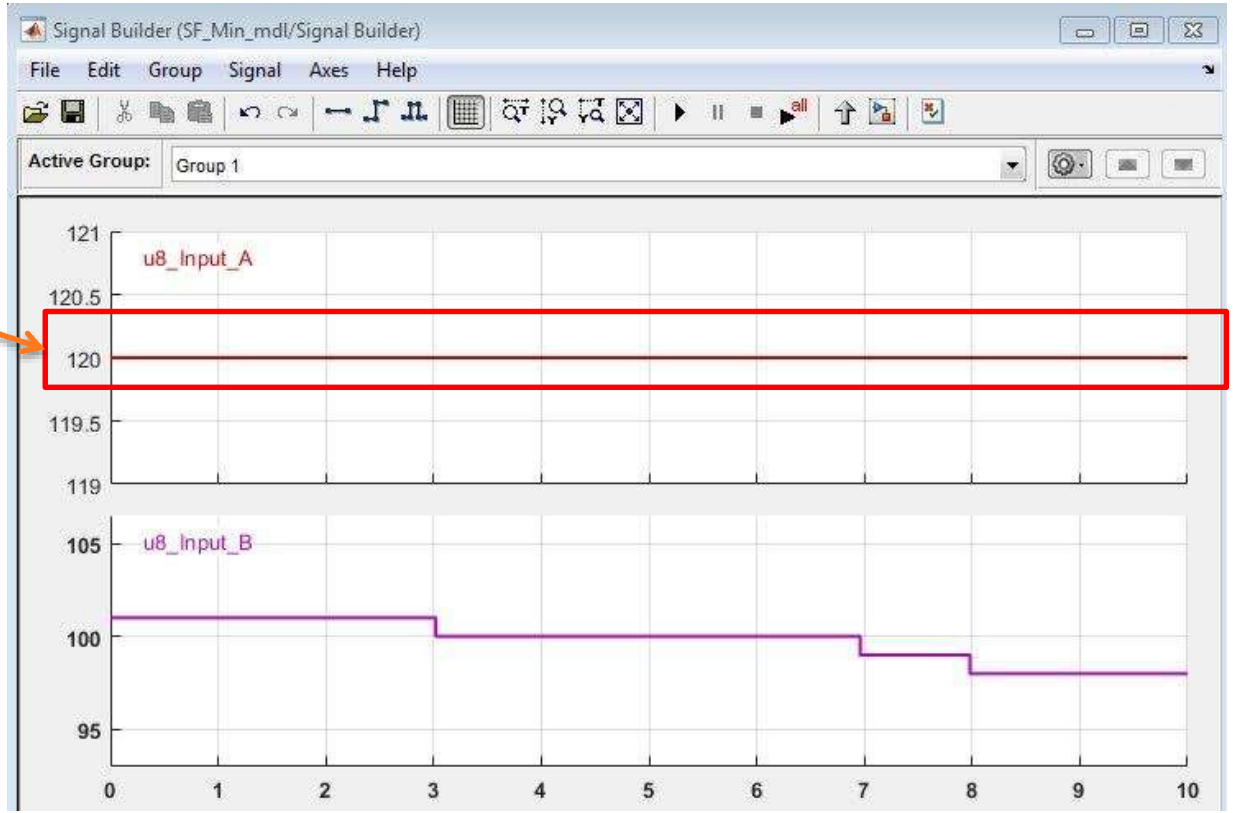
```
/* During 'Minimum': '<S2>:1' */
/* Transition: '<S2>:21' */
if (Minimum_Chart_U.u8_Input_A <= Minimum_Chart_U.u8_Input_B
basic boolean conditions:
True
False
) {
  tmp = Minimum_Chart_U.u8_Input_A;
} else {
  tmp = Minimum_Chart_U.u8_Input_B;
}
```



Example #1 (cont.)

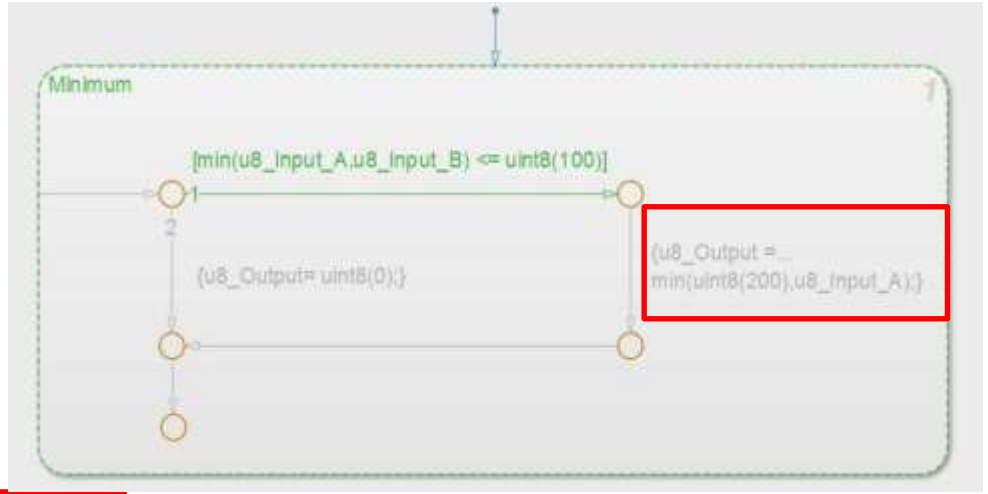
- Checking Signal Builder for the Inputs:

- **U8_Input_A,** never gets below **u8_Input_B**



Example #2

- For this specific case, actions were not executed against Model Coverage criteria, but it turned out to have missing coverage at code level.




```
/* Transition: '<S2>:12' */
/* Transition: '<S2>:11' */
if (((int32_T)200) <= ((int32_T)Minimum_Chart_U.u8_Input_A)
```

basic boolean conditions:
True
False

```
) {
    /* Outport: '<Root>/u8_Output' */
    Minimum_Chart_Y.u8_Output = 200U;
  } else {
    /* Outport: '<Root>/u8_Output' */
    Minimum_Chart_Y.u8_Output = Minimum_Chart_U.u8_Input_A;
  }
```

Agenda

Conclusion

- Using our proposed workflow, we managed to create a stand-alone compliable SIL package that can be used with any Code-coverage tool available at the organization.
- Accordingly, we were able to generate SIL Code-Coverage report for Embedded Coder auto generated code from the model under test. Which provided us with more confidence about our software under test.
  - We are mainly proposing this workflow for projects using MATLAB versions prior to R2016a
- We were also able to reuse existing code coverage tools at the organization.
- The whole workflow is automated using MATLAB script, and is already integrated into our Valeo MBD test-suite.

Valeo

Automotive technology, naturally