A Novel Approach to Functional Equivalence Testing

Ranjith Jayaram and Jetendra Kumar Borra
Mercedes-Benz Research and Development India, Bengaluru, India

Keywords: Model, Software, Functional Equivalence, Verification.

Abstract: In Model Based Software Development, sometimes it is required to transform the model and respective software code from one platform to another platform that is having different tool setup. For example, transforming a legacy model to the newly adapted architecture or transforming a model supplied from third party to production model and so on. Once the model is adapted to the new platform, there will be changes in the model, hence the code generated from the model can be different from that of legacy artifacts. After transformation activity, developers have to be sure that the new model and code is functionally equivalent to that of the old set. It is also important from quality standpoint that there are no deviations in the functionality after migration. With most of the compilation toolchains being closed source it is difficult to identify the issues during migration unlike in systems engineering. Achieving functional equivalence between the production artifacts and the reference/legacy artifacts provide conformity to the engineer of successful migration. In this paper, a methodology is proposed in which even with non-availability of few artifacts from legacy setup, functional equivalence is achieved using Model in loop and software in loop simulation results matchup. The method and the results are presented in the paper with two different use cases.

1 INTRODUCTION

The automotive industry is evolving very fast and automotive engineering is becoming more software driven. In software development, it is very common that different components of software are developed across different teams, different suppliers and integrated into production code after rigorous testing. Different strategies and tools are being followed in different organizations while planning and carrying out the software development. There are cases where the documentation for the legacy artifacts such as requirements, model, code are not available. One such case is legacy models that are being improved from time to time, grown big and not having enough documentation for requirements. When there is no stringent process for the software development, it is common that sometimes documentation is not available for the old models. In cases where proper documentation is not available, the engineers working on the respective functionality are the only source with knowledge of the requirements. Another case is, third party unit develop the model with the requirements given in descriptive format or model format, generate model based code and tested. After testing is complete, the code is flashed in Electronic Cntrol Unit (ECU) and only the ECU will be delivered to the product owner from the supplier. In all the above cases, when the software team plan to transform the models and software to new platform, with at least one of the information resources like requirements, code and test specifications missing, obtaining functional equivalence after migration guarantees the engineer that there is no deviation during migration from one platform to another platform. Functional equivalence in this context means the legacy code and the transformed code both should work in the exact manner as a product without deviation in the functionality.

2 PROBLEM

Two of the cases mentioned earlier are considered in this paper to explain our methodology and the application of the methodology is not limited to these but can be used for multiple use cases. First case considered in this paper is to adapt the legacy floating point model to TargetLink based fixed point model. The floating point model is not accompanied with code, unit test specifications and the tool used for generating code from floating point model is

Jayaram, R. and Borra, J.
A Novel Approach to Functional Equivalence Testing.
DOI: 10.5220/0011269900003274
In *Proceedings of the 12th International Conference on Simulation and Modeling Methodologies, Technologies and Applications (SIMULTECH 2022)*, pages 244-251
ISBN: 978-989-758-578-4; ISSN: 2184-2841

unknown, which make the transformation and obtaining functional equivalence difficult. Since the idea is to transform the model from one set up to another, it is important to make sure that there is no deviation in the functionality after migration. Before going forward, to avoid confusion and for easier explanation here onwards consider the legacy floating point model as reference model and the model developed with latest architecture as production model. In Model Based Development (MBD), across projects different tools are used for validating and verifying the code generated. Let us explore how different verification methods were fared in the current case. Model compare tools can be used to compare the models to identify the differences. Both the production and reference models are different in nature of development i.e, floating point and fixed point respectively. Since in production model fixed scaling is given to each variable and parameter in the model, there will be differences in the properties of each model block hence making it difficult to confirm from model comparison that code works as expected. As the model grows bigger, the model comparison activity becomes more laborious and less efficient. Due to the reasons mentioned above, model comparison do not fare well in ensuring the functional equivalence.

In order to do code comparison between both production and reference model based codes, there is no code available with the reference model artifacts. Automated code can be generated for reference model using a relevant compiler and it can be compared with the code of production model. The code from production model is generated in TargetLink platform. When auto generated reference model code is compared with the implicit automatically optimized, typecasted TargetLink based code (production code), the number of differences found is high due to inclusion of scaling and property differences. It is a laborious task for the engineers to identify the deviations among the differences. Even with the sincere efforts of the engineers there are high chances of inefficiencies, and makes the whole idea resource heavy, time intensive and inefficient. If test specification is available with the reference model, same test matrix can be used on both the production and reference artifacts for software in loop unit testing and result matchup. With some extra test cases for the fixed point data range, resolution testing and comparing the results functional equivalence can be achieved. Since the test matrix and reference model code are not available, software in loop testing and result matchup is not possible. Vehicle testing is costly and the test engineer has to drive, test for all the functionalities which is very costly and inefficient. Some of the software level bugs related to scaling and resolution of data variables can go without being identified in vehicle testing. Direct vehicle testing also contradicts with the whole idea of making the software vehicle ready with minimum issues and pre-validated sufficiently for vehicle testing. One interesting idea would be to prepare the test specification for the reference model and the production code can be validated by the same test specification. Since the requirements are embedded in the form of reference model, writing test cases using reference model sounds like a possible solution. But even with this solution how to completely validate the production code that has calculations with fixed point scaling with less efforts of resource is unanswered. The results on the production code has to be analysed for resolving the failed cases, which is time intensive for engineers. Since there is no reference code, it is not possible to do software in loop simulation on reference artifacts and there are no reference results to compare with the production code testing results. As driver drives the car with the code but not with the model it is very important for the engineer to completely validate the production code, hence this idea can be ignored.

The second case considered in the paper is a set of reference model which is Non-AUTOSAR TargetLink fixed point scaling environment model, code and test matrix for unit testing. Production model is AUTOSAR platform based model transformed from reference model and code. Starting with similar analysis like the one in first case, model comparison will not work, as there will be a change in the model architecture. Due to difference in platform setup, it is difficult to find the differences in the code as there will be restructuring in the code. Since the test matrix is available for the reference model the same can be used on the production code, by comparing the results, we can make sure that software is equivalent and hence functional equivalence can be achieved. However, for unit testing done with various coverage metrics, for example like C1 coverage there is a chance that some branches in code could be missed in testing. The functional equivalence process should be developed such that if there is any minor issue underlying, it should not go unreported without being registered with the engineer. In both the cases, even though the context and settings are different the target is to achieve the functional equivalence of the production code with that of reference. In search of solution for this problem we researched the literature and our findings from our literature survey are given in the next section.

3 LITERATURE AND MOTIVATION

In order to integrate the legacy systems into the system, there are three methods mentioned in literature (C. Chiang, 2007) wrapping, rewriting and reengineering. Considering the cost for a short duration project or project of small scale perspective wrapping process will suffice but with a compromise on quality of legacy code. For large scale projects with use case for longer period of time, reengineering option will work as good solution but consumes lot of resources. In case of converting the legacy models to a different platform rewriting helps in adopting the characteristics of the new platform faster. While integrating the legacy with new setup, or rewriting the setup, stability of the software plays a significant role. As mentioned in (P. Atcherley, 1994), testing the replacement for a legacy system is not well explored in literature. Once the legacy system is adopted to the new desired platform, functional test cases prepared from understanding of the requirements will give a good understanding of what is the region that is being covered and what is not. Adding new test cases can cover the uncovered regions. In (P. Atcherley, 1994), it was pointed that there must exist a platform to test the cases on the legacy code for better results to run both the legacy code and reengineered code on the same platform to assure that the migration is defect free. With reference to the experience report (Antonini, Canfora, Cimitile, 1994), it points that adequate setup to test the re-engineered work with original work is marginally tackled. In (Hocking, Knight, Aerllo and Shiraishi, 2014), one of the common problems and similar to the current problem is taken i.e, a 32 bit development model is tested for equivalence with its corresponding 16 bit production model using the novel concept constrained equivalence. The concept of constrained equivalence is analogical to the concept of a transfer function. Given the valid inputs for the first model, second model should have same results as first model, obviously with in the acceptable tolerance limits. The support tool that is being used in the work is Prototype Verification system (PVS) along with Simulink to simulate the models. Using constrained equivalence concept model equivalence is achieved but no details are mentioned regarding total system validation i.e. model and code together. In the current problem of interest, production model, which is developed in TargetLink platform, is used for generating optimized code and reference is floating point Simulink model. Even though the method mentioned is not appropriate for the current problem, the idea is absorbed for structuring a possible solution. For regression testing, Back to Back testing is an interesting testing concept coined decades back (MA Vouk, 1990). It is a concept that can be adapted for wide range of cases. Using this concept with the existing test matrix, we can test the new code and compare with the previous version to find out the latest changes in the code compared to its previous version. This concept has to be adapted as per the availability of the resources for the case. From the literature it is evident that there is no clear existing method to achieve functional equivalence.

With this knowledge from literature and the motivation to resolve the current problem of interest with a minimum effort solution, a novel methodology have to be developed. The solution should be cost-efficient such that there is no need for the newly developed software to be tested rigorously in expensive Vehicle/HIL (Hardware-in-the-Loop) testing to achieve functional equivalence. Instead, after passing functional equivalence test with the proposed method, couple of regression tests on Vehicle/HIL setup should prove the credibility of the software.

4 BACKGROUND

For the first case as mentioned earlier (Section 2) MIL (on reference model) – MIL (on production model) comparison stands out as good option but the code validation is still an open question and answering this question closes the problem. Before going into the methodology, brief information on BTC Embedded tester, Back to Back testing and Wrapper is provided in the following subsections for ease of understanding of the method.

4.1 BTC Embedded Tester

There are multiple tools available with engineers for testing activities, we chose BTC EmbeddedTester. BTC is one of the tools which can offer back to back testing. BTC Embedded Tester is a tool provided by BTC Embedded systems AG. It provides an ISO 26262 certified and fully automated back-to-back test between model and code. It can execute the same test cases on multiple levels i.e., MIL (Model-in-the-loop), SIL (Software-in-the-loop) and PIL

(Processor-in-the-loop). Using this tool, engineers' set of test cases for functional check can be uploaded and evaluated. If the test cases are not giving full coverage, there is a facility with the tool to automatically generate the test cases for uncovered parts of the code and missed unique test combinations. With one click back to back testing option and automated test matrix generation feature, this tool is handy for the engineers.

4.2 Back to Back Testing

This is a popular and resource efficient testing methodology. In this method, same set of test cases are used for testing different variants of the software or for testing functionally equivalent components. In multiversion experiments, at first individual testing of the software is completed and then the results are compared with its previous version, we call it as back to back testing. Based on the reliability target to be achieved back to back testing strategy is modified from case to case. The limitation of this testing is in multiversion software cycles if the issue is missed in MIL test, it will be missed in SIL testing as well. So, issues in particular branch of code will go unnoticed, i.e. if a functional aspect test is missed in MIL testing, it will be missed in SIL testing as well, because same test matrix is used for MIL-SIL testing and result comparison. To overcome this limitation, in our method an extra step is added for maximum code coverage. Back to back testing concept is given in Figure 1. The use case of Back to Back testing for the current problem of interest is comparison of results from SIL, MIL of the same model and then comparing xIL-xIL results of reference and production model. More information on Back to Back testing can be referred from (MA Vouk, 1990).

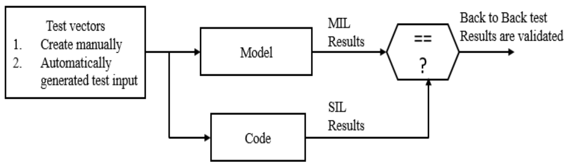

Test vectors
1. Create manually
2. Automatically generated test input
Model
MIL Results
==
?
Back to Back test Results are validated
Code
SIL Results

Figure 1: Back to Back testing.

4.3 Wrapper

A wrapper is a function or script which helps in better abstraction of data/signals of the core function by means of signal routing. The core function and the wrapper can be modified independently. In scenarios

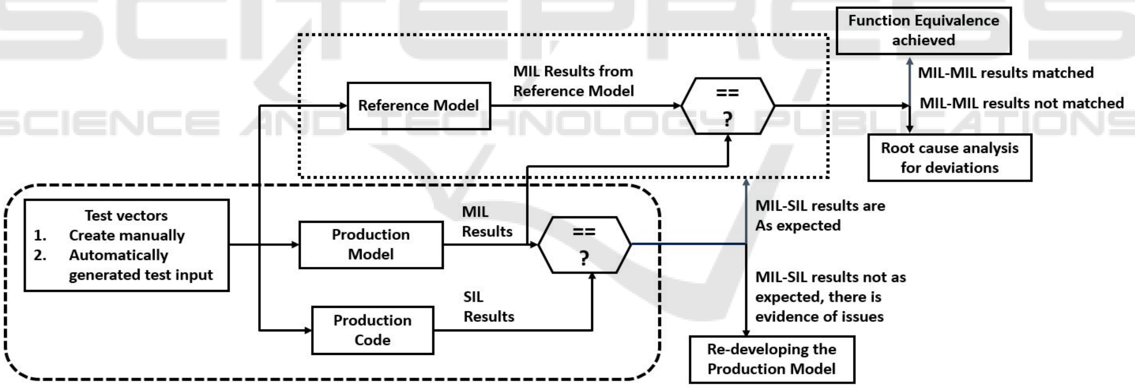

Function Equivalence achieved
MIL Results from Reference Model
Reference Model
==
?
MIL-MIL results matched
MIL-MIL results not matched
Root cause analysis for deviations
Test vectors
1. Create manually
2. Automatically generated test input
Production Model
MIL Results
==
?
MIL-SIL results are As expected
MIL-SIL results not as expected, there is evidence of issues
Production Code
SIL Results
Re-developing the Production Model

Figure 2: Functional Equivalence methodology for Case 1.

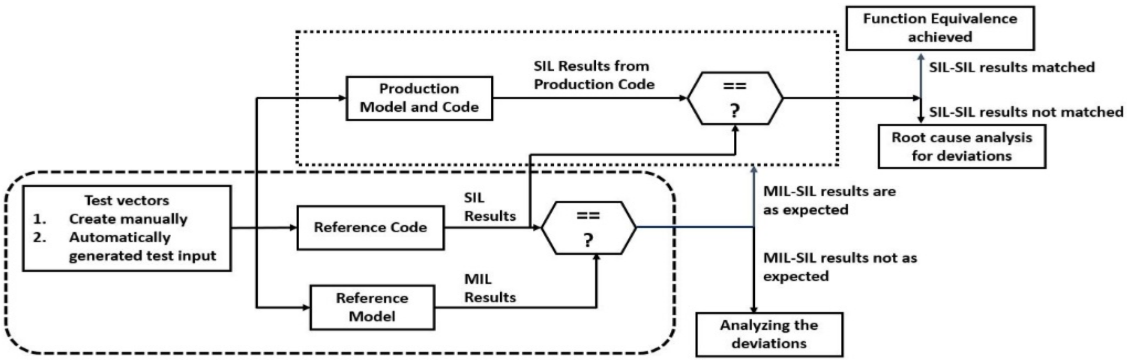

Function Equivalence achieved
SIL Results from Production Code
Production Model and Code
==
?
SIL-SIL results matched
SIL-SIL results not matched
Root cause analysis for deviations
Test vectors
1. Create manually
2. Automatically generated test input
Reference Code
SIL Results
==
?
MIL-SIL results are as expected
MIL-SIL results not as expected
Reference Model
MIL Results
Analyzing the deviations

Figure 3: Functional Equivalence methodology for Case 2.

when the interface change*s* or signal naming replacement*s* are needed, it can be achieved with wrapper without modifying the function logic.

5 APPROACH

The approach is similar for both the cases but not the same. The core idea of the approach is one set of input matrix, the one that tests the functionality, coverage and data range of signals in the model, is fed to the available reference model package and to the available production model package. We expect the output values from both the sets to be matched. The first case is reference model which is floating point model but code, test specifications are not available with the resource package. Production model is fixed-point TargetLink based model and corresponding code is TargetLink auto generated. The second case discussed here is, the reference model is TargetLink based fixed point scaled model with Code, test matix available in the resource package and the production model is, same model migrated to AUTOSAR platform with new tool chain. The approach for both the sections is explained in the subsequent sections.

5.1 Case 1

For the production model, test matrix is developed or completely auto-generated from BTC Embedded Tester and MIL-SIL results are compared. If there are no major deviations in MIL-SIL result comparison then we can proceed to next step. If there are deviations in MIL-SIL result comparison they should be addressed. Most of times by keeping one LSB tolerance setting to the result analysis, issues of considerable interest will be reported and minor deviations due to one LSB, which are very common can be avoided. With the production code tested test matrix, reference model is tested and results are compared. If the results are matching with justifiable deviations, production model is passed, else root cause analysis will be done for the failure as it points a potential bug/deviation. The process is shown in Figure 2. With the new methodology proposed instead of going in forward direction that is comparing the reference model results with production model results, the functional equivalence is done in reverse manner. First the production model and code is tested back to back with MIL, SIL with the proven test specification and MIL results, reference model is tested and MIL results are compared there by achieving functional equivalence. Explaining the method in easy steps below:

(i) Generate test matrix for production code first using reference model as requirement source.
(ii) Generate extra test cases for coverage and unique test combinations automatically from BTC.
(iii) Generate MIL-SIL results for production code.
(iv) If no deviations in MIL-SIL results, use the same test matrix from step (i) and step (ii) for MIL testing on reference model.
(v) Compare MIL results of reference model with MIL results of production model.
(vi) Justify and resolve the deviations found if any.

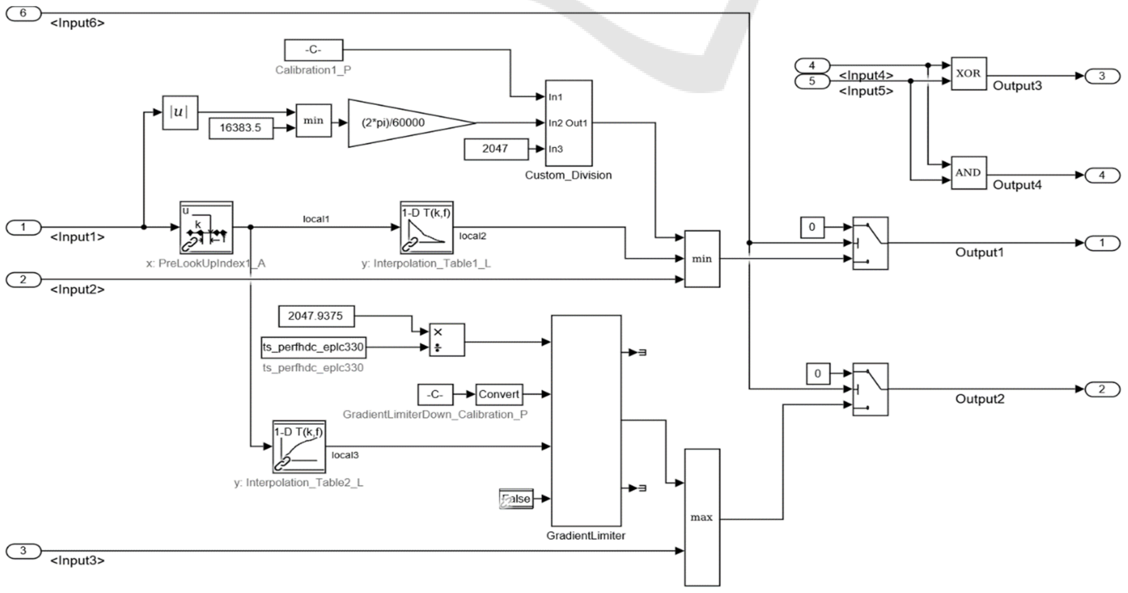
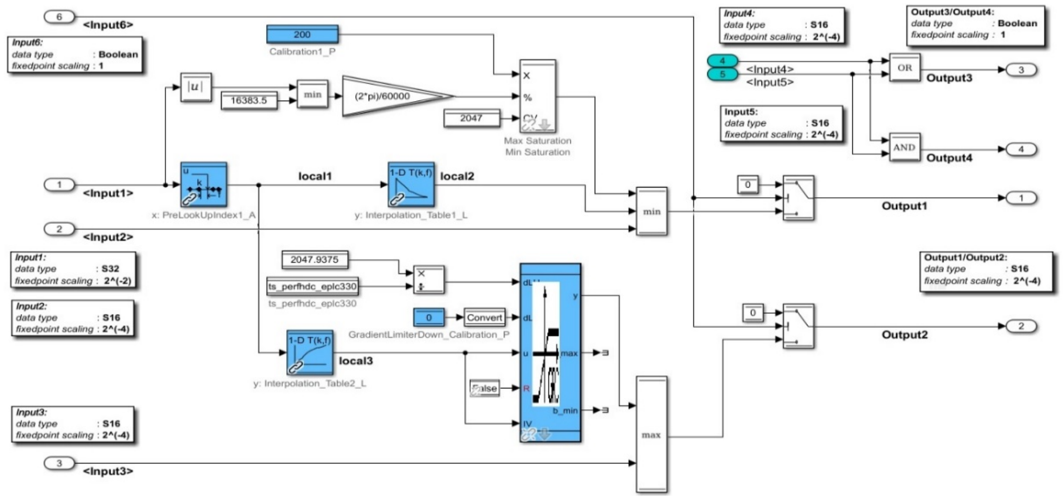
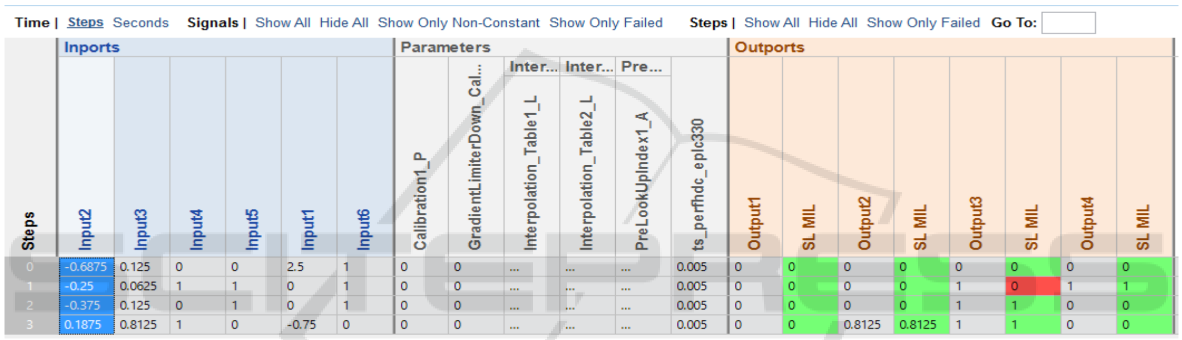


Figure 4: Floating point Model – Reference Model Case 1.



<Input6>
Input6: data type : Boolean fixedpoint scaling : 1
200
Calibration1_P
|u|
16383.5
min
(2*pi)/60000
2047
Max Saturation
Min Saturation
Input4: data type : S16 fixedpoint scaling : 2^(-4)
Output3/Output4: data type : Boolean fixedpoint scaling : 1
<Input4>
<Input5>
OR
Output3
Input5: data type : S16 fixedpoint scaling : 2^(-4)
AND
Output4
<Input1>
local1
1-D T(k,f)
local2
x: PreLookUpIndex1_A
y: Interpolation_Table1_L
min
Output1
<Input2>
Input1: data type : S32 fixedpoint scaling : 2^(-2)
2047.9375
ts_perfhdc_eplc330
Convert
GradientLimiterDown_Calibration_P
Output1/Output2: data type : S16 fixedpoint scaling : 2^(-4)
Input2: data type : S16 fixedpoint scaling : 2^(-4)
1-D T(k,f)
local3
y: Interpolation_Table2_L
False
max
b_min
Output2
Input3: data type : S16 fixedpoint scaling : 2^(-4)
max
<Input3>

Figure 5: Fixed point Model - Production Model Case 1.


Time | Steps Seconds Signals | Show All Hide All Show Only Non-Constant Show Only Failed Steps | Show All Hide All Show Only Failed Go To:

	Inports						Parameters		Inter...	Inter...	Pre...		Outports							
Steps	Input2	Input3	Input4	Input5	Input1	Input6	Calibration1_P	GradientLimiterDown_Cal...	Interpolation_Table1_L	Interpolation_Table2_L	PreLookUpIndex1_A	ts_perfhdc_eplc330	Output1	SL MIL	Output2	SL MIL	Output3	SL MIL	Output4	SL MIL
0	-0.6875	0.125	0	0	2.5	1	0	0	...	...	...	0.005	0	0	0	0	0	0	0	0
1	-0.25	0.0625	1	1	0	1	0	0	...	...	...	0.005	0	0	0	0	1	0	1	1
2	-0.375	0.125	0	1	0	1	0	0	...	...	...	0.005	0	0	0	0	1	1	0	0
3	0.1875	0.8125	1	0	-0.75	0	0	0	...	...	...	0.005	0	0	0.8125	0.8125	1	1	0	0

Figure 6: MIL – MIL result for floating to fixed-point model adoption.

5.2 Case 2

In this case the platform to which the production model is transformed is different to that of reference model. During migration from one platform to another there are chances that signal names might be interchanged or properties will be changed by mistake which might create a deviation in the functionality. If the unit test matrix is available for the reference model, it is used. In cases where function test matrix is not available, test inputs can be automatically generated from the unit testing tools. As code of the reference model is available, SIL-SIL result comparison will suffice for the functional equivalence. In this case first the reference model is taken and with the existing test matrix if the functional test cases are not reaching 100% coverage, extra test cases can be automatically generated. With the enhanced test matrix, on the reference model the MIL-SIL simulations are done and the results are compared. Since the reference code is already tested one, whenever it deemed not necessary SIL-MIL back-to-back test can be skipped. Wherever there is a deviation it will be analysed and justified. The enhanced test matrix is given as test input to the AUTOSAR based production model and the SIL results are compared. If there is an unjustifiable deviation, root cause analysis has to be done for finding the issue. The process is given in Figure 3. With this approach, if there is any issue/deviation that was not found earlier, it would not go unnoticed by the engineer during this process. In simple steps:

(i) Use the existing test matrix and generate extra test cases from BTC tool based on the need or generate full test matrix from tool.
(ii) Use the test matrix on reference model and generate SIL results.
(iii) Use the same test matrix in step (ii) on production model and generate MIL-SIL
(iv) Do MIL-SIL analysis and look for any deviations.
(v) Compare the SIL results of reference model and SIL results of production model, if found any deviations do root cause analysis

6 RESULTS AND DISCUSSION

For analysis of the above mentioned method, a demo model is taken. The floating point model (the reference model) is given in Figure 4. The model also represents the functional requirements. As mentioned in Section 5.1, the reference model is transformed to fixed point model given in Figure 5, which is production model. A deviation is introduced in the production model intentionally at input4 and input5, OR gate is used in place of XOR gate. This kind of deviation can happen during migrating or during fixing the scaling for the calculation of signal or optimizing the model. In order to replicate similar issues a deviation is purposefully introduced in the model. As mentioned in the process, first functional test cases for the production model are written, enhanced the test matrix with the automatic unique combination test cases from the testing tool and production model is tested. With the same test matrix, reference model is tested and as a purposeful issue is introduced in the model, results show that there is a deviation at Output3 signal. At the MIL-SIL back to back testing of the production model, the deviation at Output3 will not appear because the test cases are written for production model. As same logic given in model is replicated in the code, MIL-SIL results pass the back-to-back test. Now the engineer can process back to model development, the deviation can be corrected. Once the deviation is corrected, the results will be as expected. Sometimes the deviations can be due to one LSB deviation of the signals, as per the engineers decision and the product requirements those can be justified and can be moved to next step in the software development process. Given the context, one point to note is, in most of the software development processes unit testing is considered as a necessary artifact before delivering the code for software integration and code coverage percentage is the quality metric. In coverage testing it is possible that sometimes unique test combinations will be missed while writing the test matrix. One such case is explained in the below in Table 1. From the first three test cases in the table, it covers 100% coverage, when the same test matrix is used on the production model, the test passes inspite of having a deviation between reference model and production model. Unique test cases from the testing tool helps in adding extra test cases, which were missed even with 100% coverage. With the unique test combination, test case when tested on both the models, it will show the difference in results. If there is an issue which was hidden in the earlier process during the functional equivalence testing, these will pop-out. The MIL-MIL results for case 1 are given in Figure 6.

In order to explore case 2, an example model is taken which has undergone signal routing changes to provide better data abstraction of AUTOSAR model migration. The reference model is Non-AUTOSAR model as shown in Figure 7 and the production model is AUTOSAR model as shown in Figure 8. In Figure 7, the function given is even parity function, whose function is, if even number of inputs are true then output is true, else false. With migration to AUTOSAR architecture, new interface signals are created to replace old interfaces to be compatible with AUTOSAR architecture. Sometimes the properties, scaling of the signals might change during the new signal creation process or there arise scenarios where the signal routing can go wrong with migration when compared to the reference model. One more possible case of deviation is when the model is migrated, unnecessary or redundant signals and logics is removed, in such cases the unit test results of both the models will not match.

Table 1: Test cases for XOR logic.

Test scenario	Test No.	Input 4	Input 5	Reference model Output 3	Production model Output 3
Manual test cases	1	0	0	0	0
	2	0	1	1	1
	3	1	0	1	1
Unique test combination from BTC tool	4	1	1	1	0

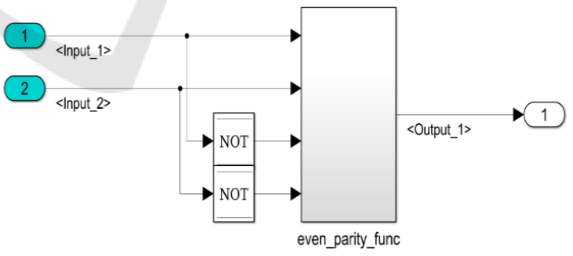


Figure 7: Reference Non-AUTOSAR model.

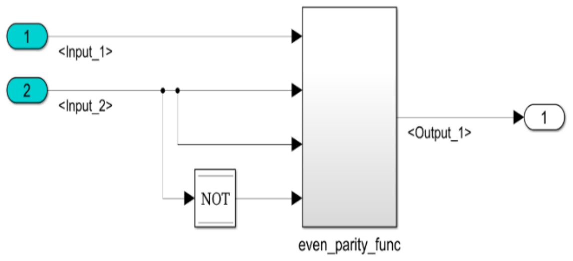
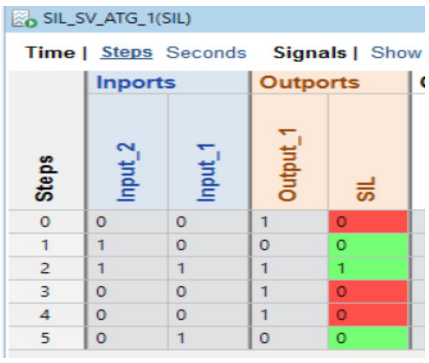


Figure 8: Production AUTOSAR model.

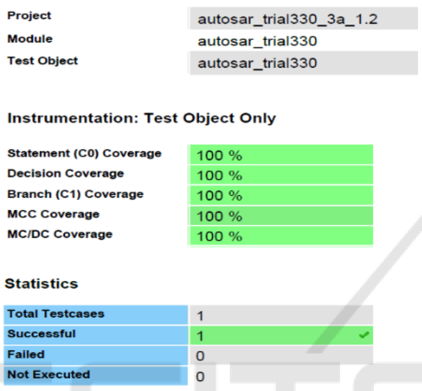


SIL_SV_ATG_1(SIL)

Time | Steps Seconds Signals | Show

	Inports		Outports	
Steps	Input_2	Input_1	Output_1	SIL
0	0	0	1	0 🟥
1	1	0	0	0 🟩
2	1	1	1	1 🟩
3	0	0	1	0 🟥
4	0	0	1	0 🟥
5	0	1	0	0 🟩

Figure 9: Functional Equivalence Test Results.



Project	autosar_trial330_3a_1.2
Module	autosar_trial330
Test Object	autosar_trial330

Instrumentation: Test Object Only

Statement (C0) Coverage	100 %
Decision Coverage	100 %
Branch (C1) Coverage	100 %
MCC Coverage	100 %
MC/DC Coverage	100 %

Statistics

Total Testcases	1
Successful	1 ✓
Failed	0
Not Executed	0

Figure 10: Conventional Test Results.

When tested this in the conventional method by designing the test scenarios i.e. test step 1, step 2 and step 5 shown in Figure 9 to cover the requirement and code coverage, we can notice that some of these deviations in functionality are not identified in the results despite achieving 100% code coverage as shown in Figure 10. Hence functional equivalence test plays a crucial role in validating the migration of model in an effective manner. Along with the existing test cases of the reference model, automatically generated unique test combinations from the testing tool are tested on reference and production models. Test scenarios in step 0,step 3 and step 4 in Figure 9 are the unique test combinations automatically generated from the testing tool. When the results are compared as there is a deviation introduced, fail cases appeared in the results. This will go to the notice of the engineer and necessary steps can be taken. In the cases where due to redundancy or optimization if some signals or any logics are removed, if the engineer analyses and confirms that the test failures are as expected due to modifications carried out, fail cases can be justified and software module can be moved to next level in the process.

7 CONCLUSION

In the literature, we did not find relevant methodology to solve the issues during migration activities. We initially started with an idea and evolved this functional equivalence process after trialing on different set of issues and software components. The sequence of steps given in the process is novel and carefully designed to identify issues, minimize the efforts. Future scope of this work to automating this function equivalence testing process and developing a Continuous Integration and Continuous Development (CI/CD) setup to make better use of resources.

ACKNOWLEDGEMENTS

We thank Kaushik Raghunath and Samiya Behera for supporting and guiding us in completing this idea.

REFERENCES

C. Chiang (2007). Software stability in software Reengineering, *IEEE International Conference on Information Reuse and Integration*.

P. Atcherley (1994). Reengineering legacy systems into new environments, *IEEE Colloquium on Reverse Engineering for Software Based Systems*.

Antonini, Canfora, Cimitile, (1994). Re-engineering legacy systems to meet quality requirements: an experience report, *Proceedings 1994 International Conference on Software Maintenance*

A.B.Hocking, J.Knight, M.A.Aiello and S.Shiraishi,. (2014). Proving Model Equivalence in Model Based Design, *IEEE International Symposium on Software reliability Engineering Workshops*.

MA Vouk (1990). Back-to-back Testing, *Information and Software Technology*

BTC Embedded Systems AG, *EmbeddedTester*, https://www.btc-es.de/en/products/btc-embeddedtester.