Implementation of ISO Check For Early Failure Detection in Vehicle Design Using FPGA

B. Thiyaneswaran[a], K. Anguraj[b], J. Jayanthi[c], M. Suba Priyadharsini[d], P. Srinivasan[e], N. S. Yoganathan[f]
[a,b,d,e,f]Department of ECE, Sona College of Technology, Salem, India
[c]Department of CSE, Sona College of Technology, Salem, India

Article History: Received: 10 November 2020; Revised 12 January 2021 Accepted: 27 January 2021; Published online: 5 April 2021

Abstract: To be more concerned with the fatal cause during the real time experiences in automotive industry in the ECU failures, it shall be a preventive measure to detect the failure of the ECU's at the probation period of modelling it. MATLAB, the tool used for designing the ECU's would be useful also to check this with the help of its inbuilt tool called MODEL ADVISOR. With the m-script, we can use to customize the rules for which there might be the reason for failure. Some of the model level failures are detected using this model checker tool. Before we dump the code into FPGA or an ECU, the code level failures can be detected via polyspace, which is also a tool by MATLAB. With this earlier detection of failures before dumping the code into ECU, this saves the lives of many people during the emergency situation when there is a possibility of failure of ECU. Also production loss of the component can be reduced along with this the standardized product can be released, so that it would be the safe for use.

Keywords: Simulink, ISO design rule, FPGA

1. Introduction

Whenever we go for the creation of new component for a vehicle, we go in for the specific design which would be user friendly, to have an overview about the description of the actual component we require. As per the confidentiality for the specific component designed by a product designer, it is required to encrypt the design. The conversion of a Simulink model to s-function suits good in this case.

Conversion of a MATLAB Simulink model to FPGA (Field Programmable Gate Array), and doing a test validation helps in the final sanity check before we program the code into the specific ECU. Also to be confident for the fatal cause of the component design, we have to ensure for the ISO compliance check, which can be done via the Model advisor which is present in MATLAB. The specific customized rules can also be integrated to make a compliance check.

This precautionary way helps the Component designer or the product based company to provide a standardized component\product, So that it would be better accreditation in terms of seller perspective. When it comes to the Consumer perspective, this brings in a satisfied feel of buying it with safety as the main concern. The deployment of the code into the ECU without testing, and getting into the stage of production and later on during the real time experiences, this model gets failed, the production and recycling it would be time consuming as well as the man effort loss along with the wastage of materials used for ECU.

2. Literature review

Saiful A. Zulkifli et al have proposed "Development of vehicle powertrain simulation module by interfacing Matlab-ADVISOR to LabVIEW"[5] The development of a powertrain simulation module which is capable of demonstrating the benefits and drawbacks of different drivetrains, based on ADVISOR, a vehicle simulation program on Matlab-Simulink platform. By interfacing the Matlab-ADVISOR program to LabVIEW, the final simulation module has a dynamic simulation interface, suitable for an intermediate level of powertrain studies.

Jiaing et al[8] have proposed, Performance Analysis of Electric Vehicle in Worldwide Harmonized Light Vehicles Test Procedure via Vehicle Simulation Models in ADVISOR [8] evaluate and verify the performance and the effect of the new driving cycle on an electric vehicle, set side by side to the major regulatory drive cycles, which include NEDC, FTP 75 and JC 08. Using Matlab's ADVISOR simulation, a simulated vehicle model of Nissan Leaf 2016 is created and place through the different driving cycles

Bassoli [3] have proposed, Automotive instrument cluster screen content validation [3],describes a novel solution to provide this kind of screenshots to the car manufacturers, called Image Grabber application, using a DSLR digital camera, a CAN case and a C# tool. The proposed solution has no original equipment manufacturer limitations, as it can be used on all types of clusters and provides an implementation cost decrease of up to 10 times, when compared with a similar hardware grabber system. Cheng et al [4] have proposed Design guideline of the EMB controller based on ISO26262 to analyse the failure modes which are required in ISO26262 Part 5, 6. And we propose design guidelines to meet safety requirements. In addition, the EMB system was implemented based on the proposed hardware and software guidelines [1]. The implemented EMB system meets Part5, 6 of ISO26262. Therefore, the proposed guidelines are made good use for product development to meet Part 5, 6 of ISO26262. In this paper, we have studied fault diagnosis function, and fail safe function should be needed to study.

Alfredo Imparato, Raffaele Rodolfo Maietta have proposed, A Comparative Study of Static Analysis Tools for AUTOSAR Automotive Software Components Development [5], following automotive safety standard ISO26262 to reduce software runtime errors. A comparative study of top performance tools is provided by analysing production code of AUTOSAR application software components for an Instrument Panel Cluster. A quantitative analysis based on an alert classification model and performance metrics has been carried out [6]. The goal of this paper is to identify the best combination of commercial static analysis tools to minimize defects of AUTOSAR software components in a context of automatic code generation. Finally, the results analysis has suggested some improvements of static analysis tools.

Imparato et al have proposed Formal Verification of Automotive Design in Compliance With ISO 26262 Design Verification Guidelines [7], to compare industrial design verification steps of WatchDog Manager in an effort to be ASIL B-compliant with a proposed no disruptive methodology to semi formally verify WatchDog Manager UML design via an automated formal framework backbone [9]. This semiformal verification framework will allow automotive software to comply with ASILs C and D formal and semiformal unit design and implementation verification recommended guidelines in ISO 26262. Semiformal UML finite-state machines are automatically compiled into formal notations based on the Symbolic Analysis Laboratory formal notation

Malik et al have proposed Security Patterns for Automotive Systems [10] to deal with the a collection of security design patterns targeted for automotive cybersecurity needs. We leverage and extend the de facto standard template used to describe design patterns to include fields specific to the automotive domain and SAE J3061 cybersecurity development guidelines

Nardi et have proposed From Stateflow Simulation to Verified Implementation: A Verification Approach and A Real-Time Train Controller Design [12] to develop a self- contained toolkit to translate Stateflow model into timed automata, where major advanced modelling features in Stateflow are supported. Taking advantage of the strong verification capability of Uppaal, we can not only find bugs in Stateflow models which are missed by Simulink Design Verifier, but also check more important temporal properties. Next, we customize a runtime verifier for the generated nonintrusive VHDL and C code of Stateflow model for monitoring.

Nimara et al have propsed evaluating Automated Software Verification Tools [13] to investigate questions regarding effectiveness and efficiency of different tools and their combinations, what the best tool is, if it makes sense at all to apply automated software verification to well-tested software, and whether tools with many or few reports are preferable.

Prause et al [15] have proposed Detection of heart conditions using HRV processor in Matlab Simulink to design the ECG test system to study about normal and abnormal ECG waveforms without really utilizing ECG machines. In this work the QRS complex which is the main component of ECG signal is used to calculate HR which carries information about HRV [14]. Further the model designed in Simulink can be implemented on FPGA.

Teoh et al have proposed Designing a Flex Ray controller — From SDL to State Flow and Simulink blocks: Generation and verification [16] to achieve the design and the implementation of the Flex Ray Controller which consists of the translation of the (Specification and Description Language) SDL diagrams into State Flow diagrams. With these diagrams, we generate the VHDL code of the controller which will be implemented in an FPGA to create the hardware chip for Flex Ray. We had used ModelSim to verify the correctness of the designed blocks. We give in this paper the adopted procedure for the Macrotick block Generation and its verification by Model Sim [17] [21].

Vayada [18] had proposed the Design of Floating-Point Arithmetic Unit for FPGA with Simulink models a VHDL code is generated with the help of Simulink HDL Coder. Dataflow models are validated by simulation in Simulink®. For the experimental validation of the designed floating-point arithmetic units they are embedded in a FPGA evaluation platform [20][22]. The arithmetic operations are executed with a minimum delay or within a single cycle of the respective clock. Required chip area is smaller in comparison to other open source solutions.

Zulkifi et al [19] have proposed the Functional safety methodologies for automotive applications, introduces some basic concepts of functional safety analysis and optimization and shows the bridge with the tradition design flow. Considerations are presented on how design methodologies are capturing and addressing the new safety metrics.

3. Proposed work flow

To model for the particular component to be designed using Simulink – MATLAB. Converting the model (.slx) to the code VHDL and then to check for FPGA outputs. There are 3 possible cases: ASIC design, FPGA simulation output and Hardware design output. We go for the FPGA, to make sure for the compliance check. The proposed work flow is show in the Figure.1.

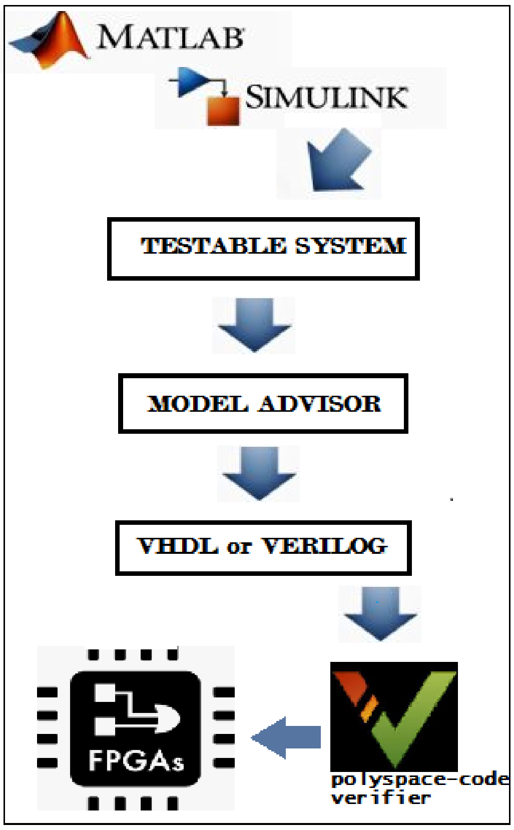


Figure.1 Proposed workflow

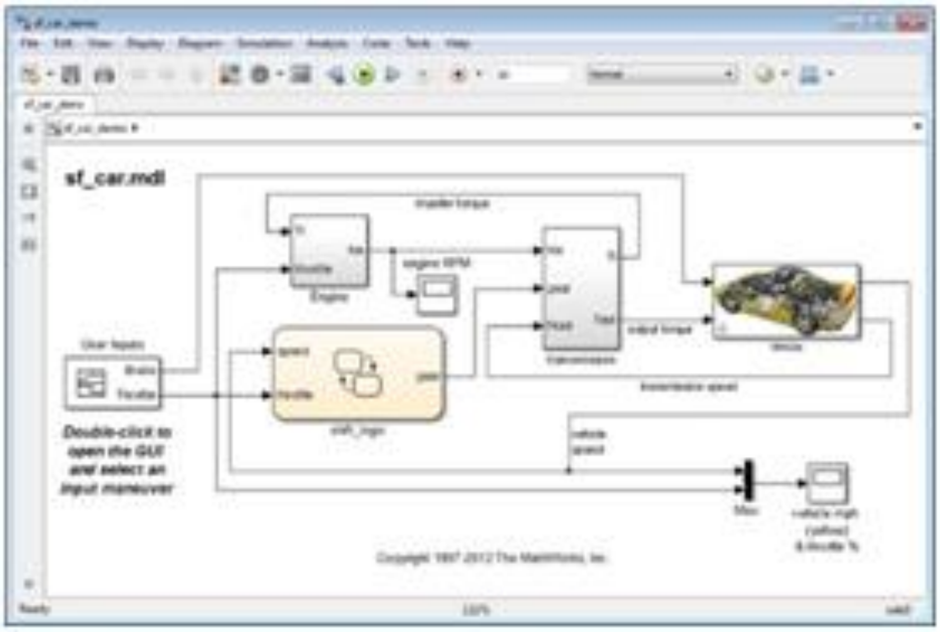


Figure. 2 Test model

The test model is being designed by Simulink in MATLAB, as shown in the Figure.2. ISO rules are taken as the requirement and it is being converted to check for the model in terms of model perspective. These m scripts are been integrated with the model advisor tool to check for the violations. When there is a growth in the embedded systems the complexity level has been increased drastically.

There are group of development persons involved with more skills and with modeling practice. In this challenging development environment, implementing modeling guidelines to ensure modeling consistency is vital. Modeling guidelines establish a homogenous approach within the design team, making it easier to reuse the models for new projects. Guidelines also help new members of the team become familiar with your development process.

For some projects, it is a best practice to use the modeling guidelines into the model. Software systems deployed in high-integrity applications in aerospace and other industries must satisfy rigorous development and verification standards. Industry standards such as ISO 26262, EN 50128, IEC 61508, and DO-178C make modeling guidelines a prerequisite.

4. ISO Rules

The approach recommended will enable to efficiently apply the standard within the modeling team and ease the qualification of your modeling guidelines. The violations are been reported when the certain ISO rule in not followed. Also the AUTOFIX option shall be introduced if there is any possibility of fixing the model is possible. The ISO rules are shown in the Table-1.

Table 1. List of ISO rules to be checked

S. No	Rule concerned	Rule ID	Rule Name
1.	Simulink	jc_0008	Definition of signal names
2.	Simulink	jc_0009	Signal name propagation
3.	Simulink	db_0097	Position of labels for signals and buses
4.	Simulink	na_0008	Display of labels on signals
5.	Simulink	na_0009	Entry versus propagation of signal labels
6.	State flow	jc_0201	Usable characters for subsystem names
7.	State flow	jc_0231	Usable characters for block names

8.	State flow	jc_0211	Usable characters for in-port blocks and out-port blocks
9.	State flow	jc_0243	Length restriction for subsystem names
10.	State flow	jc_0008	Definition of signal names

4.1. JC008: Signal Names

Description:

- Signal names shall be defined for signal lines that output from important blocks. The signal name shall be provided once, at the origin of the signal line.
- A label shall be used to display defined signal names.

Rationale:

- Defining the signal name and displaying the label for the output of meaningful results from important blocks improves the readability of the model.

4.2 JC009: Signal name propagation

Description:

- When defining the signal name for a signal that extends across a hierarchy, signal property Show propagated signals shall be selected so that propagated signal names are displayed.
- In a subsystem with a library
- In subsystems where reusable functions are set
- A signal name is not set at the out port signal of the Bus Creator block

Rationale:

- Prevents signal line connection mistakes.
- Prevents signal line name mistakes.

4.3 DB0097 :position of labels for signals and buses:

Description:

- Signal line labels and bus labels shall not overlap other labels, signal lines, or blocks
- Signal line labels and bus labels shall be positioned at the origin of the connection.

Rationale:

- Adherence to this rule prevents confusion with corresponding names, signal lines, and buses, which improves readability of the model.
- Consistent label position prevents confusion with corresponding labels, signal lines, and buses, which improves the readability of the model.

4.4 NA0008: position of labels for signals and buses:

Description:

- A label shall be displayed on the signal line originating from these blocks:
- Inport From (see exception).
- Subsystem or Stateflow Chart (Stateflow) (see exception)
- Bus Selector (the tool forces this to happen)
- Demux, Selector, Data Store Read (see exception),Constant (see exception)

Exception

- When the signal label is visible in the originating block icon display, the signal does not need not to have the label displayed unless the signal label is needed elsewhere due to a destination-based rule.

Rationale:

- Improves readability, model simulation, and workflow.
- Code generation may not be possible.

4.5 NA009: Entry versus propagation of signal labels

Description:

- When a label is displayed for a signal, the following rules define whether that label is created there (entered directly on the signal) or propagated from its true source (inherited from elsewhere in the model by using the < character).
- Signal labels shall be entered for signals that originate from:
- The Inport block at the root (top) level of a model
- Basic blocks that perform a transformative operation (For the purpose of interpreting this rule only, the Bus Creator, Mux, and Selector blocks also perform transformative operations.)
- Signal labels shall be propagated for signals that originate from:
- Inport block in a nested subsystem
- Basic blocks that perform a non-transformative operation
- Subsystem block or Stateflow Chart (Stateflow) block.

Exceptions:

- When the nested subsystem is a library subsystem, a label can be entered on the signal coming from the Inport block to accommodate reuse of the library block.
- When the connection originates from the output of a library subsystem block, a new label can be entered on the signal to accommodate readability.

Rationale:

- The result of executing a MATLAB command is reflected in the code, which makes consistency between the model and code difficult to maintain.

4.6 JC201:Usable characters for subsystem names

Description:

- Only these character types shall be used in structural subsystem names:
- Single-byte alphanumeric characters (a-z, A-Z, 0-9)
- Single-byte underscore (_)
- Line breaks, single-byte spaces, double-byte characters, and control characters shall not be used.

Rationale:

- Cannot generate code using the configured structural subsystem name.
- May not be able to generate code using the configured structural subsystem name.

4.6 jc_0231: Usable characters for block names

Description:

- Only these character types shall be used for basic block names:

- Single-byte alphanumeric characters (a-z, A-Z, 0-9)
- Single-byte underscore (_)
- Line breaks and single-byte spaces shall not be permitted when adding a new block name. However, they shall be permitted when used initially as a block name that is saved in the Simulink library.
- Double-byte characters and control characters shall not be used.

Rationale:

- Deviation from the rule can make it difficult to maintain the integrity of the model and code.

4.7 jc_0211: usable characters for in-port blocks and outport blocks

Description:

- Only these character types shall be used in Inport and Outport block names:
- Single-byte alphanumeric characters (a-z, A-Z, 0-9)
- Single-byte underscore (_)
- Line breaks, single-byte spaces, double-byte characters, and control characters shall not be used.

Rationale:

- Deviation from the rule can make it difficult to maintain the integrity of the model and code.
- Readability is impaired.

4.8 JC0243: length restriction for subsystem names

Description:

- Structural subsystem name length shall be a maximum of 63 characters.

Rationale:

- Code generation may not be possible.

4.9 JC0008: definition of signal names

Description:

- Signal names shall be defined for signal lines that output from important blocks. The signal name shall be provided once, at the origin of the signal line.
- A label shall be used to display defined signal names

Rationale:

- Defining the signal name and displaying the label for the output of meaningful results from important blocks improves the readability of the model.

5. Results and Discussion

The above listed rules are integrated in the model advisor and then it is checked for the model whether it is followed or not. When the rule is not followed, the current check for which the rule is not followed, violation will be thrown.

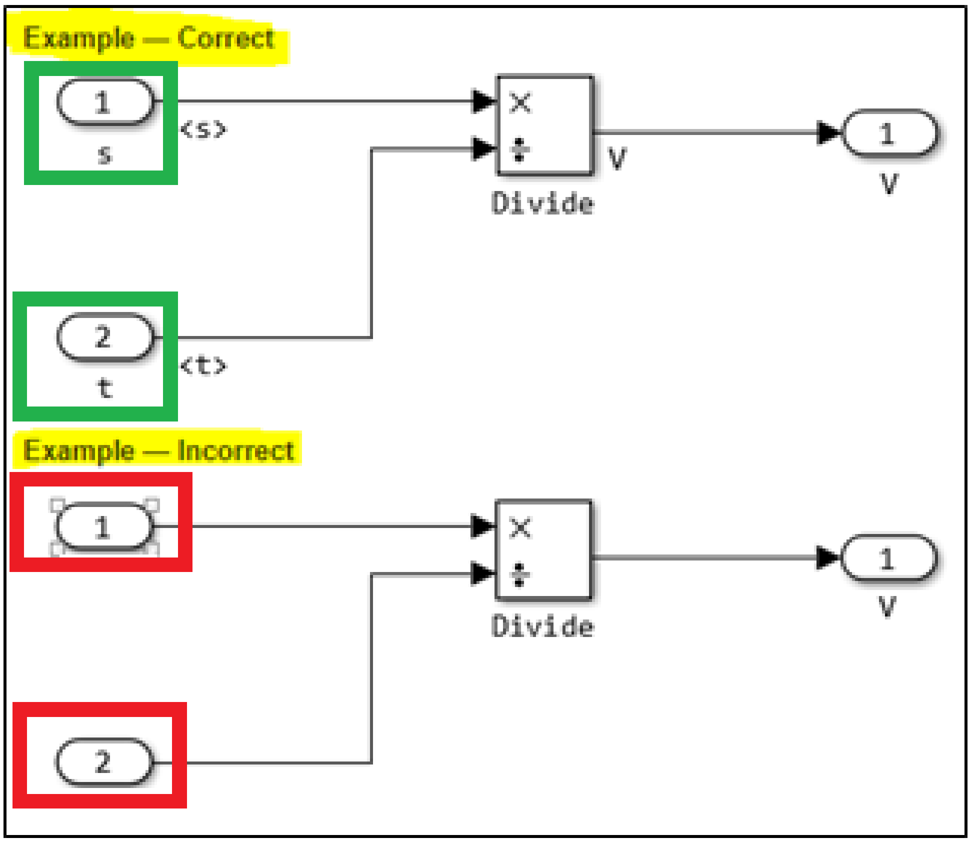
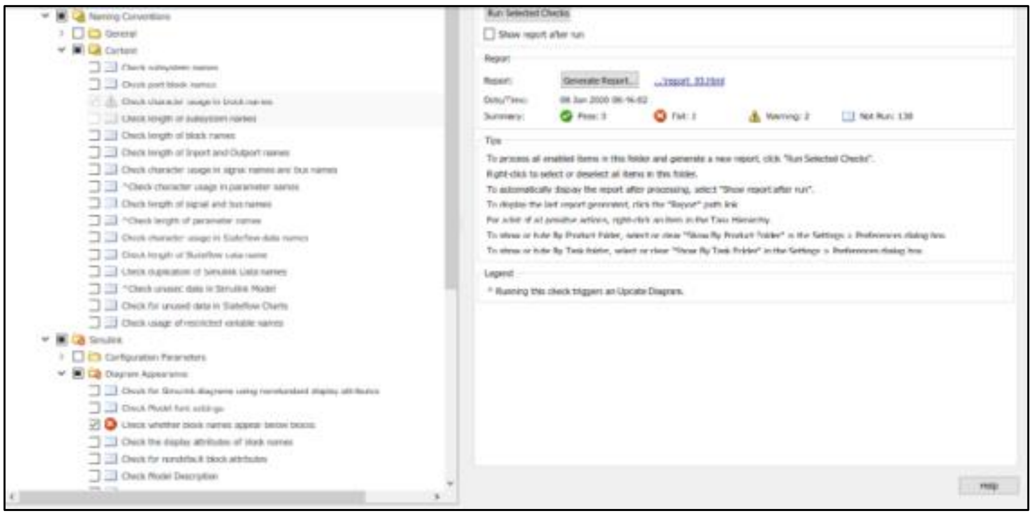


Figure.3 JC008 Modeling

When the model is as per the Figure.3 of incorrect example as shown below for rule jc_008: Definition of signal names, the check gives violation as shown below.



Naming Conventions
General
Content
Check subsystem names
Check port block names
Check character usage in block names
Check length of subsystem names
Check length of block names
Check length of Inport and Outport names
Check character usage in signal names and bus names
^Check character usage in parameter names
Check length of signal and bus names
^Check length of parameter names
Check character usage in Stateflow data names
Check length of Stateflow data name
Check duplication of Simulink Data names
^Check unused data in Simulink Model
Check for unused data in Stateflow Charts
Check usage of restricted variable names
Simulink
Configuration Parameters
Diagram Appearance
Check for Simulink diagrams using nonstandard display attributes
Check Model font settings
Check whether block names appear below blocks
Check the display attributes of block names
Check for nondefault block attributes
Check Model Description

Run Selected Checks
Show report after run

Report
Report: Generate Report... ...report_31.html
Date/Time: 08 Jun 2020 06:16:02
Summary: Pass: 3 Fail: 1 Warning: 2 Not Run: 138

Tips
To process all enabled items in this folder and generate a new report, click "Run Selected Checks".
Right-click to select or deselect all items in this folder.
To automatically display the report after processing, select "Show report after run".
To display the last report generated, click the "Report" path link
For a list of all possible actions, right-click an item in the Task Hierarchy
To show or hide By Product Folder, select or clear "Show By Product Folder" in the Settings > Preferences dialog box.
To show or hide By Task Folder, select or clear "Show By Task Folder" in the Settings > Preferences dialog box.

Legend
^ Running this check triggers an Update Diagram.

Figure.4 Corrected code report

When model level failures are been corrected the code level failures as shown in Figure.4, in case of overflow detection, precision losses, underflow detection, can be found with the help of polyspace, the tool by MATLAB, which is the static code analyzer.

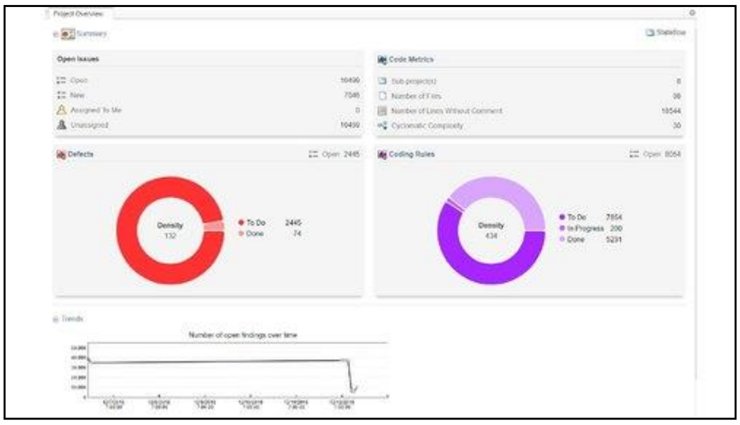
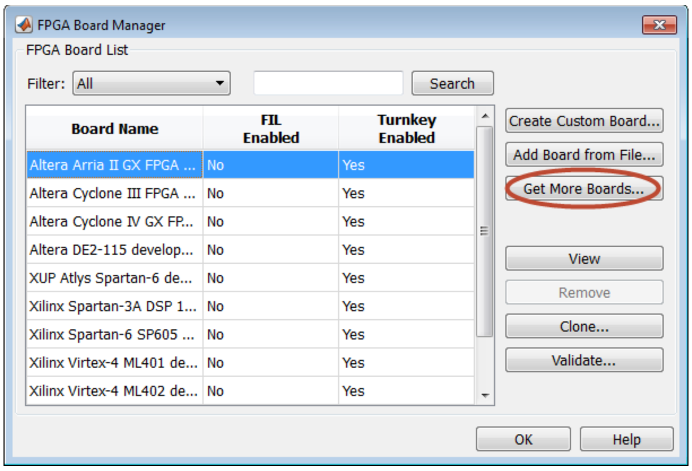


Figure.5 Standard check

The status of standard check is shown in Figure.5. After the standard check in the model advisor, and code check in polyspace, we convert the Simulink model to VHDL code and checking for FPGA


FPGA Board Manager

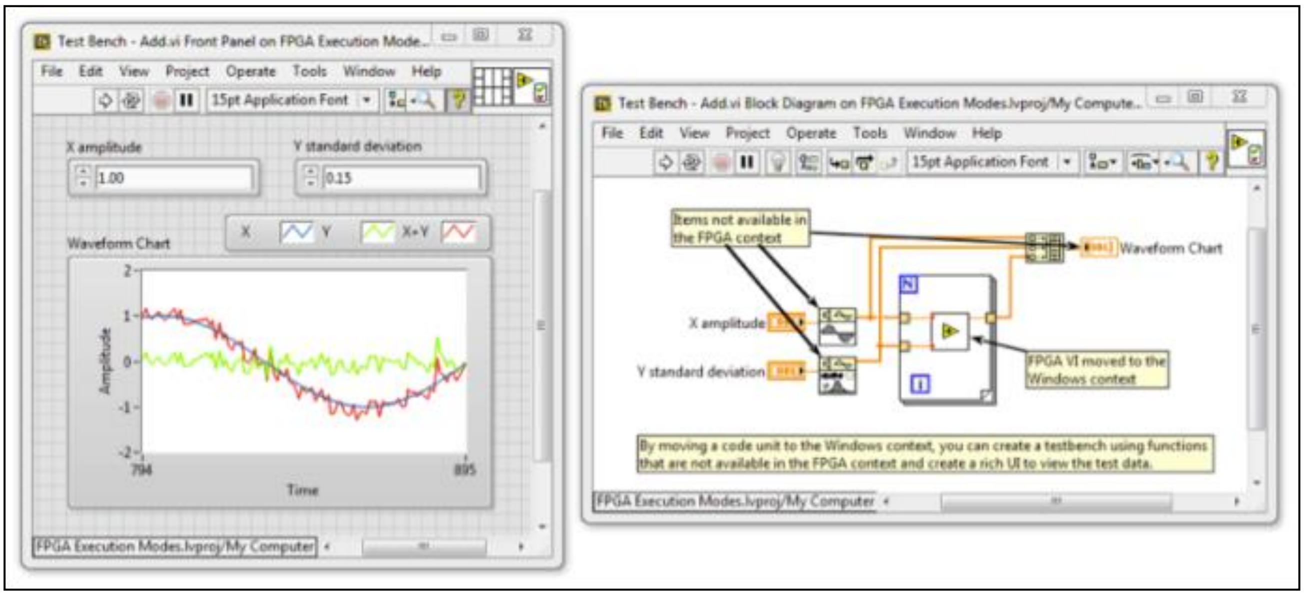
FPGA Board List

Filter: All Search

Board Name	FIL Enabled	Turnkey Enabled
Altera Arria II GX FPGA ...	No	Yes
Altera Cyclone III FPGA ...	No	Yes
Altera Cyclone IV GX FP...	No	Yes
Altera DE2-115 develop...	No	Yes
XUP Atlys Spartan-6 de...	No	Yes
Xilinx Spartan-3A DSP 1...	No	Yes
Xilinx Spartan-6 SP605 ...	No	Yes
Xilinx Virtex-4 ML401 de...	No	Yes
Xilinx Virtex-4 ML402 de...	No	Yes

Figure.6 FPGA board manager wizard

When all those model level and code level failures are detected and resolved, it is more or less effective for the deployment, with the standardized product. Now the code can be deployed and the sanity check can be effectively found in FPGA as shown in Figure.6. The finalized implementation is shown in Figure.7.


Test Bench - Add.vi Front Panel on FPGA Execution Mode...

Test Bench - Add.vi Block Diagram on FPGA Execution Modes.lvproj/My Compute...

Items not available in the FPGA context

FPGA VI moved to the Windows context

By moving a code unit to the Windows context, you can create a testbench using functions that are not available in the FPGA context and create a rich UI to view the test data.

Figure.7 FPGA board manager wizard

6. Conclusion

As a result of any fatal failures, the main drawback would be from the design perspective. So when the designs are corrected in the earlier stage this would be more useful to reduce many problems. So making the ISO check into consideration, the product becomes a standardized one. Also the code generation check would be helpful to reduce the memory consumption of the ECU, in case of removal of any unreachable code. When the standardized design and code is deployed in the ECU, the component reaches the best quality level in the market and also it acquires the customer satisfaction rating to its peak.

REFERENCES

1. Bahig, G., & El-Kadi, A. (2017). Formal verification of automotive design in compliance with ISO 26262 design verification guidelines. *IEEE Access: Practical Innovations, Open Solutions*, *5*, 4505–4516.
2. Ban, D., Jin, S., Yoo, C., & Ju, S. (2017). Design guideline of the EMB controller based on ISO26262. *2017 2nd IEEE International Conference on Intelligent Transportation Engineering (ICITE)*. IEEE.
3. Bassoli, M., Bianchi, V., & De Munari, I. (2020). A model-based design floating-point accumulator. Case of study: FPGA implementation of a support vector machine kernel function. *Sensors (Basel, Switzerland)*, *20*(5), 1362.
4. Cheng, B. H. C., Doherty, B., Polanco, N., & Pasco, M. (2019). Security patterns for automotive systems. *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*. IEEE.
5. Zulkifli, Saiful A. (2017). Development of vehicle powertrain simulation module by interfacing Matlab-ADVISOR to LabVIEW.
6. Sandhiya, D., & Thiyaneswaran, B. (2017). Extraction of dorsal palm basilic and cephalic hand vein features for human authentication system. *2017 International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET)*. IEEE.
7. Imparato, A., Maietta, R. R., Scala, S., & Vacca, V. (2017). A comparative study of static analysis tools for AUTOSAR automotive software components development. *2017 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. IEEE.
8. Jiang, Y., Yang, Y., Liu, H., Kong, H., Gu, M., Sun, J., & Sha, L. (2016). From stateflow simulation to verified implementation: A verification approach and A real-time train controller design. *2016 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE.
9. Thiyaneswaran, B., Anguraj, K., Kumarganesh, S., & Thangaraj, K. (2020). Early detection of melanoma images using gray level co-occurrence matrix features and machine learning techniques for effective clinical diagnosis. *International Journal of Imaging Systems and Technology*, (ima.22514). doi:10.1002/ima.22514
10. Malik, A., & Sharma, R. K. (2017). Detection of heart conditions using HRV processor in Matlab simulink. *2017 International Conference on Trends in Electronics and Informatics (ICEI)*. IEEE.
11. Mejdi, H., Jedli, B., & Hasnaoui, S. (2017). Designing a FlexRay controller — From SDL to StateFlow and Simulink blocks: Generation and verification. *2017 8th International Conference on Information and Communication Systems (ICICS)*. IEEE.
12. Nardi, A., & Armato, A. (2017). Functional safety methodologies for automotive applications. *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE.
13. Nimara, S., Popa, D. B., & Bogdan, R. (2017). Automotive instrument cluster screen content validation. *2017 25th Telecommunication Forum (TELFOR)*. IEEE.
14. B Thiyaneswaran, K Anguraj, M Sindhu, N. S Yoganathan, J Jayanthi. (2020). Development of Iris Biological Features Extraction for Biometric Based Authentication to Prevent Covid Spread. *International Journal of Advanced Science and Technology*, *29*(3), 8266–8275.
15. Prause, C. R., Gerlich, R., & Gerlich, R. (2018). Evaluating automated software verification tools. *2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST)*. IEEE.
16. Teoh, J. X., Stella, M., & Chew, K. W. (2019). Performance analysis of electric vehicle in worldwide harmonized light vehicles test procedure via vehicle simulation models in ADVISOR. *2019 IEEE 9th International Conference on System Engineering and Technology (ICSET)*. IEEE.
17. B Thiyaneswaran, V Bhuvaneshwaran, M.S Dharun, K Gopu, T Gowsikan (Ed.). (2020). *Breathing level monitoring and alerting by using embedded IOT* (Vol. 10). Journal of Green Engineering.
18. Vayada, M. G. (2012). *Hardware Software co-simulation for Image Processing Applications*.
19. Zulkifli, S. A., & Dali, M. A. M. (2017). Development of vehicle powertrain simulation module by interfacing Matlab-ADVISOR to LabVIEW. *2017 IEEE Conference on Systems, Process and Control (ICSPC)*. IEEE.

20. Thiyaneswaran, B., Saravanakumar, A., & Kandiban, R. (2016). Extraction of mole from eye sclera using object area detection algorithm. *2016 International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET)*. IEEE.
21. Elavarasi, S. Anitha, J. Jayanthi, K. C. Rajeswari, and V. Nandini. "AI Based Personalized Healthcare Application to Keep Up the Well-Being of People by Boost the Immunity through Food and Physical Activity during Pandemic." Annals of the Romanian Society forCell Biology (2021): 2422-2435
22. Jayanthi, J., and S. Rathi."Personalized Web Search Methods--A Complete Review." Journal of Theoretical & Applied Information Technology 62.3 (2014).