JAIC
Journal of Applied Instrumentation and Control

Model and Software in the Loop with Automatic Code Generation for Indicator Lights Warnings

José Jair Alves Mendes Junior, João Henrique Zander Neme, Max Mauro Dias Santos, Sergio Luiz Stevan Jr.

***Abstract*— In this paper Model Based Design (MBD) is used to simplify the development of three types of visual warning in an automobile. Along the signals the proposed system provides the outputs to send encapsulated signals in an automotive network. The method presented in this paper uses auto code generation to ease the developers work and give engineers experienced in embedded programing a very useful option to avoid expenditure of time often waste in manual programing. The C code generated automatically can be used in several hardware and the methodology presented is helpful in a wide broad of applications.**

Index Terms— Model Based Design; Model in the Loop; Software in the Loop; Warning Lights.

I. Introduction

The quest for costs and time reduction in new products development are the goal for every industry. Many industries are under pressure to reduce their development times when they produce unique and innovative products [1]. Model Based Design (MBD) is a prominent option to achieve standardization in system development. Conceptually, in MBD, the development process is executed in a single platform, which allows developers to create models of the desired plant and controller, resulting in a functional and easy to verify system [2-3].

One of advantages of MBD is the ease of engineers and designers to find and fix errors early in the system design, at a stage where time and cost for system modifications are diminished [4]. Another key advantage of MBD is the use only one software platform for testing and verification during most part of the development period. Therefore, using the same software one can create, test, and find errors in the logic. Ideas and improvements for the main project can be idealized and executed in the early stages of MBD. To begin a model, it is necessary to define the system requirements according to the controller actuation.

Thus, the present work is directed to the early stages of MBD, called Model in the Loop (MIL) and Software in the Loop (SIL), but mainly focusing in the SIL and the auto code generation, held in a drive process of brake warning lamps ABS, alert brake, and Stability Control (ESC). Highlighting such steps, it is possible to prove that by executing the first two loops of development, one can ensure that most part of the final project is obtained and most errors can be detected.

To illustrate the ease of development promoted by MBD, the first two steps (MIL and SIL) can be executed and verified in the same software. As mentioned before, MIL will be only mentioned succinctly. The main focus of this work is the SIL stage with the auto code generation feature.

The present work is divided as follow: Section II explores the MBD process, Section III shows the application requirements, Section IV presents the MIL, Section V presents the Test Case to this application, Section VI shows the Auto Code Generation process, Section VII shows the Results and the Discussion, and at least, Section VIII presents the Conclusion.

II. Model Based Design

MBD is a design methodology widely used for automotive embedded software, where the engineering process gets together Original Equipment Manufacturer (OEM) and suppliers, benefiting with information exchange, development workflow, and toolchain in a standardized way [5].

MBD comprises four validation techniques, which are four stages of operation, called Model-in-the-Loop (MIL), Software-in-the-Loop (SIL), Processor-in-the-Loop (PIL), and Hardware-

José Jair Alves Mendes Junior, Federal Technological University of Paraná (UTFPR), Ponta Grossa, Paraná, Brasil, e-mail: jjjunior@utfpr.edu.br;
João Henrique Zander Neme, Federal Technological University of Paraná (UTFPR), Ponta Grossa, Paraná, Brasil, e-mail: neme@outlook.com;
Max Mauro Dias Santos, Federal Technological University of Paraná (UTFPR), Ponta Grossa, Paraná, Brasil, e-mail: maxsantos@utfpr.edu.br;
Sergio Luiz Stevan Jr, Federal Technological University of Paraná (UTFPR), Ponta Grossa, Paraná, Brasil, e-mail: sstevan@utfpr.edu.br.

in-the-Loop (HIL), as shown in Figure 1 [6]. On the first steps (MIL, SIL, and PIL) the importance of MBD is emphasized, showing the effectively and efficiently of the methodology. Before listing these characteristics, it is necessary to understand what these steps are.

These stages of development can greatly simplify the work of creating control software for mechatronic systems. Commonly the control software is called controller and the mechatronic system to be controlled is called plant.

In model based software development, the design gradually develops into an executable artifact with different layers of graphical abstractions called models [7]. These models can be used to automatically generate executable codes, suitable for the target platform, using code generators [8].

MIL is the central reference for the next stages, since at this stage the operating logic is defined and also the control strategy and the definition of all input and output variables is executed. From MIL, when it comes to verification, all subsequent stages serve to verify the logic developed in this first stage. At this step it can be noted if the system is implementable

Within the MIL phase and in MATLAB/Simulink® environment, is also developed the functional model of the desired system. This way one can make a very acceptable simulation of the system in operation. Diagrams and state diagrams in Simulink®/Stateflow® are the starting points for function prototyping. They represent the controller and its environment graphically and can be simulated offline for a first validation [9].

As the development of systems at this stage is very fast, changes and system’s test can be executed faster compared to the other steps in traditional approaches. Generally, the MIL development platform is the same as the SIL and PIL, which allows greater flexibility in the designed model.

The great ease of MIL comes from the use of programming in graphical form. With this feature, the focus is not based on structured textual codes, but graphics engines, which help in viewing and understanding. This characteristic is important because the model will be design not only by one individual developer, but also by a range of professionals who can reduce the understanding time compared to textual structures.

Thus, MIL breaks down barriers created by traditional development, as all professionals involved in the development work on the same platform. The desired requirements for the system are observed by the same group that will develop the first functional design of the project and on the same software. After the development of the operating logic a test case is created to be inserted as system input signals. If on the simulation any requirement were not met, the same group has the ease to find the problem.

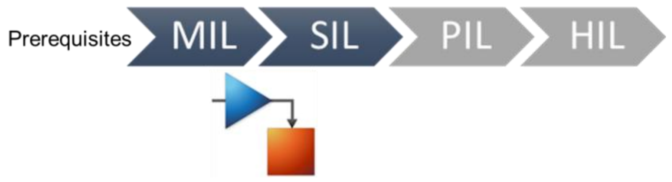
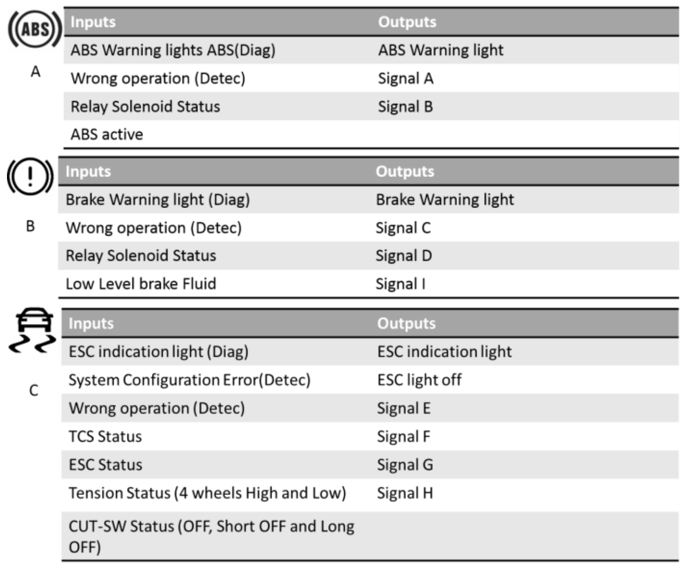


Fig. 1 MBD phases. The highlines phases are the ones executed in this work.

III. Requirements

Each system (brake warning lamps ABS, brake warning, and stability control) has its peculiarities. Figure 2 shows the relationship of the system’s inputs and outputs. Entries are related to state variables and equipment, functions of diagnostic tool - marked "Diag" - and fault detection in the system - marked "Detec".

The diagnostic functions represent the use of specific equipment, which can be connected to the automotive communication network for diagnosis. As it is inserted, it must send targeted commands to the devices, sensors, and actuators, and they should respond, triggering, or not and/or sending signals. Its usefulness lies in the ease of finding faults in devices, since it can check systems one by one. Therefore, it was considered for each system, in addition to the general input that represents the connection of the diagnostic tool, an input variable to simulate the trigger action of the lamps and actuators of each system. Besides the state of the lamps, corresponding signals are also sent that can be encapsulated in an automotive network (for example, CAN and FlexRay Protocol) representing the action that is occurring.



A

Inputs	Outputs
ABS Warning lights ABS(Diag)	ABS Warning light
Wrong operation (Detec)	Signal A
Relay Solenoid Status	Signal B
ABS active	

B

Inputs	Outputs
Brake Warning light (Diag)	Brake Warning light
Wrong operation (Detec)	Signal C
Relay Solenoid Status	Signal D
Low Level brake Fluid	Signal I

C

Inputs	Outputs
ESC indication light (Diag)	ESC indication light
System Configuration Error(Detec)	ESC light off
Wrong operation (Detec)	Signal E
TCS Status	Signal F
ESC Status	Signal G
Tension Status (4 wheels High and Low)	Signal H
CUT-SW Status (OFF, Short OFF and Long OFF)	

Fig. 2 List of inputs and outputs: A) ABS Indicator Lamp, B) Brake Warning Indicator Lamp, and C) ESC Indicator Lamp.

The activities described in Table I show the operation prerequisites of the entire system divided by its general requirements and each step. The requirements also take into account the priority level that every action must have. Therefore subsequent tasks described are less priority than those described first. The general exception is in the insertion of a diagnostic tool, as it can happen at any time, and action should stop any other action that is happening in the software and direct the focus to detect faults and errors.

IV. Model Design

The developed models based on the requirements on Table I are shown in Figure 3 (ABS Indicator Lamp), Figure 4 (Brake Warning Lamp), and Figure 5 (ESC Indicator Lamp). All the models were built on MATLAB/Simulink® platform with Stateflow® diagrams. Each model is a state machine which would determine the output according the established requirements. This models were developed allocating Charts (blocks on which the logic is written) with the highest priority outwardly while the lower priority inwardly. The three Charts were programed with “OR” operation, avoiding to make unnecessary parallelism that could consume memory resources and a greater allocation of data to generate the final code.

These Charts are made by states (rectangles), in which are insert the actions to be performed. Two actions were used: entry, which do one action when the state is set; and during, which do the actions while the state is set. The lines in blue represents the transitions and between the states and its conditions. In each state, there is an initial condition responsible to inform the first state to be performed.

The first two requirements are fulfill with the Chart Tool in all models (Figure 3, 4, and 5) with the “Tool” Chart. They priories the diagnostic tool and monitoring if there are any action to perform. When this device is not connected, the models acting in theirs particularities.

In the Figure 3, when the device is not connect, the requirements from 03 to 07 are fulfill inside the following charts: Rank 2, Rank 3, Rank 4, and Rank 5. Each of these charts are set to be active when a malfunction is detected (Malfunction), the solenoid relay is turned off (Solenoid_Fault), or Antilock Brake System is active (Atilock_Brake).

On the other hand, in the Figure 4, the requirements from 08 to 12 are satisfied. The Charts (from Rank 2 to Rank 5) act when the diagnostic tool is not connect. The Charts are active when there are a malfunction (Malfunciton_EBD), the solenoid has a failure (turned off), or the level of brake fluid is low (Brake_Fluid).

The Figure 5 presents the model developed to perform the requirements from 13 to 18. There are more states in this model due to their requisites, with states from Rank 2 to Rank 6. The transitions are given to the state from system (SyError), malfunction on stability and traction systems (Malfunction_ESC and Malfunction_TCS), problems on solenoid relay

TABLE I SYSTEM REQUIREMENTS

Number	Requirement
	General Requirements
REQ-01	The diagnostic tool has higher priority than any other action
REQ-02	If there is no action to perform, all outputs must remain unchanged.
	ABS Warning Lamp Requirements
REQ-03	When connected with the diagnostic tool, one can turn the warning lamp on and off and send “1” or “0” to signal B.
REQ-04	If a malfunction is diagnosed, the lamp must be on and B must be “1”.
REQ-05	If it is detected that the solenoid relay is turned off, the lamp must be on and B must be “1”.
REQ-06	If the ABS system is active, A must be “1”.
REQ-07	If there is no action to perform, the lamp must be off; also A and B must be “0”.
	Brake Fluid Warning Light
REQ-08	When connected with the diagnostic tool, one can turn the indicator lamp on and off and send “1” or “0” to signal D and I.
REQ-09	If a malfunction is diagnosed, the lamp must be on, also D and I must be “1”.
REQ-10	If it is detected that the solenoid relay is turned off, the lamp must be on, also D and I must be “1”.
REQ-11	If the brake fluid level is low, the lamp must be on and the value 1 sent to C, D and I.
REQ-12	If there is no action to perform, the lamp must be off; also C, D and I must be “0”.
	ESC Indicator Light
REQ-13	When connected with the diagnostic tool, one can turn the warning lamp on and off and send “1” or “0” to signal F and G
REQ-14	If an error is detected in the system configuration, the ESC lamp must be on; also F and G must be “2”.
REQ-15	If a malfunction is diagnosed, the ESC lamp must be on; also F and G must be “2”.
REQ-16.1	If the TCS is on, the ESC lamp must blink and G must be “1”.
REQ-16.2	If the ESC, the ESC lamp must blink and F must be “1”.
REQ-17.1	If 4 wheel drive high is on the ESC lamp must be off, ESC OFF will be on only if CUT-SW Long Off is active, E will receive “1” if CUT-SW Long Off is not active (0 otherwise) and H receives “1” only if CUT-SW is OFF (0 otherwise).
REQ-17.2	If 4 wheel drive low is on, ESC lamp must be off, ESC OFF will be on; E receives “0”, and H receives “0” only if CUT-SW Long Off (1 otherwise).
REQ-18	If there is no action to perform, the lamp must be off; also F and G must be “0”; also E and H must be “1”.

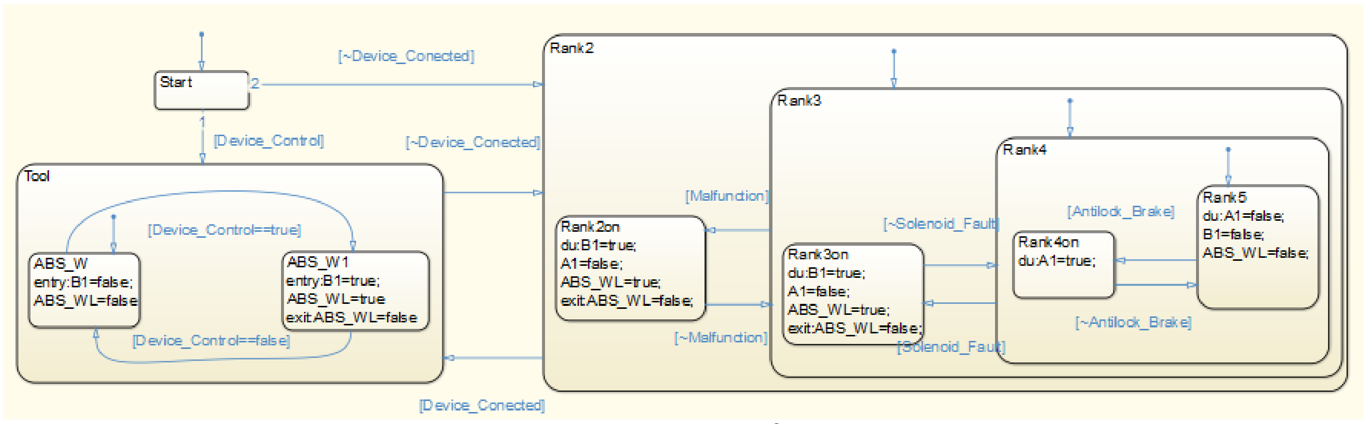


Fig. 3 Developed controller for the desired system on Stateflow® block diagram, for ABS Indicator Lamp.

(Solenoid_Fault), and the conditions among the Stability, Traction, CUT-SW, WDH, and WDL.

V. Test Case

Software test selects the input data to control the test program and observe the outcome after test execution. Developers analyze the final result and can identify possible bugs and improvements. In the software life time, software test is important phrase in the all life and takes up 30-50% cost [10]. A test case suit must be developed in a way that all conditions and logics are placed in test.

Figure 6 shows the values inserted on the system. For the simulation, Simulink® Signal Builders were used to create signs. These were generated in a time interval of 15 seconds. Once the test case is generated, the system is then tested by imputing all created signals in the model.

The verification phase happens when the developers check if all outputs are working as expected. Subsequently, all system's outputs of are presented in Figure 7. After checking all outputs it was proved that the logic contained in the MIL model was corresponding with the requirements.

VI. Auto-Code Generation

Implementation of controller in real-time is a time consuming task in any industry controller design process. Due to increasing demands from the industry, the use of automatic code generation tools, which helps to reduce project completion time, has gained good acceptance [11].

MATLAB® tools for auto-code generation are not novelty in system development. As a matter of a fact there are mature tools that simplify the code generation and saves time and resources. Simulink Coder® generates and executes C and C++ code from Simulink® diagrams, Stateflow® charts, and MATLAB® functions. The generated source code can be used for real-time and nonreal-time applications. Using the model design at MIL it is possible to generate a generic code that can be compiled and used in a specific target.

It is possible to test the generated code using Simulink itself. This feature is eased by Simulink using a block called S-function. An S-function is a computer language description of a Simulink block written in MATLAB®, C, C++, or FORTRAN. Using the same test case as inputs and Simulink® scopes in the

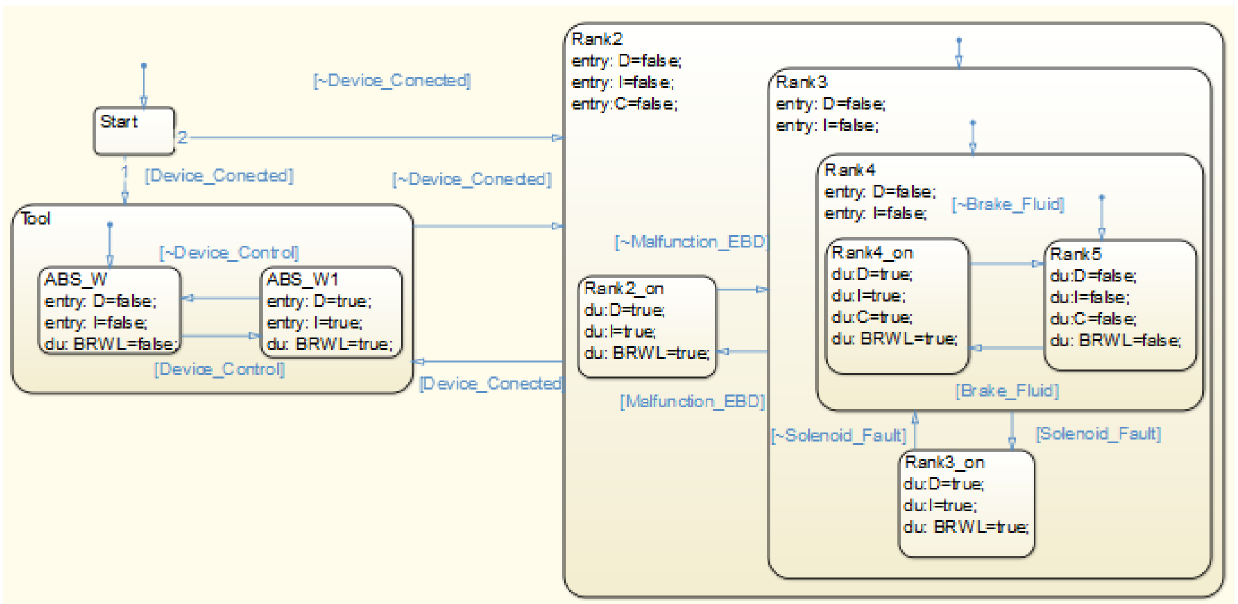


Fig. 4 Developed controller for the desired system on Stateflow® block diagram for Brake Warning Light.

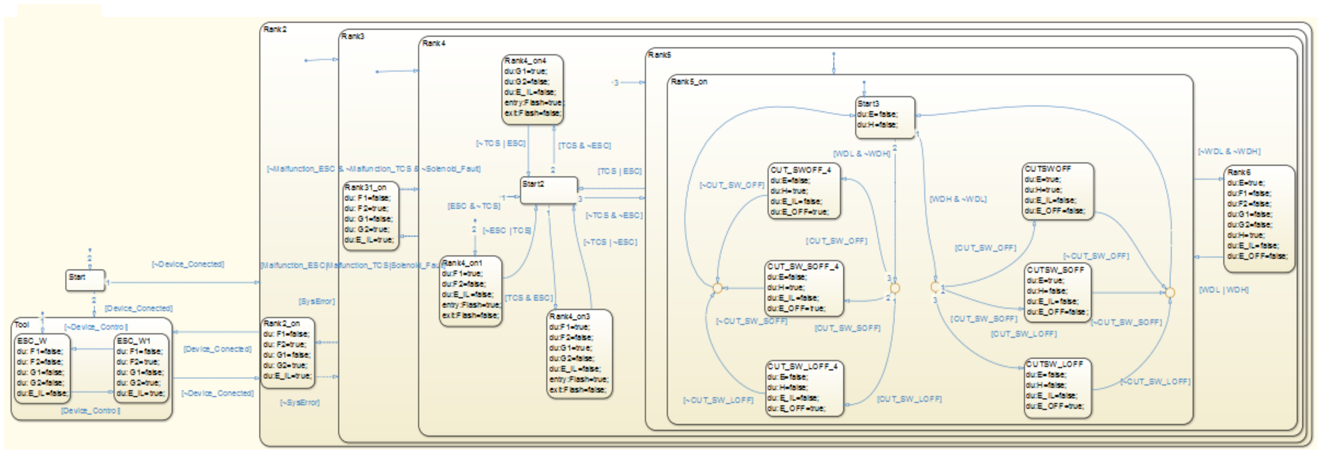


Fig. 5 Developed controller for the desired system on Stateflow® block diagram for ESC Indicator Lamp

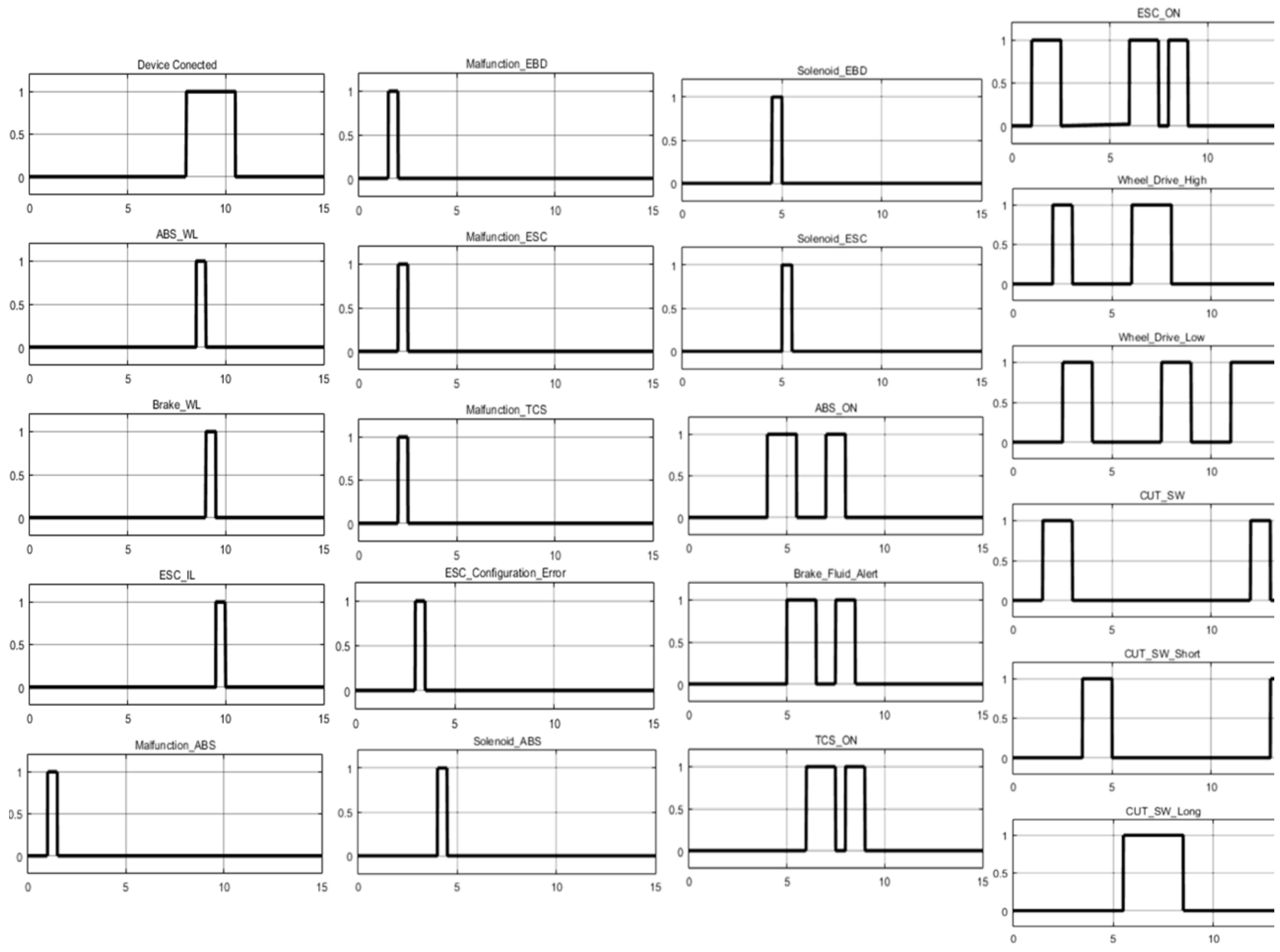


Fig. 6 Designed Test Case for each input. The cases were implemented to map all the possibilities on the controller.

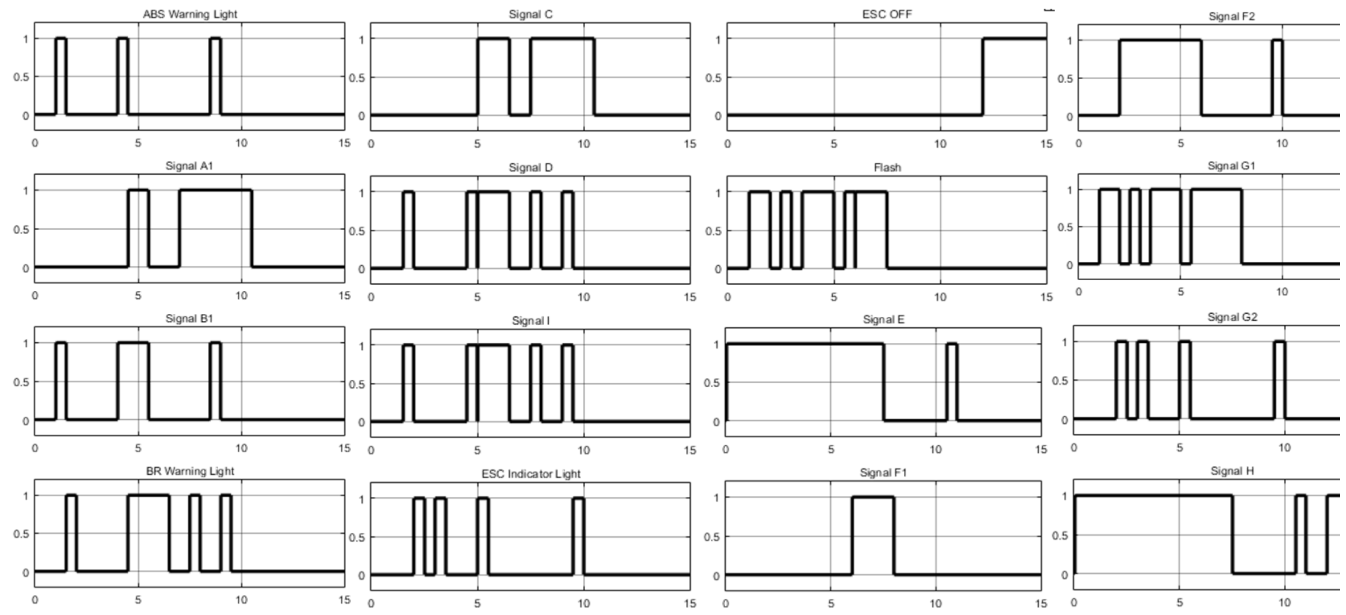


Fig. 7. System outputs after testcase.

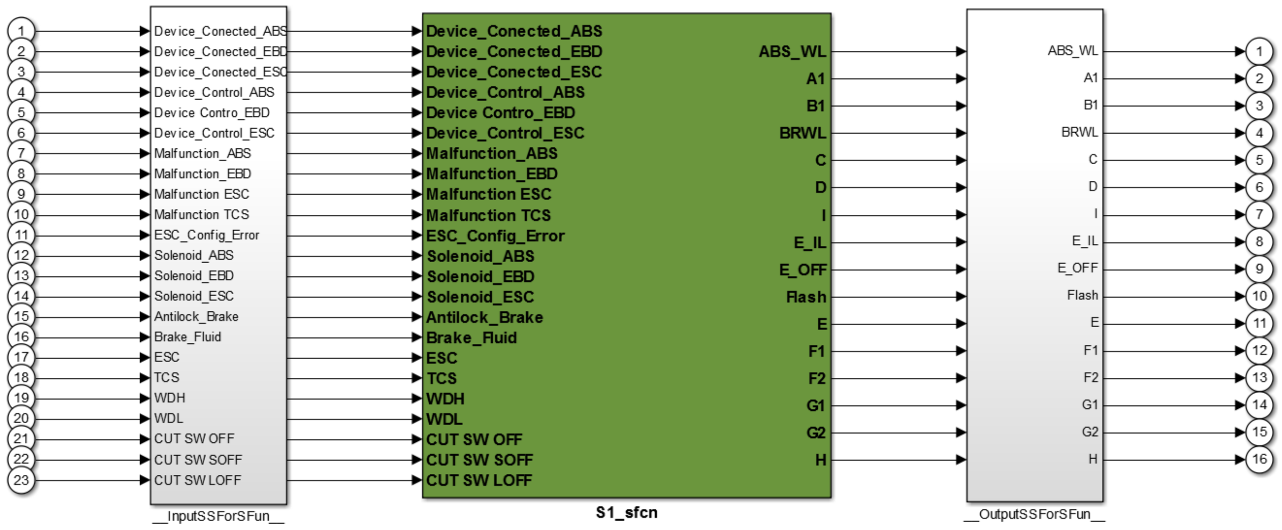


Fig. 8. S-function containing the generated

TABLE II SYSTEM VERIFICATION

	ABS WL		A1		B1		Brake WL		C		D		I	
	Req	Ver	Req	Ver	Req	Ver	Req	Ver	Req	Ver	Req	Ver	Req	Ver
Device Connected	3	3			3	3	8	8			8	8	8	8
ABS WL	3	3	6	6	3	3								
Brake WL							8	8			8	8	8	8
ABS ON														
Brake Fluid Alert							11	11	11	11	11	11	11	11
Malfunction ABS	4	4			4	4								
Malfunction EBD							9	9			9	9	9	9
Solenoid ABS	5	5			5	5								
Solenoid EBD							10	10			10	10	10	10

	ESC IL		ESC Off		Flash		E		F1		F2		G1		G2		H	
	Req	Ver	Req	Ver	Req	Ver	Req	Ver	Req	Ver	Req	Ver	Req	Ver	Req	Ver	Req	Ver
Device Connected	13	13							13	13	13	13	13	13	13	13		
ESC IL	13	13							13	13	13	13	13	13	13	13		
TCS ON					16.1	16.1							16.1	16.1	16.1	16.1		
ESC ON					16.2	16.2			16.2	16.2	16.2	16.2						
Malfunction ESC	15	15							15	15	15	15	15	15	15	15		
Malfunction TCS	15	15							15	15	15	15	15	15	15	15		
ECS Config Error	14	14							14	14	14	14	14	14	14	14		
Solenoid ESC																		
Wheel Drive High			17.1	17.1			17.1	17.1									17.1	17.1
Wheel Drive Low			17.2	17.2													17.2	17.2
CUT SW			17.2	17.2			17.1	17.1									17.1,2	17.1,2
CUT SW Short			17.2	17.2			17.1	17.1									17.1,2	17.1,2
CUT SW Long			17.1,2	17.1,2														

outputs, it is possible to check if the auto generated code behaves exactly like the model design in MIL.

In this work, an S-function was implemented. Figure 8 shows the S-function that contains the generic C-code matching the design model. The middle block is the S-function while the one in the left is the test case block.

VII. RESULTS AND DISCUSSION

It is not sufficient to simply generate code without ensuring it performs exactly as the original algorithm. Table 2 helps to better understand the verification process. As it is possible to see, all requirements were fulfilled.

The numbers inside the tables refer to the requirements described in Table 1. The encapsulated messages trigged by the events are named A1, B1, C, D, I, E, F1, F2, G1, G2 and H. The column "Req" indicates the requirements for the project, while the "Ver" column indicates whether this requirement was verified. For an example, requirement 13 says that if the Diagnostic Tool is connected and the operator sends a signal to turn the ESC Indicator Light on, the Indicator Light must be turned on, and messages F and G must be sent. As can be noted in Table 2, the cells for ESC IL and messages F and G are marked in the "Req" column, indicating that this is the actions required. In addition, the cells on the "Ver" column ate marked because by scrutinizing the signals it was noted that the requirements were verified.

Once the code is build and tested it is possible to compile it for a chosen microcontroller. Simulink allows the developer to automatically compile and load a generic code in a specific target. MathWorks offers a wide range of support packages for different manufacturers. This step would be called Processor-in-the Loop and is not part of the scope of this paper.

VIII. CONCLUSION

The system achieved the proper behavior for meeting the requirements on the MIL phase and the SIL phase confirmed what was expected. It is clear that MBS is a powerful tool for development of complex systems.

Since early stages, it can be seen that most modifications, improvements, and changes can be executed, reducing significantly the time and resources required in the traditional development approach. When working with other systems or methodologies, mistakes, and bugs are only discovered in late stages of development, and the work need to be restarted from scratch. Among these advantages comes the flexibility, because at any time the model can be changed and improved, and the modifications will be easily applied for subsequent phases.

Within the presented proposal, MBD presented itself as a useful tool, explaining why it is used in the automotive and aerospace industries. However, its use can be extended to all development models, as it is a methodological tool for the development of different systems. The auto code generation feature was very helpful, once manual coding is very time consuming. Generation code with Simulink® didn't took more than a few seconds, discarding any need to prove its advantages against the traditional approach. Using the same proposal for future work, we intend to cover the PIL phase, embedding on a microcontroller and proving that the answer corresponds to the one obtained in previous steps.

REFERENCES

[1] T. Kelemenová, M. Kelemen, L. Miková, V. Maxim, E. Prada, T. Lipták, and F. Menda. "Model Based Design and HIL Simulation". American Journal of Mechanical Engineering, 2013, Vol. 1, No 7, 276-281.

[2] T. Lennon. "Model-based design for mechatronics systems". 2007. The Mathworks Inc. Available in: <http://machinedesign.com/archive/model-based-design-mechatronics-systems> Acces in Jul 17 2015.

[3] B. Kirby, H. Kang. "Model Based Design for Power Systems Protection Relays, Using Matlab & Simulink". International Conference on Developments in Power System Protection, 2008, 654-657.

[4] C. Fantuzzi. "Chapter 02 Modeling". Modena: Università Degli Studi di Modena e Reggio Emilia, 2014.

[5] M. M. D. Santos, J. H. Neme, F. R. Franco, S. L. Stevan Jr., W. Torres, A. B. Lugli, A. A. M. Laganá, J. F. Justo. "Model-Based Design of Exterior Lighting Control Function for Automobile:MIL, SIL and RCP". International Journal of Innovative Computing, Information and Control Volume 11, Number 5, October 2015.

[6] E. Bringmann, A. Krämer. "Model-based Testing of Automotive Systems". International Conference in Software Testing, Verification and Validation, 2008, 485-493.

[7] D. Kamma, S. Kumar G. "Effect of Model Based Software Development on Productivity of Enhancement Tasks - An Industrial Study". Software Engineering Conference (APSEC). 2014 21st Asia-Pacific. Jeju, 1-4 Dec. 2014.

[8] G. Karsai, J. Sztipanovits, A. Ledeczi, and T. Bapty, "Integrated development of embedded software," in proceedings of IEEE, vol. 91, Jan. 2003, pp. 145–164

[9] M. M. D. Santos, J. H. Neme, F. R. Franco, S. L. Stevan Jr., W. Torres, A. B. Lugli, A. A. M. Laganá, J. F. Justo. "Rapid Control Prototyping for Automotive Sotware in Power Windows Systems". International Journal of Innovative Computing, Information and Control Volume 11, Number 4, August 2015.

[10] Z. Liu, N. Gu, G. Yang. "An Automate Test Case Generation Approach: Using Match Technique". Computer and Information Technology, 2005. CIT 2005. 21-23 Sept. 2005. 922 – 926

[11] F.C. Teng. "Real-time control using Matlab Simulink". IEEE International Conference, 4 (2000), pp. 2697–270

Received: 21 February 2016
Accepted: 28 July 2017
Published: 15 August 2017

José Jair Alves Mendes Junior is graduated in Automation Industrial Technology at Federal Technological University of Paraná (UTFPR-Ponta Grossa), Master in Electrical Engineering at UTFPR- Ponta Grossa, Doctoral Student in Electrical Engineering and Industrial Informatics at UTFPR-Curitiba, currently is professor on Electronic Department at UTFPR-Ponta Grossa.

João Henrique Zander Neme is graduated in Electronic Engineering at Federal Technological University of Paraná (UTFPR-Ponta Grossa), Master in Electrical Engineering at UTFPR- Ponta Grossa.

Max Mauro Dias Santos is graduated in Electric Engineering at Catholic Institute of Minas Gerais, Master in Electrical Engineering at Federal University of Santa Catarina (UFSC), Doctor in Production Engineering at UFSC, currently is professor on Graduate Program in Electric Engineering at UTFPR-Ponta Grossa.

Sergio Luiz Stevan Jr is graduated in Electric Engineering at Federal University of Paraná (UFPR), Master in Electrical Engineering and Industrial Informatics at UTFPR-Curitiba, Doctor in Electronic Engineering at University of Aveiro, currently is professor on Graduate Program in Electric Engineering at UTFPR-Ponta Grossa and in Graduate Program in Applied Computer at State University of Ponta Grossa (UEPG).