# Observability Checking to Enhance Diagnosis of Real Time Electronic Systems

Manel Khlif, M. Shawky

# Observability Checking to Enhance Diagnosis of Real Time Electronic Systems

Manel KHLIF, Mohamed SHAWKY
*Heudiasyc UMR 6599- Université de Technologie de Compiègne*
*Centre de Recherches de Royallieu BP 20529*
*60200 Compiègne FRANCE*
*Firstname.lastname@hds.utc.fr*

## Abstract

*This paper describes a new property checking approach in order to enhance the diagnosis ability of an electronic embedded system, included in an automotive application. We consider functional diagnosis that is not necessarily oriented towards electronic components, and may result from flaws in the design process. The idea is, at the highest levels of the design procedure, to be able to assess the observability degree of the aimed system. Our approach is based on the analysis of the system simulation results. It allows checking the observability property in a real time electronic system in order to improve its diagnosis capacity.*

*To reach this objective, we have set up an iterative modeling process. The model of the real time electronic system represents the input for the simulation step, which precedes the property checking. It is possible to model the system in two different ways: either with a combined architectural and functional modeling using SystemC, or solely functional modeling using Matlab/Simulink. Then, we check the observability property of the system, if it is not verified, our contribution consists in adding a feedback to the modeling step to improve the model. Otherwise, we validate the result through a test in an electronic automotive platform using as input the Simulink model used for the simulation step.*

*Index Terms*- Observability, property checking, co-modeling, co-simulation, real time system diagnosis.

## 1. Introduction

Car manufacturers propose continuously advanced driving assistance functions that involve more than one computing unit. A computing unit uses information issued from sensors or other computing units, yielding a "system with distributed functions". However, one of the disadvantages of this distribution is the difficulty of the real time diagnosis to detect and localize a fault. To take into account all the computing considerations of a highly distributed architecture, we use a combined modeling methodology that reveals the existing links between architectural and functional modeling [2].

Our contribution in this paper is to enhance the diagnosis ability of a real time distributed electronic system, especially in the automotive field. Therefore, we check the observability of the system necessary for real time on-line diagnosis. The electronic distributed architecture that we study is embedded on board of a truck. It is composed of a set of ECUs (Electronic Computing Unit) connected by the bus CAN (Controller Area Network) [3]. Every ECU is composed of a processor, a memory, a CAN interface and Input/Output interfaces.

To present our contributions, this paper is structured as follows:

First, we present the definition of the observability property. Then, we present the observability property analysis and checking process that we propose in Section II.

Section III and IV present the property checking and the observability analysis principles, as we practice them in our design expansion process.

In section V, we present the design flow starting with a combined architectural/functional modeling (using SystemC). In section VI, we give the details of OBSAN Tool (OBServability ANalyzer) that we have implemented. We discuss the results in section VII and the validation on a physical platform in section VIII.

Finally, we conclude this paper and present our future works.

## 2. Observability Property

Before adding a diagnosis process to a system, we should check its diagnosability [4], [5]. To estimate the "diagnosability level" of a system, the resolution level of faults detection and isolation and faults prediction capacity have to be verified. The presence of a fault is detected by observing of one or more discrepancies. In this way, faults detection is the result of the observation of the discrepancies of the system compared to its appropriate behaviour [6], [7]. Thus, the observation of the behaviour of the system is necessary to validate its diagnosability.

For this reason, we propose in this paper a property called "Observability" as a sub-property of the diagnosability.

We consider that the on-line diagnosis is realized by additional functions with respect to those needed for the usual operation of the system. The local diagnosis functions periodically check the correct behaviour of the nominal functions, by accessing their inputs/outputs, and checking if the actual results correspond to the expected ones.

In our approach, we aim checking the observability property in an electronic automotive system to allow adding a process of real time observation. This should allow an easy observation of the system's behaviour for diagnosis without interfering with the nominal operation of the system (Figure 1).

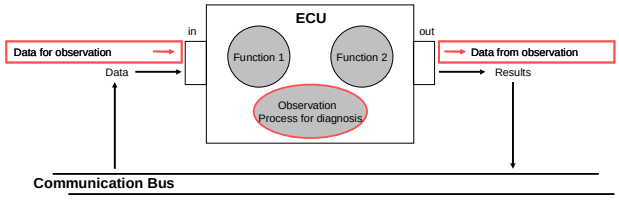



**Figure 1. Observation process.**

We define the observability degree as the time available for on observing process to carry out the periodic surveillance process. We consider that we are dealing with locally synchronous and globally asynchronous digital systems: within a single ECU, all input/output accesses are aligned with the edges of a main clock, and throughout the network of ECUs, the data diffusions are done through asynchronous demands over the CAN bus. Hence, to determine the available time intervals for the observation process, we seek the free cycles not exploited by the nominal operation of the system (Figure 2). After that, we compare the duration of the available cycles with the time necessary to execute the observation process using the same hardware resources. The system is called "potentially observable" if the 'non-occupied' periods allow the execution of the observation process.

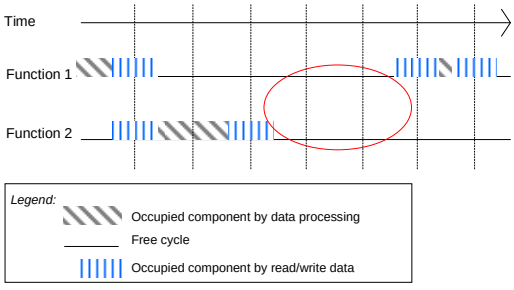



**Figure 2. Processes execution cycles.**

In the next section, we present research works close to our objectives, using different property checking techniques for electronic architectures.

## 3. Electronic Architectures Property checking

The existing techniques of property checking use HDLs (Hardware Description Language) for specification. Most of these techniques use PSL (Property Specification Language) for properties specification. PSL is a language developed by Accellera standards organisation, for specifying properties or assertions about hardware designs [8]. The properties can then be simulated or formally verified. Since September 2004 the standardization of the language has been done in IEEE 1850 working group. In September 2005, the IEEE 1850 Standard for Property Specification Language (PSL) was announced.

Property Specification Language is used with multiple electronic system design languages such as:

- VHDL: Very high speed integrated circuit Hardware Description Language (IEEE 1076),
- Verilog (IEEE 1364),
- System Verilog (IEEE 1800), and
- SystemC by OSCI.

Then, these HDL based techniques use model-checking tools to check the specified properties [9], such as in [10], [11] for SystemC model checking, [12] for VHDL model checking and [13] for PSL.

However, to be able to check a property, it has to be specified in the model, whereas, for our study, we propose an approach of property checking without the need of an early property description.

Electronic design engineers, especially in the automotive field, when they model a new function as a first attempt, they mainly need to optimize its hardware resources use, in order to estimate its cost. Therefore, our technique has a definite advantage as it checks the observability property without adding any more specifications definition.

## 4. Observability Analysis Process

In order to check the observability, first we imagined to analyze the model by parsing the source code to check the free cycles. This approach would have needed a deep analysis of the code, pinpointing the I/O accesses, expanding the loops, etc.; tasks quite equivalent to a compilation phase. We then noticed that during the simulation of the source code, the simulation engine actually follows the same analysis steps. Hence, we oriented our efforts to the interpretation of the simulation results of a model representing the appropriate behaviour of the system, which is easier and faster. In fact,

when we simulate a model, we can generate a trace file recording all the value changes of the system signals and variables. Such trace file should be sufficient to seek free clock cycles by seeking cycles when there are no values changes.

Hence, in order to be able to check the observability of a system, an alternative approach would be to carry out an analysis on generated trace file, in order to detect the non-occupied clock cycles by the nominal operation of the system. It means that we should detect the cycles non-occupied by the value changes during simulation. Then, we check if the system is sufficiently ready to accept on line the observation process for diagnosis in these free cycles. If the free cycles' duration is not sufficient, we should try to propose modifications to improve the basic model (Figure 3).

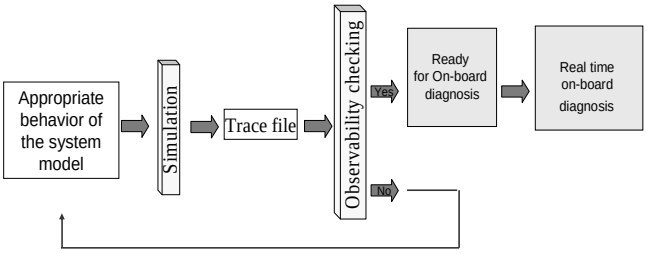



**Figure 3. Observability checking iterative process.**

## 5. SystemC HW/SW Co-design

The automotive industry usually employs Simulink models to express the embedded functions specifications. However, when the hardware architecture reaches a complex level of functional distribution, it becomes difficult to a diagnosis designer to maintain the HW/SW link for each function or sub-function of the system in the diagnosis model. Therefore, to keep track of the hardware mapping of the software functions, we use HW/SW co-design languages to model and simulate the appropriate behaviour of the system representing an automotive distributed function. If we detect a functional fault, we can localize the corresponding hardware module as we know exactly the existing link between the sub-function and the hardware sub-component [2].

### 5.1. SystemC language

One of the most promising SystemC advantages is HW/SW co-modeling to develop virtual platforms, because it supports a unified language of HW/SW modeling [14].

We have selected SystemC as a modeling language because it has many advantages:

- It allows HW/SW modeling with the same language
- The models could be easily connected to any other hardware models [15], or functional models (e.g. in Simulink) [16], [17], [18];
- SystemC environment includes also a simulator: it consists of a C++ library and an event-based simulation engine;
- Any C or C++ library can be included in a HW/SW co-model.

### 5.2. The embedded electronic architecture

The whole architecture consists of n ECUs communicating through the CAN network. In this part of the work, we have modeled the CAN protocol real-time behaviour to realize communications between ECUs models. We have simplified the details to ease the modeling; by implementing a virtual arbiter on the bus. With the Transaction Modeling approach, the communication between components is described as function calls 0.

### 5.3. SystemC co-simulation trace

The simulation of the SystemC model generates a trace file in the VCD format, specified in the standard IEEE 1364-1995. The dump file is structured in a free format. White space is used to separate commands and to make the file easily readable by a text editor. The VCD file starts with header information giving the date, the simulator's version number used for the simulation, and the timescale used. Next, the file contains definitions of the scope and type of variables being dumped, followed by the actual value changes at each simulation time increment.

```
$date
     Mar 17, 2008       10:03:45
$end

$version
              SystemC 2.1.v1 --- Jun 11 2007 17:21:36
$end

$timescale
     1 ps
$end

$scope module SystemC $end
$var wire    1  aaa  clk        $end
$var wire   32  aab  vir_in [31:0]  $end
$var wire   32  aac  vir_out [31:0]  $end
$var wire   16  aad  logic_in [15:0]  $end
$var wire   16  aae  logic_out [15:0]  $end
$upscope $end
$enddefinitions  $end

$comment
All initial values are dumped below at time 0 sec = 0 t
$end

$dumpvars
1aaa
b1 aab
b0 aac
bxxxxxxxxxxxxxxxx aad
bxxxxxxxxxxxxxxxx aae
$end

#1000
0aaa

#2000
1aaa
b10 aab
b111 aad
b111 aae
```

**Figure 4. VCD file format.**

Only the variables that change value during a time increment are listed. The simulation time recorded in VCD file is the absolute value of the simulation time for the changes in variable values (Figure 4).

## 6. OBSAN (OBServability ANalysis) Tool

### 6.1. Working environment

We use Matlab tool as working environment for the implementation of OBSAN Tool. The choice of Matlab was justified by the easiness of the connection of a Matlab program either to Simulink models or SystemC models that we use for the simulation of the system behaviour. Matlab, allows also an easy manipulation of VCD files by converting them in a string array of one column. Thereafter, our tool analyzes the contents of each line of the array by carrying out string manipulations. Finally, Matlab generates graphic charts as output.

### 6.2. Input parameters

As input parameters, OBSAN Tool reads:

- The content of the VCD file,
- the duration of time necessary for the observation per clock cycle,
- the duration of time required by the hardware component for data processing,
- time necessary for accessing the hardware ports for read/write,
- the length of an observation period, expressed as a multiple of the clock period.

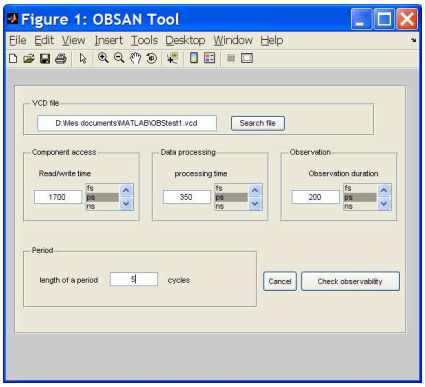



**Figure 5. OBSAN tool input user interface.**

Then, the tool computes the system observability degree based on these inputs (Figure 5).

## 7. Results

Figure 6 shows the first result of observability checking; the message “This system is observable with the mean frequency per period = 4”, means that the free clock cycles are sufficient to accept an observation process whose duration is approximately 4 times the system clock period.

On the vertical axis, we may see that the observability takes a Boolean value: observability= “1” if the correspondent clock cycle in the horizontal axis is sufficient for observation, otherwise, observability=”0”.

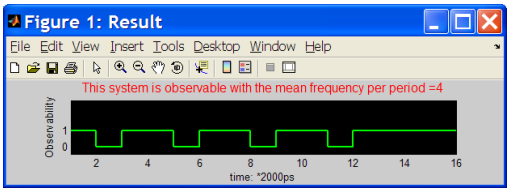



**Figure 6. OBSAN tool output user interface.**

We went a bit further in our analysis by determining “the observability timing plan” of the system. This planning is established from the result of observability checking by adding the observation process when observability=”1” and with a rational frequency, yielding the activation planning of the surveillance process (Figure 6).

## 8. Validation

To validate the results of our observability checking analysis, we applied it to a function that an industrial partner has proposed to us. We first described this function, and then we describe its implementation on our test platform.

### 8.1. Smart Distance Keeping system

We have tested our approach on the Smart Distance Keeping (SDK) function, given by a truck manufacturer. “SDK” is equivalent to the Adaptive Cruise Control (ACC) function, except that the distance/speed regulation is based only on a fixed distance of 50 m (compliant to European regulations for heavy trucks).

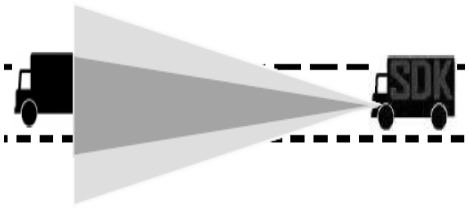


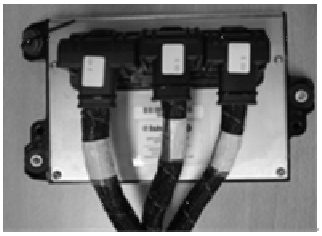

**Figure 7. Smart Distance Keeping function.**

Thus, using an embedded radar, the SDK sub-system maintains a safe headway time, i.e. the inter-vehicle distance is varying as a function of the velocity and is maintained at a minimum legal distance of 50 m (Figure 7), [19].

### 8.2. DIAFORE platform specifications

The platform that we use for tests is called DIAFORE (Diagnosis of distributed functions), and consists of three

ECUs. The first two ECUs are equivalent, while the third ECU is based on a more powerful microcontroller (Figure 8), [20][21]. The ECUs exchange data via the CAN bus.



Microprocessor: Motorola MPC555, 40MHz
Memory: 448K Flash, 26K RAM, 8K EEPROM
Operating Voltage: 9-16VDC
Inputs: 15 Analog Inputs
3 Low Frequency Digital Inputs
1 Emergency Stop Input
Outputs: 12 3A Peak/1A Hold Injector Drivers
6 6A Low Side PWM
1 5A PWM H-Bridge
1 Relay Driver (Main Power)
Datalinks: 2 CAN 2.0B Channels
1 RS485 Channel

**Figure 8. Sample of the target ECUs**

A Smartcraft star node and an interconnection device are used to connect the different ECUs with a development laptop through the CAN bus (Figure 9).

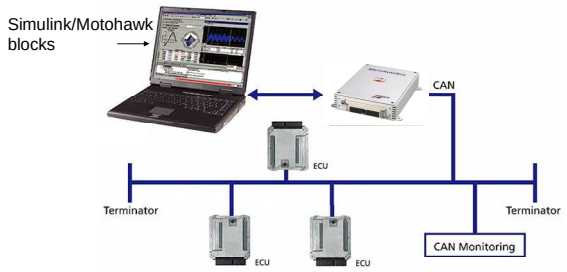



**Figure 9. DIAFORE platform interconnected components.**

## 8.3. Implementing the application on a ECUs network

In order to implement an application on an ECU, first a Matlab/Simulink model of the application should be created. Then, "Motohawk" physical mapping blocks, including I/O blocks, timer blocks, triggers, data storage in memory, CAN bus configurations and more internal controlled circuits have to be instantiated to establish the link with the physical target ECU (programming the physical timers and configuring the CAN bus and I/O pins, Figure 10) [22].

Then, once our Simulink model for the application is completed and ready to be tested, the RTW (Real Time Workshop) Simulink library and the a cross-compilation tool (Greenhills software [23]) bond together to generate the C code of our model that contains both basic Simulink blocks with diagnosis process blocks and physical mapping blocks. Finally, a given tool transfers the assembled generated C code to the ECUs memories. This interactive tool helps also to visualize the input/output values and to control the stored or predefined data.

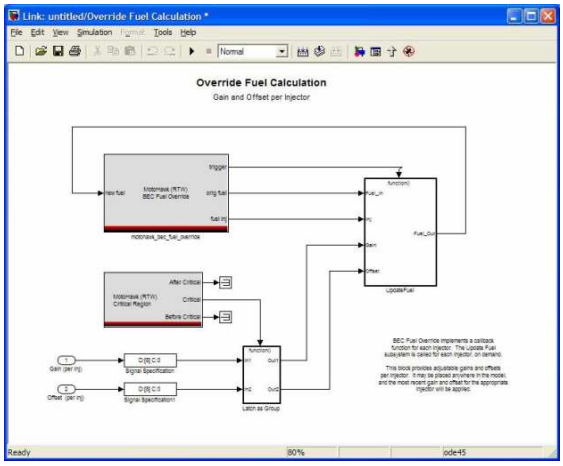



**Figure 10. Simulink Motohawk blocs.**

As the described validation process admits only Simulink models, we had to envisage a second way of modeling based on Matlab/Simulink instead of SystemC. We figured out that Modelsim tool, a mixed VHDL/Verilog/System C simulator (from Mentor Graphics Inc.), has the capability of generating simulation traces in VCD format. This tool could be interfaced to Simulink using "Link for ModelSim library". So we set up an alternative solution where the design is modeled using the usual Simulink blocks, the signals whose observability should be checked are probed by Modelsim "to-vcd-file" blocks. The behavioural model of this co-simulation probe block is done in VHDL.

As soon as a simulation session starts, Simulink and ModelSim collect the results in a VCD file which can be processed using the OBSAN Tool for observability checking as described before.

## 9. Conclusions and perspectives

This paper proposed an alternative approach for enhancing the diagnosis of electronic embedded systems by checking observability. The simulation environments used to validate our approach are SystemC and Matlab/Simulink. The impact of a rigorous property checking method on real time diagnosis for automotive applications has been shown.

However, this result has been obtained for one simulation run of the system. In a complex system, and from an analysis point of view, the system events would have an apparent random behaviour that may be analyzed statistically. Thus, to

improve the realness of our analysis, we need to significantly increase the number of simulation runs.

As a future work, we aim to gather valid observability frequencies of the same model in order to improve our approach with a statistical interpretation of the results. To reach this goal, we need to randomize data inputs values and the value change instants and run numerous simulations using the randomized values. We aim also to extend the observability property at several system levels in order to achieve different levels of observability.

We will also investigate the feedback to the design process when the observability checking tool will be part of an iterative design and modeling process.

## 10. Acknowledgements

The implementation on the physical platform has been done on the equipped vehicles of our laboratory, where the engineering team, including Mr. Dherbomez, Ms Serban and Mr. Tahan has been involved.

## 11. References

[1] Khlif, M., Shawky, M. Co-modelling and simulation with multilevel of granularity for real time electronic systems supervision. *EUROSIM/UKSIM 2008. IEEE 10th International conference on computer Modelling and Simulation.* Emmanuel College, Cambridge, England, 1 – 3 April 2008.

[2] Khlif, M., Shawky, M. Enhancing Diagnosis Ability for Embedded Electronic Systems Using Co-Modeling. *International Joint Conferences on Computer, Information, and Systems Sciences, and Engineering (CIS$^2$E 07 - IETA 07 IEEE conference).* December 3 - 12, 2007 (on line conference).

[3] Paret, D. 2007. Multiplexed Networks for Embedded Systems: CAN, LIN, FlexRay, Safe-by-Wire, *Wiley*, ISBN: 978-0-470-03416-3

[4] Nouioua, F., Dague P. A Probabilistic Analysis of Diagnosability in Discrete Event Systems. *The 18th European Conference on Artificial Intelligence (ECAI'08.* University of Patras, Patras, Greece /July 21st to 25th, 2008.

[5] Hamscher, W. and al, 1992. Readings in model-based diagnosis. *Morgan Kaufmann*,ISBN: 1-55860-249-6, San Francisco, CA, USA.

[6] Misra, A., and al. Diagnosability of Dynamical Systems. *Third International Workshop on Principles of Diagnosis.* pp. 239-244, Rosario, WA, Oct. 12-14, 1992.

[7] Misra A., and al. Modeling Paradigm for Failure Detection, Isolation and Recovery. *Technical Report #95-001,* Measurement and Control Systems Laboratory, Department of Electrical Engineering. Vanderbilt University, Jan. 1995.

[8] Accellera Inc. Property Specification Language Reference Manual-Version 1.1- June 9/2004

[9] Bormann, J. and al. Model Checking in Industrial Hardware Design. *Design Automation Conference*-1995- p= 298-303.

[10] Tahar, S. Assertion and Model Checking of SystemC. *First Annual North American SystemC Users Group (NASCUG) Meeting*- San Diego, California, USA-June 07, 2004

[11] Moinudeen, H. and al. Model Based Verification of SystemC Designs. *Circuits and Systems, 2006 IEEE North-East Workshop on*. Gatineau, Que. June 2006 - ISBN: 1-4244-0417-7, p: 289-292.

[12] D'eharbe, D. and al. Model checking VHDL with CV. *Formal Methods in Computer Aided Design (FMCAD)*, 1997.

[13] Model Checking PSL Using HOL and SMV . Model Checking PSL Using HOL and SMV. *Hardware and Software, Verification and Testing*. ISBN: 978-3-540-70888-9. Springer may 2007.

[14] Grötker, T. and al. 2002. System Design with SystemC. *Springer*, Chapter 8, p. 131. ISBN 1402070721.

[15] Bombana, M. Bruschi, F. 2003. SystemC-VHDL co-simulation and synthesis in the HW domain. *Design, Automation and Test in Europe Conference and Exhibition*, pp. 101-105, Messe Munich, Germany.

[16] Warwick, C. SystemC calls MATLAB. *MATLAB Central*, March 2003, http://www.mathworks.com/matlabcentral/

[17] Czerner, F. and Zellmann, J. 2002. Modeling Cycle-Accurate Hardware with Matlab/Simulink using SystemC. *6th European SystemC Users Group Meeting (ESCUG)*. Stresa, Ilalia.

[18] Boland, J-F. and al. 2004. Using Matlab and Simulink in a SystemC verification Environment. *2nd North American SystemC User's Group*. Santa Clara, CA, USA

[19] Claeys, X. and al. 2003. Chauffeur Assistant Functions. *Report restricted to RENAULT TRUCKS*, Contract number IST-1999-10048, Lyon, FRANCE.

[20] MGM.*Mototron Inc*. ECM-0555-080-0703-F Data Sheet-20/10/2006

[21] MGM.*Mototron Inc*. GCM-0563-048-0802 Data Sheet-14/08/2006

[22] http://www.mototron.com/support/wiki/index.php?title=MotoHawk

[23] http://www.ghs.com/