SystemC FMU for Verification of Advanced Driver Assistance Systems

Keroles Khalil and Magdy A. El-Moursy
Mentor, A Siemens Business, Cairo, Egypt
E-mail: keroles_khalil@mentor.com, and magdy_el-moursy@mentor.com

***Abstract*- Integrated framework to simulate electronic system (including digital and analog devices) with the mechanical parts of a heterogeneous automotive system is presented. The electronic system, consisting of many Electronic Control Units (ECUs), is modeled to simulate the mechatronic system functionality. The recently developed Functional Mock-up standard approach is used to have a model for a complex cyber-physical automotive system. The framework simulates real system including the Hardware (HW) and the Software (SW) to run on the virtual ECUs. It allows Co-development of the automotive system SW and HW while the mechanical system is in the loop. Hardware and Software debugging is demonstrated using the developed methodology. The development cycle for the automotive mechatronic system could be greatly shortened using the proposed framework.**

***Index Terms*- Virtual Platform, Transaction Level Model (TLM), Functional Mockup Interface (FMI), Heterogeneous System, Automotive.**

I. Introduction

Autonomous Driving (AD) requires sophisticated electronic system. The designs of today's automotive electronic systems contain many (System-on-Chips) SoCs [1, 2]. Deep verification is required to ensure that there are no bugs or risks of failure. Safety of Advanced Driver Assistance Systems (ADAS) could not be guaranteed without verification of the whole system including thermal, mechanical, and electrical parts. Cars with ADAS have extensive data processing and decision making capabilities. A driver is either warned so a potentially dangerous situation can be avoided, or the systems on board take over control once this dangerous situation is inevitable (e.g. by trying to steer away from the impact point). In this paper modeling the whole automotive system is adapted to verify the operation of ADAS including the software (SW) and hardware (HW).

Virtual Platforms (VPs) are emerging as the key technology to build virtual prototype for real products and enable the co-design of SW and HW systems. VPs enable fast architecture exploration, early SW development, and efficient performance analysis [3-7]. A VP is the ensemble of simulation models representing HW blocks and their interconnection. VPs are usually modeled via a high-level programming language tailored for modeling and simulation, such as SystemC/TLM2 [1]. In a hybrid automotive system, synergy among different tools/models crossing domains (System of Systems) including Virtual Platforms, Plant simulators, digital simulators/HW Emulators and mechanical simulators is required. Virtual Platforms model ECUs (Electronic Control Unit as Virtual VECUs) including processors, HW peripherals and controllers to run automotive software stacks [8-12]. The ADAS VECUs have multiple sensors on board of which the readings are combined to obtain a better perception of the actual situation. In this paper the presented framework is used to verify both autonomous and non-autonomous VECUs. A VECU which acts as ADAS ECU has AUTOSAR SW [13] that runs on top of it. The VECU is connected with many types of sensors and actuators through Functional Mock-up Interface (FMI) standard [14] as shown in Figure 1.

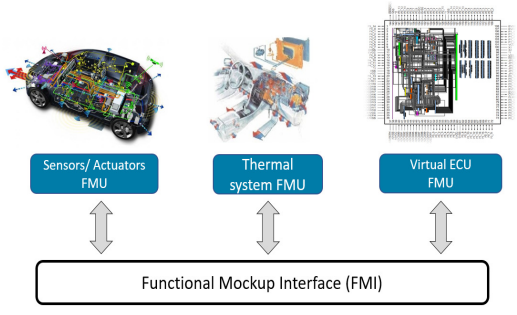

Figure 1. Virtual ECU interfacing with different FMUs

SystemC/TLM VPs are generated using Vista tool [15] and is connected with sensors/actuators created by MATLAB [16] and TASS Prescan [17]. TASS gives the capability to develop dangerous scenarios that can occur in the physical surrounding of a vehicle. Those simulated scenarios allow engineers to have more time to focus on real

system design issues in early stage. Using the proposed framework, the developer can verify his real code on the VECU and deal with many sensors and actuators in different scenarios and study the effects of using ADAS in such scenarios. The presented methodology integrates VECU as a Functional Mock-up Unit (FMU) using FMI.

The work in [1] presents an approach to create VPs as FMUs. The VP is shipped as FMU shared object with FMI model description XML (eXtensible Markup Language) in FMU file. The FMI master loads the VP DLL (Dynamic-link library). The proposed mechanism in this paper is different depending on creating a FMU wrapper which can be loaded by the FMI master with the VP through IPC (Inter-Process Communication) mechanism. The Input/Output (I/O) interconnects of the VP are controlled by FMI V.2.0 co-simulation APIs (Application Programming Interface) triggered by the FMI master. I/O (Input/Output) changes are propagated in an event-driven execution mechanism that is compatible with the nature of the SystemC/TLM based VP. By this mechanism the portability is high comparing to the other mechanism. The VP can be modified and compiled without changing the FMU part. The remainder of the paper is organized as follows. The design methodology using the proposed FMU is presented in section II. A case study of a full automotive system is provided in section III. Some conclusions are presented in section IV.

II. THE DEVELOPED METHODODLOGY

In order to wrap a SystemC virtual platform as FMU, two wrappers are used. One is inside the VP, Internal FMU Control (IFC) and the other is outside, External FMU Wrapper (EFW) as shown in Figure 2. The IFC is the TLM SystemC model responsible for communicating with the EFW and synchronizing the VP with the other FMUs (could be electrical or mechanical). IFC is connected with EFW through IPC mechanism. EFW is FMU consists of XML and DLL. It can be parsed and loaded by external FMI master inside other simulation tool. FMUs are assumed to be compliant with FMI V.2.0 co-simulation standard. Also, it is assumed that the FMUs are provided as pre-compiled shared library (*.DLL on Windows or *.so on Linux systems). Runtime library dependencies are assumed to be available on the same host on which the VP is running. The EFW can be configured to have any number of input and output pins. Analog ports are represented as REAL numbers and digital pins are represented as INTEGER numbers. Each pin has a unique reference value (*V_reference*). EFW configuration can be set in the XML file. IFC has the same configuration of EFW. In addition, the synchronization period parameter "*VP_Do_step_time*" needs to be set. The IFC inside the VP communicates with the EFW DLL which is loaded by another FMI master tool. IPC is established on one physical socket but it has a dynamic array of virtual sockets "*Connectsocket*". The number of virtual sockets depends on the number of I/O ports which should be connected to the VP IOs. The first virtual socket which keeps the synchronization between the VP and the FMI master acts as a synchronization socket. Synchronization is performed through a synchronization signal which is sent by IFC once the *fmi2DoStep* API is called by the FMI master. The *fmi2DoStep* sends the synchronization signal in the virtual synchronization socket and blocks the FMI master until an acknowledgement signal (ACK) comes from the VP as shown in Figure 2. The IFC has Time Control sub-block to handle the synchronization signal. Once it receives the synchronization signal, it allows the VP to run for the equivalent time period which is defined by "*VP_Do_step_time*" parameter.

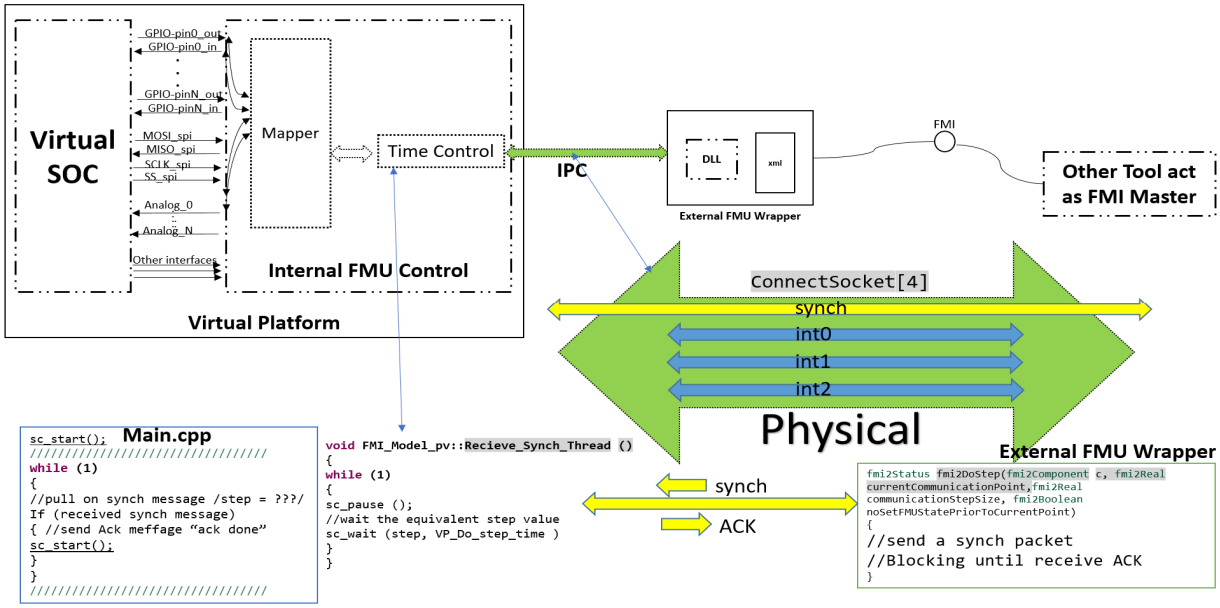

Figure 2. The synchronization mechanism between VECU FMU and the other FMUs

The EFW XML has a dynamic array of *S_FMU_PORT* structure for each pin as shown in Figure 3. The *S_FMU_PORT* structure defines the *FMU_Container_Value* which represents the value to be sent/received from/to the VP to/from FMI master. The *FMU_Container_Type* represents the data type (*REAL* or *INTEGER*). *V_reference* represents the reference value for the pin which is defined in the XML file. If the pin value is updated by any side (master or slave), the updated value is set. The pin direction in the XML file identifies if the pin is input or output. *C_socket* represents the virtual socket which is mapped to the pin as shown in Figure 4. The default initialization value of *V_reference* is -1. Initialization is performed once the FMI master begins calling the FMU initialization APIs.

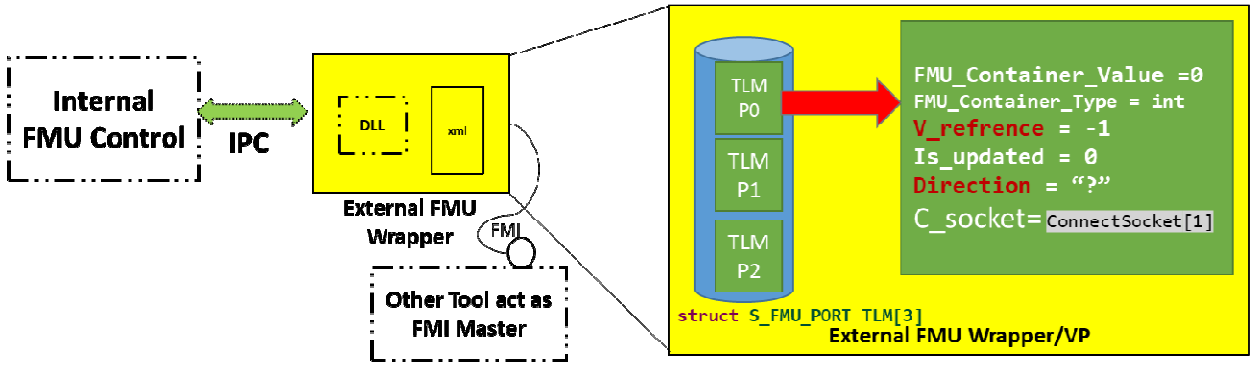

Figure 3. The Dynamic array of FMU ports structure

Once FMI master sends an *INTEGR* value to the FMU, it calls the *fmi2SetInteger* API then EFW searches for all the *S_FMU_PORTs* which have the matched reference value with input direction. EFW sets the *FMU_Container_Value* with the value which is sent by other FMUs. If the XML *V_reference* does not exists then the *V_reference* is set to -1. Finally, the value is sent to the corresponding virtual socket as shown in Figure 4.

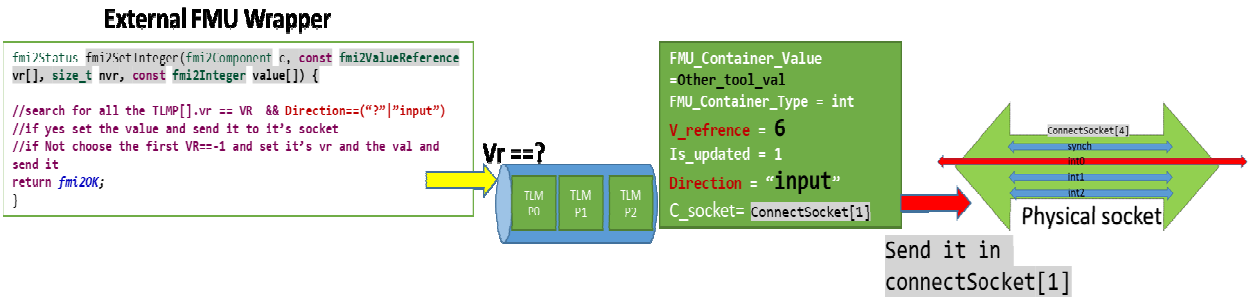

Figure 4. The EFW transmit sequence

On the other side, IFC handles the reception and transmission of the pin values by the Mapper sub-block. EFW sends a string using the IPC virtual socket. The string consists of five data values separated by a semicolon (pin value, data type, variable reference, updated flag and direction). The Mapper inside the IFC parses the string and issues a transaction on the General Purpose input Output (GPIO) TLM pin which matches the *V_reference* in the XML as shown in Figure 5. Each TLM (Transaction Level Model) pin has a unique *V_reference*.

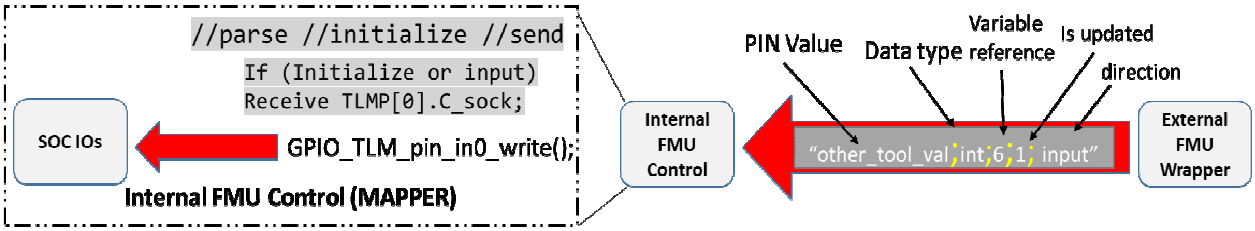

Figure 5. IFC receiving sequence

Similar behavior happens when the GPIO issues a transaction to send data on the GPIO pin. The Mapper sends the data value to the EFW on the corresponding virtual socket. Once the FMI master reads data from the FMU, it calls the *Fmi2getint* API and the saved value is returned to the FMI master. The exchanged data between the FMI master external tool and the VP which is wrapped as FMU slave is shown in Figure 6. A case study of many VECUs communicating together (while one of them ADAS VECU acting as FMU connected to Matlab FMI Master) is presented in the following section.

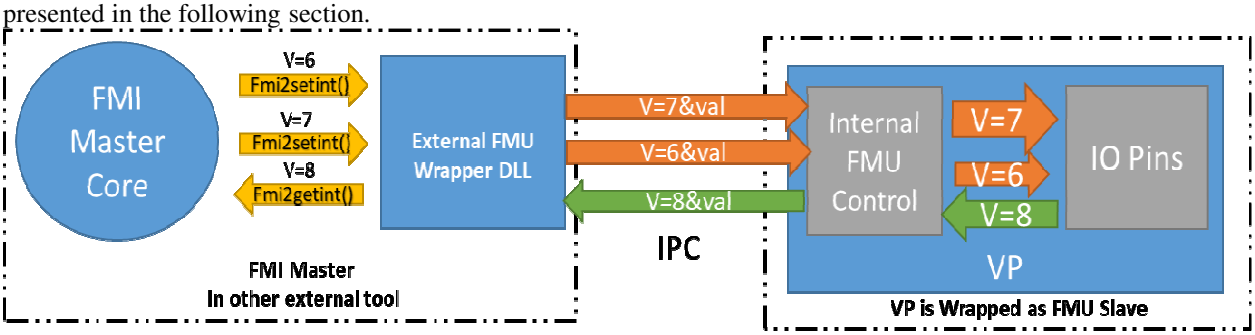

Figure 7. The data exchange between the VECU pins and the FMI master in external tool

III. CASE STUDY

ADAS of an autonomous vehicle consisting of four VECUs connected to Controller Area Network (CAN) Bus is used in the presented case study to demonstrate the use of VP as FMU as shown in Figure 7. A dashboard VECU is modeled using CANoe ® [18] for packet generation and monitoring. Different inputs such as (Pedal/Angle, Brake/Angle, Shift and Stop Engine) and outputs [Monitor Speed/Revolutions per minute (RPM)] are stimulated and observed through the dashboard. Engine Control VECU is modeled as a PowerPC Virtual Platform using Vista Architect® [15] to run the AUTOSAR stack/application. The Engine Control VECU reads input combination from dashboard and sends Command/Pedal/Brake Angle to Transmission Controller VECU. The Transmission Controller VECU is modeled as a virtual SoC based on ARM CortexM3 that runs a baremetal application. It reads Pedal Angle from Engine Controller VECU and calculates RPM/Speed. The braking system is modeled using Simcenter

Amesim® [19] with mechatronic components, exported as slave FMU, and connected to Vista FMI master running on the host machine. The FMU input is a brake force which comes from the Digital-to-Analog Converter (DAC) model inside the Engine Control VECU and the output is the force on the 4-Wheel Calipers of the vehicle.

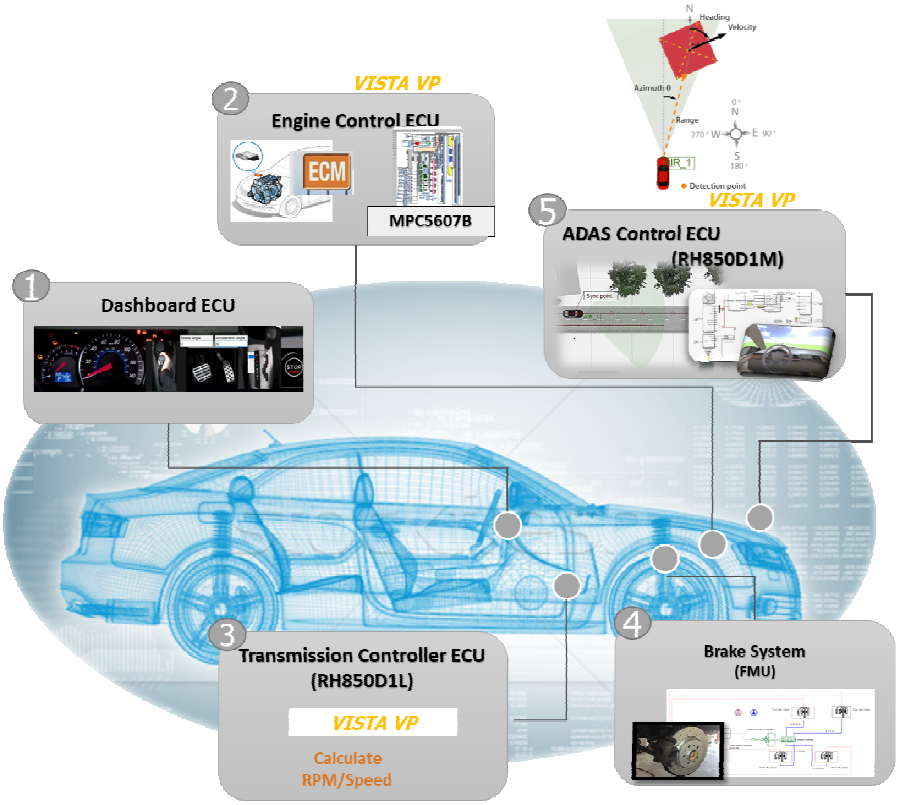

Figure 7. Advanced Emergency Braking System Controlled by ADAS VECU (Case Study)

The scenario begins with Actor Information Receiver (AIR) sensor detecting a man moving across the road modeled using PreScan® [17]. The AIR sensor is connected to a virtual ADAS control VECU running Advanced Emergency Braking System (AEBS) bare-metal application and wrapped as FMU. The ADAS VECU is integrated inside MATLAB Simulink® from MathWorks [16] as shown in Figure 8. The AIR sensor output is zero in case no object is detected. The Analog-to-Digital Converter (ADC) of the VECU is connected to the AIR Sensor. One driving scenario is to be chosen out of 10,000 for testing the ADAS. The simulated vehicle is driven normally on a simulated road. The AIR Sensor in the car detects a person running across the road in front of the car at a distance of 20m. The ADAS VECU uses this sensor information and determines that a collision would occur with the person unless avoidance action is taken. The ADAS sends a signal to the braking system to stop the car.

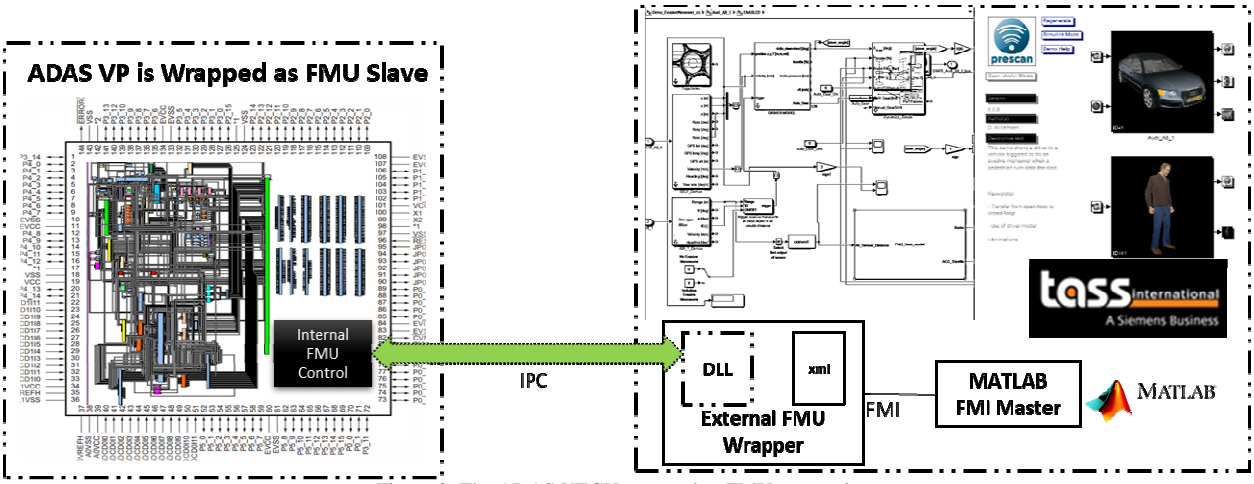

Figure 8. The ADAS VECU wrapped as FMU connection.

Each VECU has a unique CAN ID, the communication between the VECUs is done by CAN frames as shown in Figure 9.

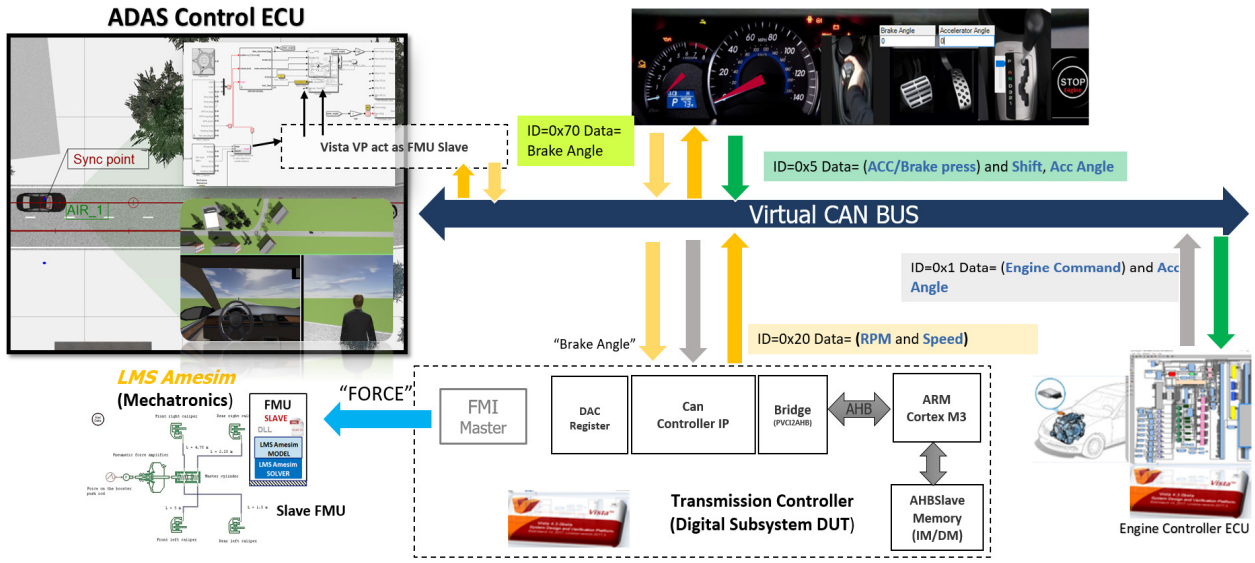

Figure 9. CAN Frames Data Flow on Virtual CAN Bus

ADAS VECU is wrapped as FMU and imported inside MATLAB Simulink. It receives the Speed/RPM CAN Packet from Transmission Controller VECU and Controls the car animation in PreScan. Once a human appears in front of the car, the Sensor detects the object and the ADAS VECU automatically sends a CAN Packet (ID =0x70) with data containing the Brake angle. The Transmission Controller VECU sends a force to the mechanical brake system in Simcenter Amesim and the car stops. The natural decreasing slope of the car speed is generated by road friction. Once the AIR sensor detects the object at distance 20m, the ADAS control VECU sends brake signal frames automatically resulting in a sharp decrease in the car speed and the car stops at distance of 14m. A lot of scenarios can be verified using the presented methodology. Two scenarios are verified in the case study as shown in Figure 10.

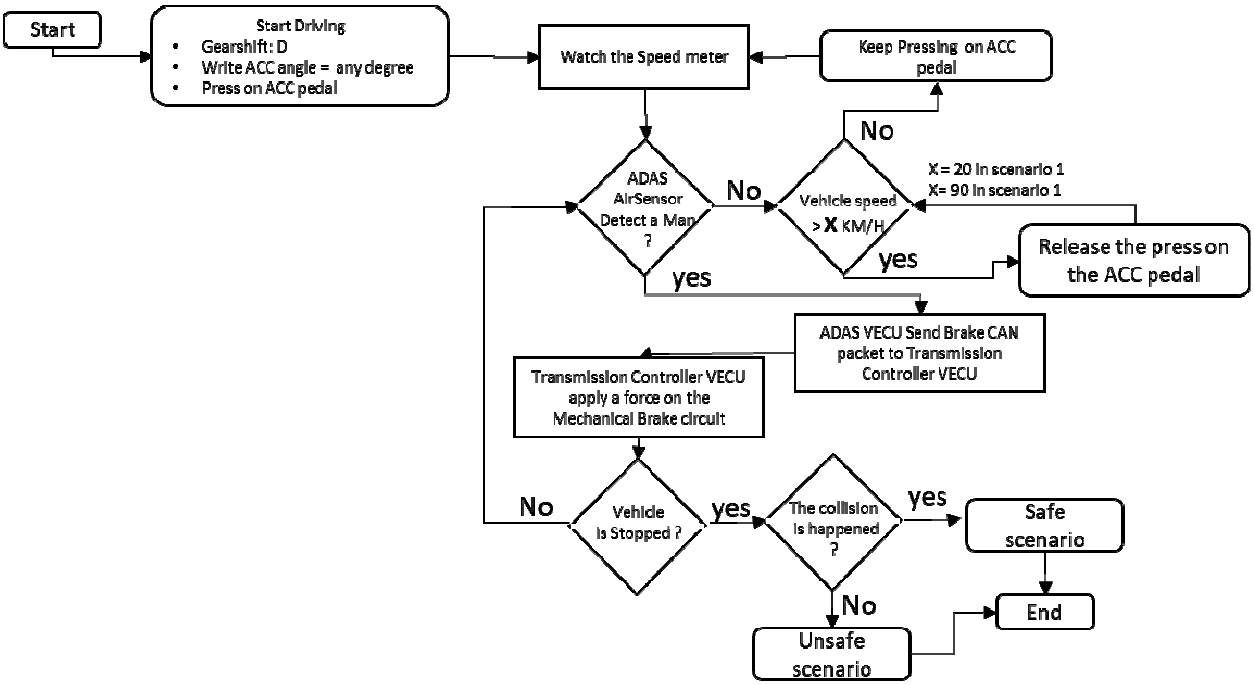

Figure 10. Scenarios flow chart

The simulated scenario is shown in Figure 11. The car speed decreases from 20 km/s to 0 km/s during 1 sec, this is considered a safe scenario.

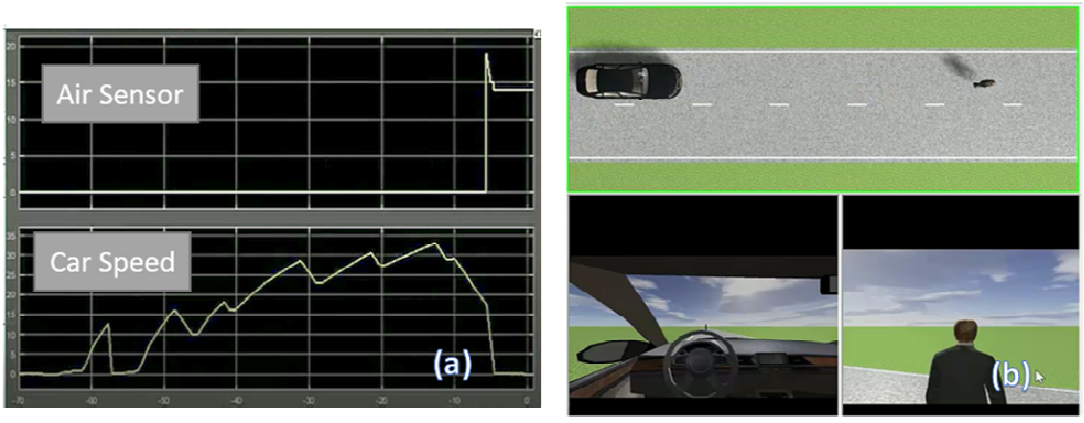

Figure 11. (a) The AIR sensor distance (m) and car speed (km/hr) (b) The ADAS VECU stops the car once it detects the human in front of it

In another scenario, the car speed decreases from 90 km/s to 40 km/s during 971 ms as show in Figure 12. In this case the collision between the vehicle and the human will happen as a result of driving at high speed and suddenly facing the man, this is considered an unsafe scenario. Many sceneries can be verified with different speeds. AUTOSAR SW analysis for each scenario can be generated and verified as shown in Figure 13.

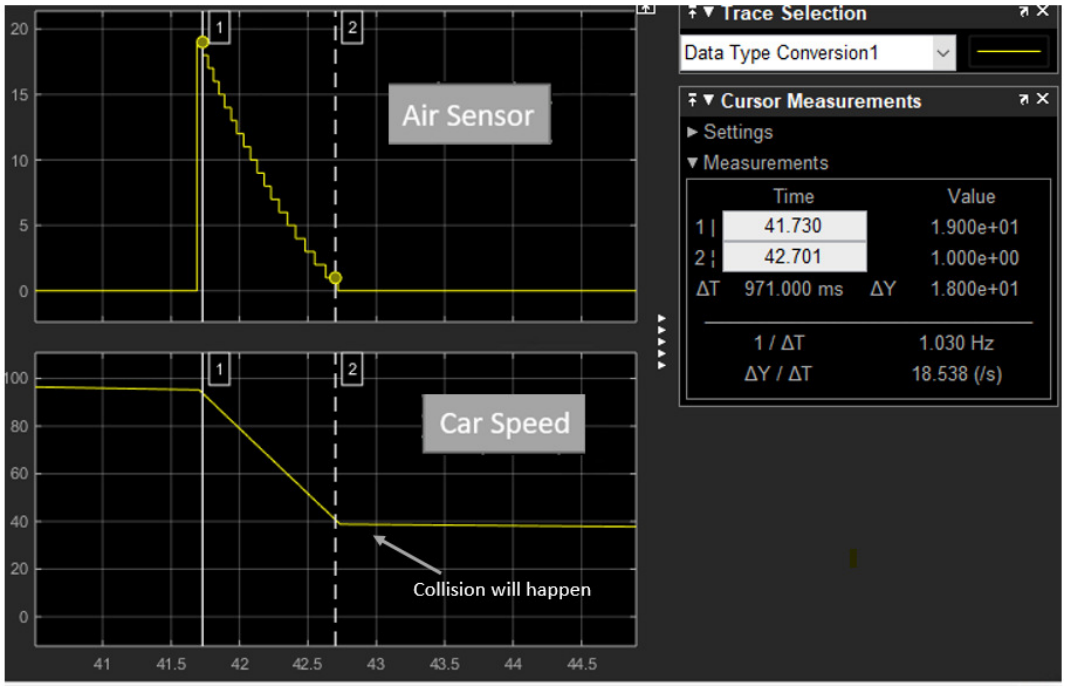

Figure 12. Collision occurs as a result of car running at high speed. The AIR sensor distance (m) and car speed (km/hr)

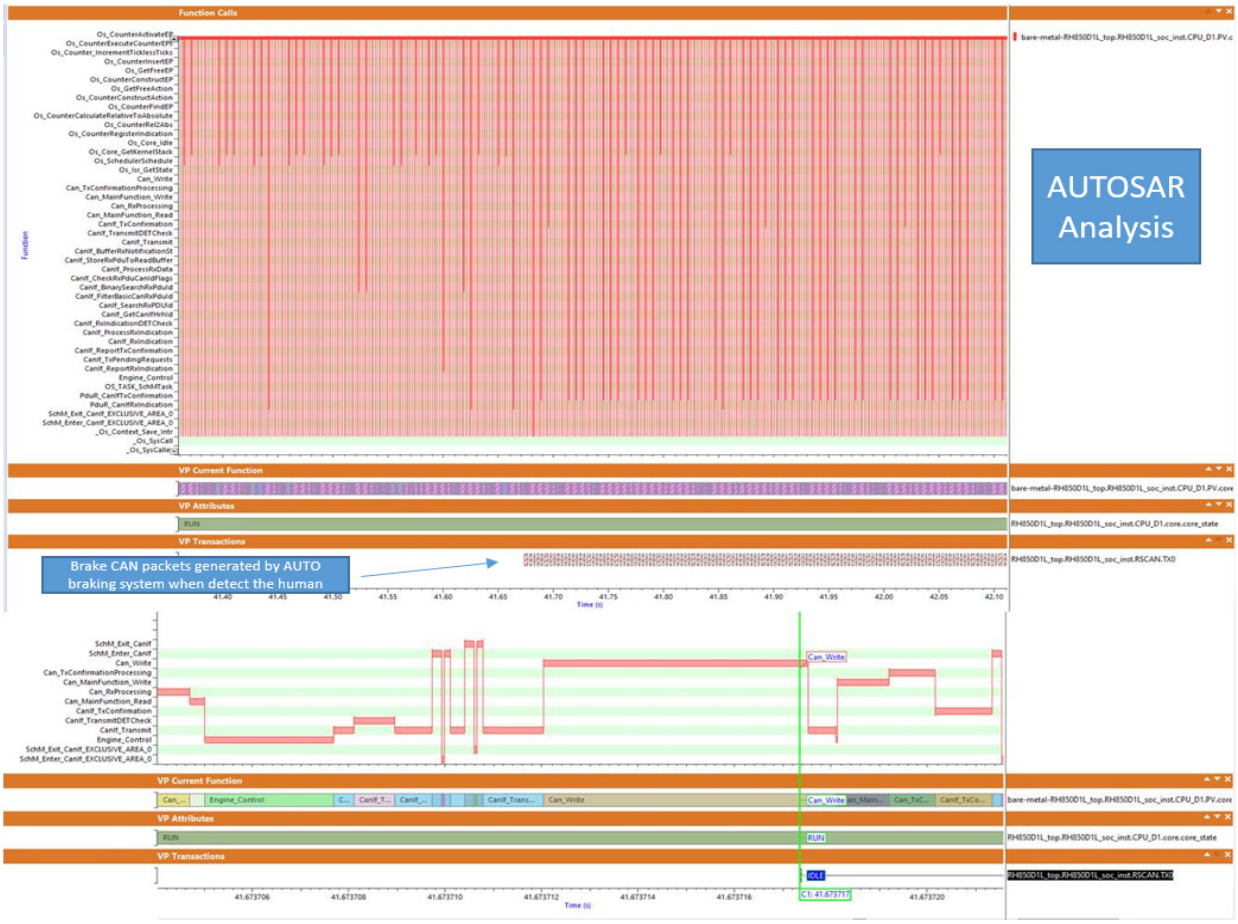

Figure 13. AUTOSAR SW analysis for ADAS VECU

IV. CONCLUSIONS

As shown in the paper, simulating a full heterogeneous automotive system is becoming feasible using virtual platforms for the target Electronic Control Units (VECUs) and the Functional Mock-up Interface (FMI) standard. Not only electrical system but also mechanical system could be simulated and verified using the presented methodology. Virtualization is efficiently used to run AUTOSAR SW on a functional simulation. An Advanced Driver Assistance System (ADAS) including electrical (digital and analog) and mechanical subsystems is modeled and simulated in short time (with high performance). Integrating the heterogeneous parts of a system of systems is becoming essential for automotive system development. In this paper, FMI standard is used to model full SoC platforms as Functional Mockup Units (FMU). FMI allows simulating digital, analog and mechanical parts using an integrated framework. Real software runs efficiently on the modeled system allowing the verification of the system operation. ADAS is validated on the developed virtual system. Debugging the software with the hardware is becoming easier with the presented methodology.

REFERENCES

[1] R. L. Bucs, L. G. Murillo, E. Korotcenko, G. Dugge, R. Leupers, G. Ascheid, A. Ropers, M. Wedler, and A. Hoffmann. "Virtual hardware-in-the-loop co-simulation for multi-domain automotive systems via the functional mock-up interface," in Specification and Design Languages (FDL),p.1-8(2015).

[2] R. Leupers, F. Schirrmeister, G. Martin, T. Kogel, R. Plyaskin, A. Herkersdorf, and M. Vaupel. "Virtual Platforms: Breaking New Grounds," in Design, Automation & Test in Europe Conference & Exhibition (DATE), Germany, p. 685-690(March 2012).

[3] M. Shedeed, G. Bahig, M. W. Elkharashi, and M. Chen, "Functional Design and Verification of Automotive Embedded Software: An Integrated System Verification Flow," *In The Proceedings of The IEEE Saudi International Electronics, Communications and Photonics Conference*, pp. 1-5, July 2013.

[4] R. Saleh, S. Wilton, S. Mirabbasi, A. Hu, M. Greenstreet, G. Lemieux, P. Pande, C. Grecu, and A. Ivanov, "System-on-Chip: Reuse and Integrate," *In The Proceedings of The IEEE*, Vol. 94, No. 6, pp. 1050-1069, June 2006.

[5] R. Leupers, F. Schirrmeister, G. Martin, T. Kogel, R. Plyaskin, A. Herkersdorf, and M. Vaupel, "Virtual Platforms: Breaking New Grounds," *In The Proceedings of The IEEE Design Automation and Test in Europe Conference and Exhibition*, pp. 685-690, March 2012.

[6] C-S. Peng, Li-C. Chang, C-H. Kuo, and B-D. Liu, "Dual-Core Virtual Platform with QEMU and SystemC," *In The Proceedings of The IEEE International Symposium on Next-Generation Electronics*, pp. 69-70, November 2010.

[7] L. B. and S. Sujatha, "Creating Virtual Platform for Cloud Computing," *In The Proceedings of The IEEE International Conference on Computational Intelligence and Computing Research*, pp. 1-4, December 2010.

[8] M. A. El-Moursy, A. Sheirah, M. Safar, and A. Salem, "Efficient Embedded SoC Hardware/Software Codesign using Virtual Platform," *In The Proceedings of The International Design & Test Symposium*, December 2014.

[9] M. Safar, A. Bakr, M. A. El-Moursy, and A. Salem, "AUTOSAR Software Debug and Analysis using Virtual Platforms," *In The Proceedings of The IEEE International Workshop on Automotive Reliability & Test*, December 2016.

[10] C.-C. Wang, R.-P. Wong, J.-W. Lin, C.-H. Chen, System-level development and verification framework for high-performance system accelerator," *In The Proceedings of The International Symposium on VLSI Design, Automation and Test*, pp. 359–36, April 2009.

[11] F. Mendoza, C. Kollner, J. Becker, and K. Muller-Glaser, "An Automated Approach to SystemC/Simulink Co-Simulation," *In The Proceedings of The IEEE International Symposium on Rapid System Prototyping*, pp. 135–141, May 2011.

[12] F. Wawrzik, W. Chipman, J. Molina, and C. Grimm, "Modeling and Simulation of Cyber-Physical Systems with SICYPHOS," *In The Proceedings of The International Conference on Design and Technology of Integrated Systems in Nanoscale Era*, pp. 1–6, April 2015.

[13] AUTOSAR (AUTomotive Open System ARchitecture). Available at: https://www.AUTOSAR.org/

[14] Functional Mock-up Interface (FMI), http://fmi-standard.org

[15] Mentor, ASiemens Business, Vista Architect®, https://www.mentor.com/products/fv/vista.

[16] MathWorks®, MATLAB/Simulink. Available at:https://www.mathworks.com/products/simulink.html

[17] TASS A Siemens Business, PreScan. Available at:https://tass.plm.automation.siemens.com/prescan

[18] Vector CANoe, https://vector.com/vi_canoe_en.html.

[19] Siemens PLM Software, Simcenter Amesim. Available at:https://www.plm.automation.siemens.com/en/products/lms/imagine-lab/amesim/