# Virtual Development of Engine ECU by Modeling Technology

**Haifeng Xu, Yukihide Niimi and Takayuki Ono**

**Abstract** Along with the evolution of vehicle electronic systems from domain-specific control to the integrated control of the entire vehicle system, ECU systems have become increasingly complicated and large-scale. This has made it difficult to set out an optimal architecture of the ECU system efficiently at the early planning stages. As well, the conventional ECU development methodology is also becoming difficult to achieve the increasingly strict requirements for safety design based on multi-ECU systems. Conventionally, optimizing electronic systems requires fabricating many prototypes and evaluating them repeatedly, but with their increasing scale, this method has become impractical. We therefore believe virtual development is a required step. Although functional-level simulators and implementation-level simulators are being used currently, these are both separate and independent. Because of this, it is necessary to introduce virtual development as a new physical-level development environment to connect functional-level and implementation-level. In terms of not only function but also safety design, virtual development has the ability to inject failures that are difficult to recreate in an actual device. Therefore we have started introducing the virtual development of ECU systems by using system level modeling and simulation technology with SystemC language which provides the concept of time. In the phase of physical-level design, because a virtual ECU system is developed by designing each functional model of system such as ADC and drive circuit and connecting these models as a system, the behavior of the whole ECU system can be verified easily without having actual devices. Therefore the optimized structure of ECU, such as microcomputer, software and peripheral LSI, can be determined efficiently at the

F2012-D02-004

H. Xu (✉) · Y. Niimi · T. Ono
DENSO Corporation, Kariya, Japan
e-mail: haifeng@eeda.denso.co.jp

early stages of ECU development. Safety design can also be achieved efficiently because the data transferred in the system can be changed to failure data forcibly by covering functional models with failure models. We believe that maximizing the performance of ECUs in electronic systems, and ensuring that these systems meet safety design requirements will require methods to visualize things that are difficult to visualize, and that this visualization is needed both before and after manufacturing. Virtual development of ECU systems by using system level modeling and simulation technology with SystemC language provides a useful method to achieve these requirements.

**Keywords** Modeling · Simulation · SystemC · ECU · Safety

## 1 Introduction

As part of the push towards a lower-carbon society, electronic control systems for automobiles are developing and evolving from domain-specific control in the vehicle (power train, body, safety, etc.) to the integrated control of the entire vehicle. The ECU, which forms the backbone of such control systems are thus growing in scale and complexity. The development of ECUs in this changing environment requires having an overview of the entire electronic system at the planning stages; this overview would set out an optimized ECU structure in which even the structure of the chipsets are defined; without such an overview, it will be difficult to keep up with vehicle requirements and specifications. As well, the more stringent design requirements for safety that straddle multiple systems are becoming difficult to achieve using the conventional ECU development approach [1].

In response to these issues, DENSO has started introducing the virtual development of electronic systems in order to further leverage our experience developing the logical architecture and physical architecture that support our vehicle product planning, as well as our experience implementing this architecture over the entire vehicle. This paper discusses the virtual development of engine ECUs based on the perspectives described above.

## 2 Issues in ECU Development

Currently, when developing automotive electronic systems, an overview of the entire vehicle is created, and the architecture is developed using logical models with a high level of abstraction to make the structure of the entire vehicle easy to understand, and the functions to distribute to each ECU are decided.

In the next stage, the ECU development phase, the software and hardware allocation is reviewed along with the microcomputer and ASIC configuration, and the ECU implementation specifications are decided. The phase after that is the actual implementation design phase, where implementation-level simulators are used to perform detailed design. However, at this detailed design stage, the various constraints interact in complex ways, and specifications must be repeatedly verified with the constraints in the ECU system specifications and the constraints in the implementation. This is because as the development process moves downstream, the amount of information required increases; however large amounts of information becomes apparent for the first time downstream. In order to keep such coordination to the minimum, it is necessary to determine as much information at the upstream stages and to create a large-scale, detailed verification environment.

## 3 Aims of Virtual Development

Conventionally, implementing the processes mentioned above requires fabricating large numbers of prototypes and evaluating them repeatedly, but with their increasing scale, this method of optimization has become impractical. Because of this, we have introduced virtual development as a new physical-level development environment to connect the logical and implementation domains. In terms of not only function but also safety design, in which failsafe specifications are considered to ensure the completeness of failsafes for different malfunctions, one of the elements that is required of virtual development is the ability to inject failures that are difficult to recreate in an actual device. Thus, it was decided to use modeling technology based on SystemC, a language that can be used to rapidly run system level simulations while having the notion of time, in order to create the virtual development environment. This ECU modeling technology is described in more detail below.

## 4 ECU Modeling Technology

### *4.1 ECU Modeling Concepts*

Based on the issues described above, the virtual development should be applied to meet the following requirements.

(1) Optimizing the allocation of hardware and software.
(2) Reviewing the configuration of microcomputers and ASICs.
(3) Performing failure simulations.

Though the models that would be used to meet requirements 1–3 may need different levels of abstraction [2], we believe that it is possible to come close to connecting these models with differing levels of abstraction in what is practically a

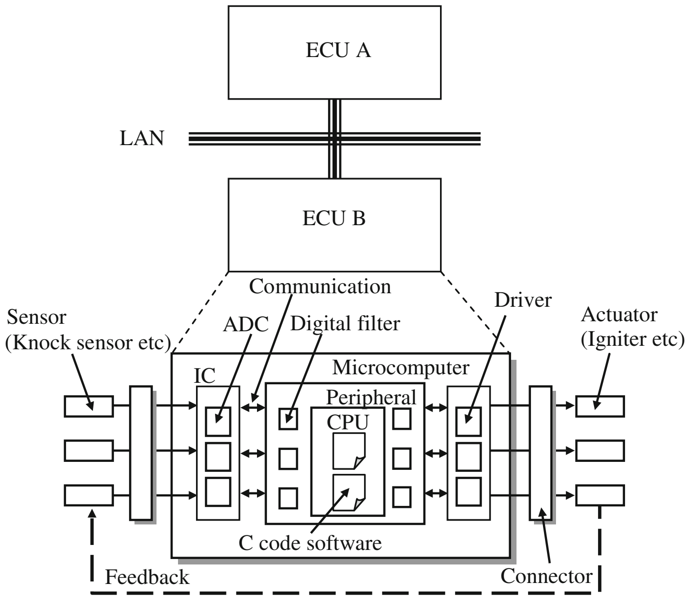


**Fig. 1** Schematics of system model

single virtual environment. As shown in Fig. 1, building a virtual ECU environment by creating and connecting all the necessary models of LAN, various sensors, functional blocks in control units (ECUs), and actuators being controlled will allow us to verify the feasibility of the operation of the entire system and to review the suitability of software and hardware structure and microcomputer performance.

## *4.2 Component Modeling*

Based on the existing system structure, we modeled each functional block including the AD converters in the ICs as well as the microcomputer peripherals such as the drive circuits and digital filters. This allows not only the overall activity to be observed but also the detailed behavior of each block. The microcomputer manufacturer provided the model of the microcomputer core which is a cycle-accurate ISS model, and this model was connected to the other models. By doing so, Requirement 1 (layout and review of hardware and software) and Requirement 2 (verifying microcomputer performance by calculating the CPU processing load and RAM/ ROM usage) are satisfied. However, because having everything at a detailed level of abstraction results in the disadvantage of increased simulation time, the behavior in the models is investigated making strategic use of transaction level and pin level interfaces between models to adjust the abstraction based on whether or not a block is under detailed review [3]. This allows the total number of runtime events in the

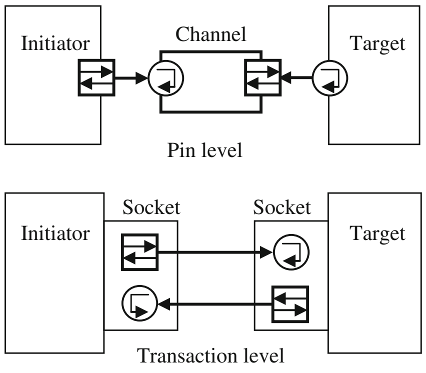


**Fig. 2** Model interface

simulation to be decreased in order to create an environment in which large-scale systems can be run at high speeds. Figure 2 shows two kinds of model interfaces.

Figure 3 shows use of transaction level interface and pin level interface.

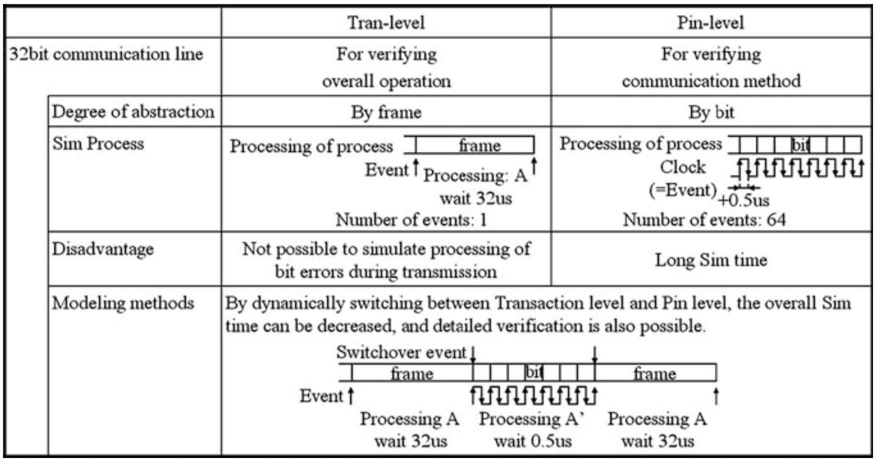
When it considers how to model 32-bit communication line, transaction level interface is for verifying overall operation, and pin level interface is for verifying communication method. These are different abstractions. The degree of abstraction is frame-based for the first type and bit-based for the second type, and the simulation process has a single event for the first type and 64 events for the second type. The disadvantage for the first type is that bit errors during transmission cannot be simulated, and for the second is that the simulation takes too long. It was for these reasons that we developed a modeling method that maintained the advantages of both and resolved their disadvantages. The method is to add a switching event between transaction level and pin level modeling so as to enable dynamic switching. This can reduce the overall simulation time while still allowing performing detailed verifications.

## 4.3 Failure Modeling

To inject failures, the failure modes were first analyzed. The results of this analysis reveals that failures can occur in various locations including physical connections and gates inside ICs, but all of these failures can be classified into a few modes such as disconnection, locking, corruption, drift, and oscillation.

Another issue is where and how to inject these failures. Because the locations where failures can be observed in an actual device are at its various terminals, a failure model is laid over the functional model as shown in Fig. 4, and the failures are defined in the output (a GND short failure is shown), which forces the system to treat the data transferred as abnormal values; this simplifies the failure model and makes failure injection easier, all without making any changes to the functional model.

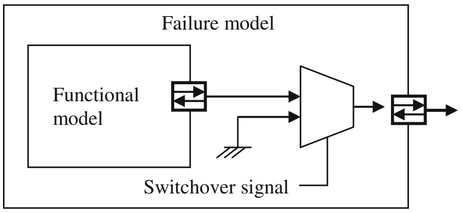
The final issue is the timing of the failure. The failure model added above is given a failure changeover signal as an input with the value and time of occurrence



| | | Tran-level | Pin-level |
|---|---|---|---|
| 32bit communication line | | For verifying overall operation | For verifying communication method |
| | Degree of abstraction | By frame | By bit |
| | Sim Process | Processing of process frame<br>Event Processing: A<br>wait 32us<br>Number of events: 1 | Processing of process bit<br>Clock (=Event) +0.5us<br>Number of events: 64 |
| | Disadvantage | Not possible to simulate processing of bit errors during transmission | Long Sim time |
| | Modeling methods | By dynamically switching between Transaction level and Pin level, the overall Sim time can be decreased, and detailed verification is also possible.<br>Switchover event<br>frame bit frame<br>Event<br>Processing A wait 32us Processing A' wait 0.5us Processing A wait 32us | |

Fig. 3 Use of different interface

Fig. 4 Failure model



set in the initial settings; because such failures can be analyzed in the same way as a regular simulation, it is easy to express not only steady-state failures but also transient failures, and we are able to use this method to verify safety design.

However, when performing a bit corruption failure in the communication data during transmission between models, in order for the failure model to inject a malfunction with respect to particular bits, the functional model transmits data one bit at a time and this results in the disadvantage of increased simulation time [1]. Therefore, an effective modeling method that can be used to freely inject bit corruption malfunctions during transmission even with frame-based communications is explained below.

Specifically, the framed-based data that the functional model tries to transmit is outputted at the transmission start time, and the failure model saves the outputted data until the transmission end time. If a bit corruption failure occurs at the given time during saving, the abnormal data is calculated based on the time that the failure occurs, and the saved transmission data is substituted with the abnormal data, and at the transmission end time, the saved abnormal data is transmitted. In this way, it is now possible to shorten the simulation time and perform detailed verifications at the bit level.

## 5 Application Examples

Two examples are shown below to explain the application of virtual development. Example 1 covers Requirement 1 (optimizing the allocation of hardware and software) and Requirement 3 (failure simulations). Example 2 covers Requirement 2 (reviewing the configuration of microcomputers and ASICs).

### *5.1 Example 1*

During the early stages of ECU development with conventional development methods, which have no actual devices, CPU processing loads cannot be verified, so it is difficult to verify the suitability of allocation of hardware and software. The introduction of virtual development technology is an efficient way to solve this problem.

This example is the development of the ECU for a 4-cylinder engine. At the ECU implementation specification review stage, the virtual ECU is built based on the CPU model. At this point, the allocation of hardware and software has not been decided, so a temporary allocation based on design experience may be devised. If an allocation cannot be devised, a software implementation may be used.

Next, the operation of the virtual ECU system is verified. In terms of the control software, the statistical data for each task and function call can be obtained, which allows the CPU processing load across the range of engine speeds to be analyzed as shown in Fig. 5. If there is some extra CPU processing capacity, some of the hardware-based processing can be transferred to the software, and if there are any high-load tasks, the corresponding processing can be transferred from the software to the hardware; all of this information is useful when reviewing the system. When doing so, as shown in Fig. 6, the feasibility of the operation of the entire system across the range of engine speeds and the timing of operations can be observed, and the performance of the portions that have been made hardware can be verified as well.

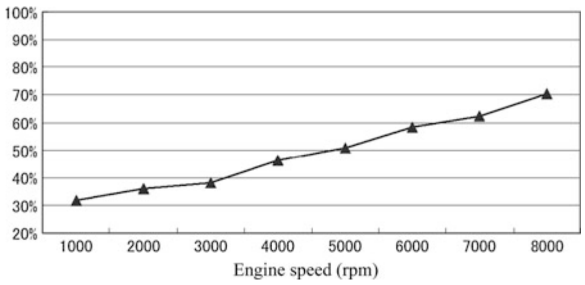
Next, Fig. 7 shows the injection of a failure into the actuator drive circuit as well as the results of the system failsafe analysis.

Here, after an over current failure was injected into the drive circuit, the power to the actuator was cut off. The specification called for 600 ms max., and the power was cut in 500 ms, so the effectiveness of the system failsafe was confirmed.

### *5.2 Example 2*

At the early stages of ECU development, all of the possible ECU implementation possibilities need to be laid out, so that they can be optimized and to review their suitability in terms of mass production and cost. Because of this, the different

**Fig. 5** CPU processing load



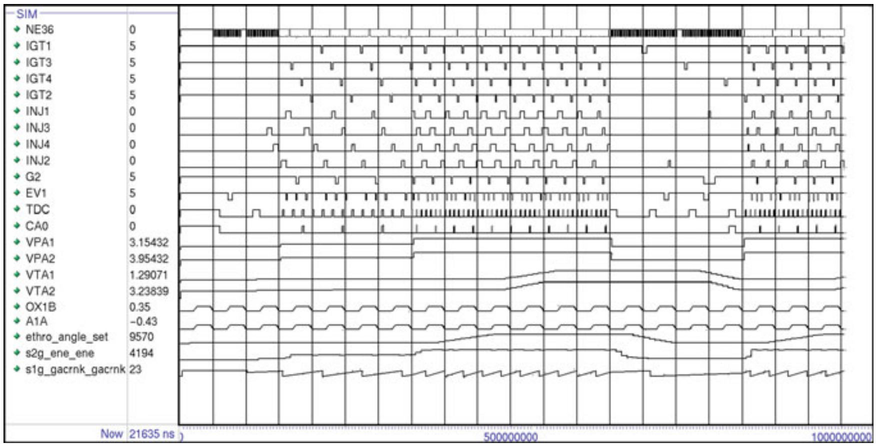
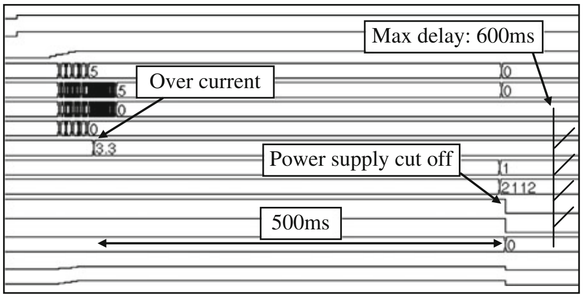


**Fig. 6** Analysis of engine ECU

**Fig. 7** Analysis of failure model



configurations of microcomputers and ASICs need to be reviewed. For example, when reviewing the configuration of microcomputers and ASICs, one option is to use a custom microcomputer with a wide range of functions and to use a general-purpose IC. Another option is to use a general-purpose microcomputer and build custom functions into an IC. By using virtual development technology, it is possible to verify whether each configuration option satisfies the performance requirements.

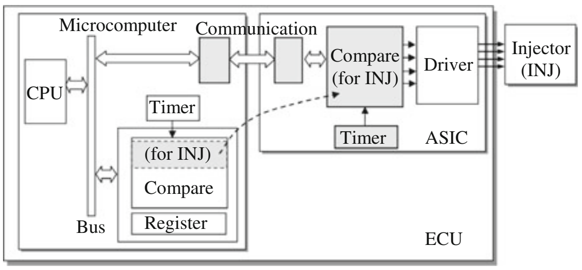


**Fig. 8** Possible configuration of microcomputer and ASIC

Explained below is an example of using a general-purpose microcomputer with few compare channels to build into the ASIC the custom microcomputer's compare function for driving the injectors.

When a compare function is built into the ASIC, as shown in Fig. 8, specifications for communication between the CPU and compare function, implementation specifications for the timer required for the compare function to operate, and software specifications required for communication must be considered. Also, the communication performance and CPU processing load must be verified to satisfy the ECU performance requirements. The review procedure with virtual development technology is outlined below.

In Step 1, the implementation specification options are reviewed and the models are built. A specific example of a specification is this: The CPU uses the existing CSI communication channel in a general-purpose microcomputer to read the timer value stored in the ASIC, calculates the expected injector operation time and injector number, and sends the result to the ASIC; the compare function determines the injector operation timing based on the received data. Such detailed specification options are reviewed in this step to build the required models and create the virtual ECU.

In Step 2, the operation is verified. Examples of what is verified are the feasibility of the operation, accuracy of the operation (injection start timing and duration), CPU processing load, and RAM/ROM usage. In terms of methods to verify operation, the operating frequency for CSI communication is adjusted to verify operation across the range of engine speeds. For example, Fig. 9 shows the delay in injection start time when the CSI communication operation frequency is varied from 4, 2, to 1 MHz in accordance with the specifications of the general-purpose microcomputer. At frequencies of 2 MHz and higher, the delay time is within the allowable range defined in the ECU performance requirements. At 1 MHz, the communication delay is too long, and the injection operations at high engine speeds are abnormal. The injection duration and the CPU processing load and RAM/ROM usage can be verified in the same way, so the suitability of microcomputer performance can be evaluated.

In Step 3, the results are fed back into the implementation specification options. Based on the results of the operation verification, it is judged if the ECU performance requirements are satisfied. If necessary, the implementation specification options are improved through feedback, and the operation is verified again.

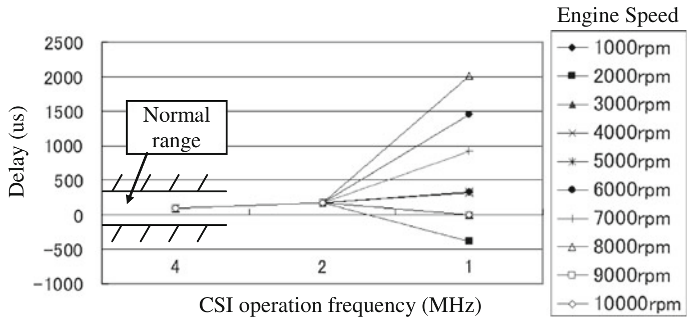


**Fig. 9** Analysis of injection delay

In the ways described above, the introduction of virtual development technology allows verifying various implementation specification options that are difficult to verify in actual devices and deriving the optimal solution efficiently.

## 6 Outlook

The role of electronic systems in the move toward a lower-carbon society is expected to grow increasingly important. We believe that maximizing the performance of ECUs in electronic systems, which continue to grow in scale and complexity, and ensuring that these systems meet safety design requirements will require methods to visualize things that are difficult to visualize, and that this visualization is needed both before and after manufacturing. We would like to use the modeling technology described in this paper as a base for creating a virtual development environment and to carry out the development of vehicle electronic systems and products that contribute to society.

## References

1. Niimi Y, Ono T, Tsuchiya N (2012) Virtual development of engine ECU by modeling technology. [J] SAE technical paper, 2012-01-0007:1–5
2. Bailey B, Martin G, Piziali A (2007) ESL design and verification [M]. Morgan Kaufmann, USA
3. STARC (2008) TL modeling guide, 2nd edn [M] Semiconductor technology academic research center, Japan