

Autocoding of Power Plant Control Software

Shintaro MINOWA*

Toshiaki HIROTA*

Hiroki YOKOYAMA*

Tetsuya ISHIGURO*

Satoru OHTSUKA*

ABSTRACT

An autocoding system has been developed for use in CAD of control systems. This system conducts simulations of fixed point calculations equivalent to those used in ECU implementation, and autocodes the designed control specifications. The use of this system enables simple formulation of the data accuracy (LSB) and data type required for software coding. In addition to coding software in a Honda format, the system autocodes the label information required in setting parameters and monitoring for vehicle tests. As a result, the autocoded software can be employed in ECU implementation without alteration. Verification of the accuracy of calculations using a coverage check enables ECU malfunctions to be prevented before implementation. This has reduced the time spent in correcting the software, and autocoding has shortened the period to ECU implementation by one-third. A level of performance equivalent to the simulation has been achieved with no malfunctions in a single implementation.

1. Introduction

Power plants require complex, large scale and high accuracy control such as environmental compliance, fuel economy enhancement, and on-board diagnosis. However, shortening the development period, increasing the number of development models and otherwise increasing development efficiency are also important issues. Therefore, the use of software libraries has begun in order to ensure power train ECU software quality and increase the development efficiency. This has separated the portions dependent on the hardware from the application portions, and allows the development of applications by combining existing software.

Applications are designed using simulation tools (MATLAB, SimuLink), and the established control logic was formerly compiled into control specification sheets and then manually coded in the Honda format. However, this process had the following concerns.

- (1) Discordant design aims due to manual coding
- (2) Conflict between simulations and actual calculations during implementation

As a means of realizing a method that does not produce these conflicts, a system was developed for performing the calculations used for implementation in the Honda format on the simulation tools, and then autocoding these results.

2. System Development Goals

The goals of software autocoding are as follows.

- (1) Eliminating conflicts in logic interpretation between logic designers and programmers
- (2) Reducing software description errors
- (3) Reducing the development time by automating the work

2.1. Shortening the Development Process

The ECU performs fixed point calculations instead of the more commonly used floating point calculations in consideration of processing speed and memory capacity, etc. Figure 1 shows a comparison of floating point and fixed point wave forms. Fixed point calculations differ from floating point calculations in that the parameter size and LSB

* Tochigi R&D Center

must be set properly in order to obtain the required calculation accuracy. In the conventional development process, software coding (manual coding) is performed for the model diagram after the floating point operation check. This meant that fixed point calculations were first performed after implementation, and the software had to be revised in order to correct any operation discrepancies found. The control logic design tools used with the newly developed system are a control design CAD (SimuLink)⁽¹⁾ and a general-purpose integer type autocoding tool (TargetLink)⁽²⁾. These tools perform fixed point calculations on the model diagram and automatically code the software for that logic. Applying this system makes it possible to complete the series of correction processes on a PC, so a reduction in the ECU development time can be expected. Figure 2 compares the conventional ECU development process with the process when applying the newly developed system.

2.2. Development Environment without a Correction Process

The general-purpose integer type autocoding tool has an optimum software coding function suited to ANSI-C and Target (ECU chip). However, this differs from the Honda format, so manual corrections are necessary for software implementation. The label name, LSB, data type and other parameter information in the sensor outputs and other parameters normally used by Honda are registered and managed in an original library as shared information. For this reason, the following functions were created as a system linked to this library and as a development environment that does not produce correction time from logic construction to software coding.

- (1) A function for acquiring label information from the library and reflecting it to the model
- (2) A function for describing the label form for fixed point calculations in the Honda format

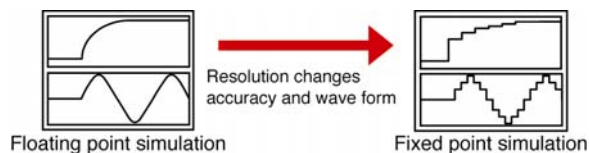


Fig. 1 Comparison of floating point and fixed point wave form

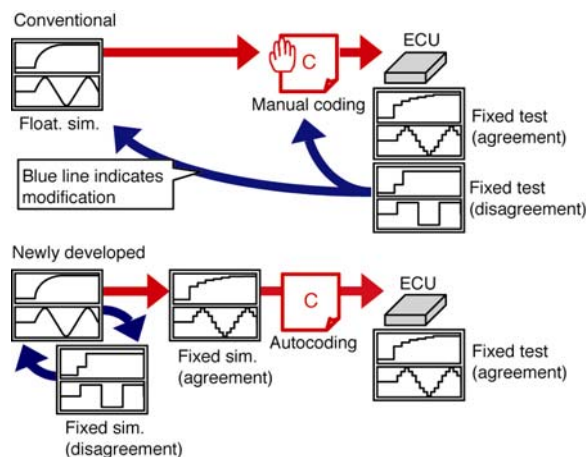


Fig. 2 Comparison of ECU design processes

- (3) A function for checking operation through a coverage check
- (4) An autocoding function that conforms to the Honda format
- (5) A function for generating documents to take the place of conventional specification sheets

Figure 3 shows an outline of the newly developed system. This system eliminates the need for manual corrections during fixed point calculations and implementation of the autocoded software, etc.

3. Adapting Model Diagrams to Honda Format

The functions of the newly developed system are described below.

3.1. Label Information Setting Function

The library registers not only the label information, but also the control logic specification sheets that use those labels and the software that is based on these specification sheets. Therefore, when creating new control logic, the registered control logic and software can be reused in accordance with the application. The library centrally manages and shares the label information used in the logic as CSV files, and the same label name is used for labels that handle the same information, even if the logic differs. This makes it possible to set the LSB, data type and other parameters without having to set the label information for each logic. In addition, when the original library label information is changed (renewed), these changes are reflected to all the logic that uses that label. The following functions were created to match these settings with the label information settings made using the newly developed system.

- (1) A function for referencing the library label information when creating model diagrams
- (2) A function for renewing model label information when the library label information is renewed
- (3) A function for creating new label information and registering it in the library

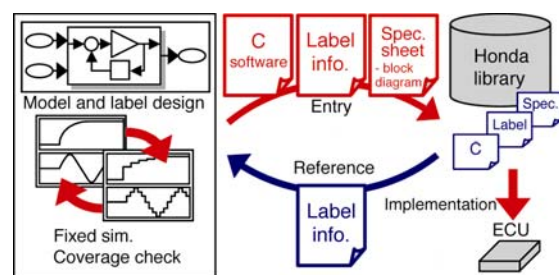


Fig. 3 Newly developed process

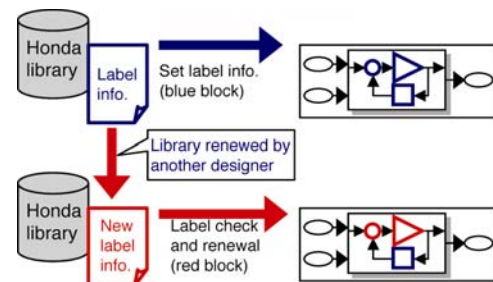


Fig. 4 Label information check and renewal tool

This makes it possible to automatically set the LSB and other label information for existing labels even when creating new models, and allows auto renewal of that label information to the latest library information. In addition, the label information auto renewal function can also be used to avoid the risk that newly created labels in a model might overlap with existing label information. Figure 4 shows this function.

3.2. Auto Switching of Data Classes

In fixed point calculations using the general-purpose integer type autocoding tool, the labels used by the logic must be defined within the logic. However, in the actual logic, sensor signal values and other parameters are sometimes provided by extern labels, which make it necessary to change the data class declaration each time the calculations are performed. Therefore, a label type was created that automatically switches the data class declaration according to the autocoding setting, with global labels for simulations and extern labels for autocoding. This made it possible to perform fixed point calculations without changing the data class used in software coding. Figure 5 shows an outline of the data class auto switching function.

3.3. Replacement of Map Search Routines

Normally when using maps and tables, the blocks provided for the general-purpose integer type autocoding tool are used. However, Honda defines and uses original map search routines, so the general-purpose integer type autocoding tool blocks cannot be used as is. Therefore, a function was created for replacing the search routines and outputting in the Honda format during autocoding. The search routines are original, but the blocks used for model creation use the same blocks as the general-purpose integer type autocoding tool, so replacement can be performed without any awareness on the part of control logic designers. The search routines are replaced from the standard search routines not only during software coding, but also from the control logic investigation calculations to ensure uniformity of operation with implementation.

3.4. Stipulation of Data Types

In the same manner that general labels are shared as a library, the data types used by Honda are also stipulated. The general-purpose integer type autocoding tool has an extremely large number of standard data types to ensure generality, so human error may occur when selecting the type. Therefore, the newly developed system uses a format where only Honda original data types can be selected. In addition, type information was added to the label name to allow users to know the data type being used. This work was

formerly done manually, but automatically adding this information during the data type selection reduces errors and increases efficiency of the work.

3.5. Initialize Statement

When constructing the control logic, it is sometimes necessary to initialize the labels in specific operation modes such as engine start or stop. However, each company has its own initialization method and there is no standardization, so this function is not provided in the general-purpose integer type autocoding tool. Therefore, an initialize statement block was created using the existing label definition block, and a function was created for automatically adding initialize logic during source coding as a C function in accordance with the library rules.

3.6. Inclusion of Existing Software

When constructing the control logic, logic may be developed which uses the output values of existing software as the inputs. Existing software may not have a model diagram, and may exist only as manually coded software. In these cases it is necessary to understand the software contents and create a model diagram in order to use this software in logic development. Therefore, a function was created for including software without edit as a block within the model diagram. Figure 6 shows an outline of this function. The included software follows the coding rules stipulated by the library, so the block inputs and outputs are generated automatically and the software is included as a model diagram block. In addition, linking with the data file allows operation to set the coefficient setting values of the included software.

3.7. Model Library

Some block structures such as counters and switches are used frequently when constructing model diagrams, and thus far designers had created model diagrams of these blocks each time they were used. Therefore, these blocks were created in advance, converted to subsystems, and used as a

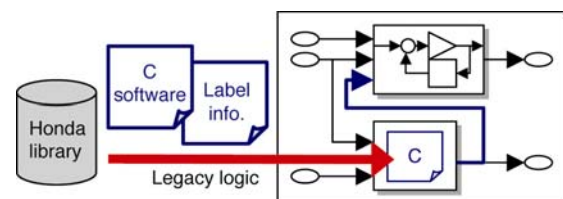


Fig. 6 Embedded function of legacy C software to new system

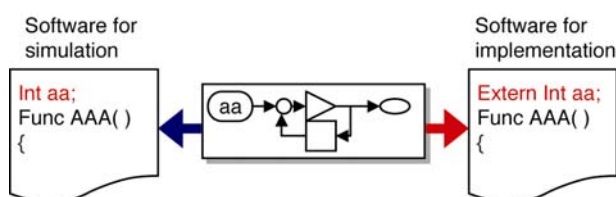


Fig. 5 Auto switch of C software generation

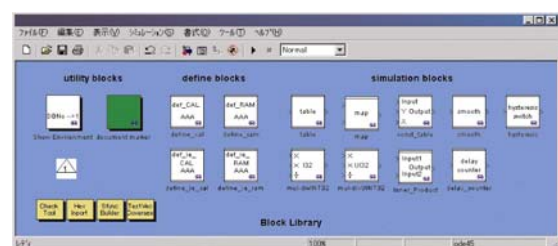


Fig. 7 New system model library

model library. Figure 7 shows the model library. This reduced the modeling time, eliminated the need to check the operation of portions registered in the model library, and allowed standardization of model diagram descriptions. In addition, the functions created using the new system can be easily called up from the library.

4. Validation Check

Operation is checked by preparing optional input data. In most cases, measurement data obtained from vehicle tests is generally used as the input data. In addition, autocoded software is basically not checked manually, and the LSB settings are also made automatically except for blocks where succession is performed manually. This naturally increases the importance of the input data used when making the settings. That is to say, if the data input and output values during actual use exceed the input and output values used to make the LSB settings, or if combinations of these values are not tested, this may cause issues because overflow or other phenomena could not be checked until vehicle tests or after the product is introduced to the market. Likewise, if all branches generated by operation switching of “if” branches or flags within the software are not checked, it is possible that logic overflow up to those branches will not be checked, and also that the validity or even the need for the logic will not be checked. Therefore, the following functions were created to increase the software reliability.

- (1) A function for generating input data (test vectors) with optional upper and lower limit values
- (2) A function for investigating branch coverage
- (3) A function for forcibly switching branches that have not been executed

4.1. Test Vector Generation Function

The test vector generation function was created to generate inputs for the coverage check function. This function generates two-level data that allows the amplitude to be set and changed freely. There are two types of wave form shapes: an all pattern and design of experiment (DOE)⁽³⁾ type that is expressed by a combination of the two-level data, and a pulse width modulation (PWM) type that expresses this same data as a PWM waveform. Figure 8 shows the generated wave forms.

4.1.1. Wave form types

The all pattern input consists of all patterns for two levels, where the number of input ports n for the target control logic is used as the number of factors, and covers all combinations for the case where the input data is thought of as two levels. However, this means that as the number of input ports increases, the generated input data volume also increases significantly. Therefore, DOE was included to

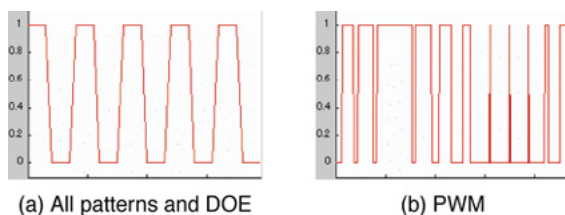


Fig. 8 Shape of test vector

allow the construction of data signifying an experimental amount equivalent to all patterns even with a small amount of input data. However, the selection of orthogonal factors is left to the designer constructing the logic, and the orthogonal Latin squares, that is to say the input data bases, are created using a function provided for the control design CAD.

The PWM wave form generates two-level data by pulse width modulation based on the fixed values of the frequency and the frequency variation for each input port. The generated wave form takes a shape where the output cycle of the wave form for other ports repeats in accordance with the amount of data for the longest cycle among the PWM waveforms for each input port.

4.1.2. Repeating wave form amplitude

In logic that performs the high order filtering and multi delay data hold used in modern control, no matter how the data is handled, the data is rounded at the filter output stage, and cannot be used for subsequent logic checks. In addition, there are also fundamental concerns such that as the order increases, the amount of data becomes insufficient. Therefore, a function was created for amplifying the input data generated by repeating the above wave forms. Figure 9 shows the test vector wave form for 5-time repeat mode. The amplitude when repeating can be selected from the following three types.

- (1) Optionally set maximum and minimum values for each input port (Fig. 9 (a))
- (2) Changing the maximum and minimum values while maintaining a fixed amplitude in accordance with the number of repeat times (Fig. 9 (b))
- (3) Changing the set maximum and minimum values in accordance with the number of repeat times and gradually reducing the amplitude (Fig. 9 (c))

This makes it possible to check the LSB toughness by detecting overflow, and also to check performance near values that are important for actual logic operation.

The test vector generation function generates input data according to the number of input ports for the model. However, the specifications make it possible to further reduce the amount of input data by excluding the corresponding input ports from the factor combinations when values that have no effect on the logic are present in the model configuration or when two-level values exist in the flag information, etc.

4.2. Coverage Check Function

The coverage check results are generated automatically by the control design CAD when the test vector simulation ends. At this time if the process did not pass through all the branches within the logic, the coverage results are displayed as less than 100%. Therefore, a function was created to display the pertinent locations in the model and to optionally

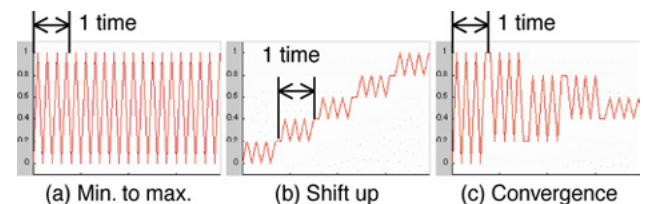


Fig. 9 Shape of test vector (5-time-repeat mode)

original expanded guidelines were created for using the general-purpose integer type autocoding tool, and diagrams and documents are generated based on these guidelines.

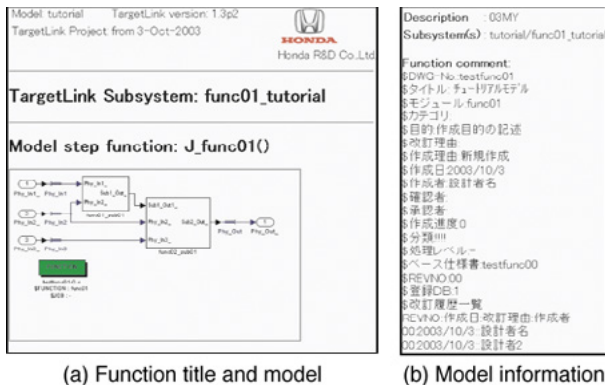


Fig. 11 Spec. sheet produced by auto document generation

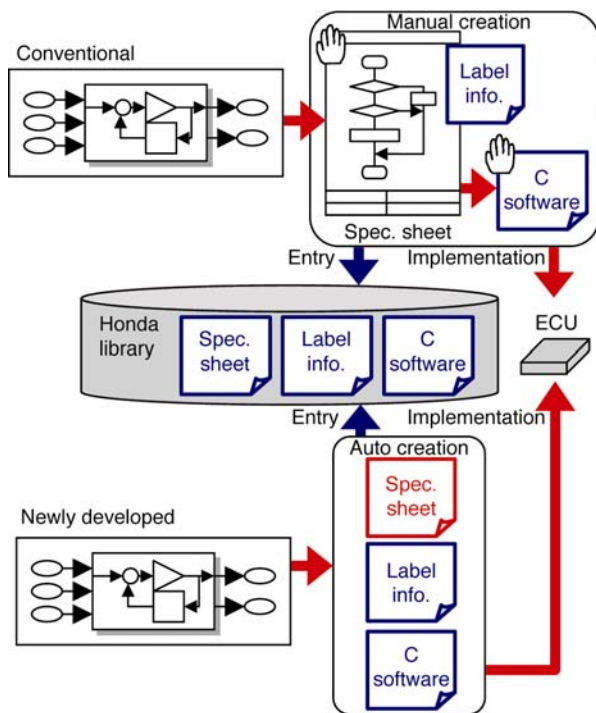


Fig. 12 Comparison of flows of model data up-load

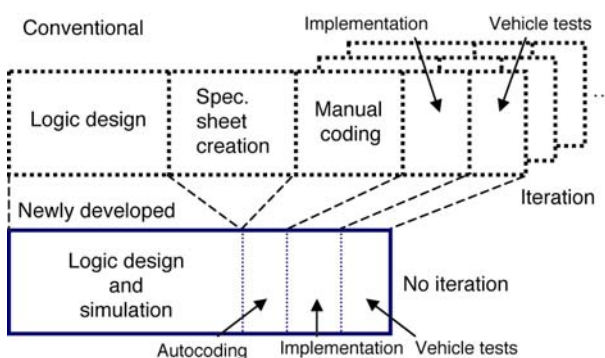


Fig. 13 Reduction of flow of design logic and load to ECU

Figure 12 shows a comparison of library registration flows. Auto document generation has further shortened the process time, and allows automatic output of all the information registered in the library. This has reduced the period from logic construction to ECU implementation by one-third compared to the conventional method. Figure 13 compares a breakdown of ECU implementation periods.

7. Conclusion

Development of this new system produced the following results.

- (1) ECU implementation is possible without edits to software autocoded from model diagrams.
- (2) A level of performance equivalent to the simulation has been achieved in a single implementation.
- (3) Time formerly spent in correcting the software following implementation has been eliminated, thus shortening the period from logic development to ECU implementation by one-third.

References

- (1) The MathWorks, Inc.: Using Simulink Version 5, USA, The MathWorks, Inc., p. 476 (2002)
- (2) dSPACE GmbH: TargetLink Production Code Generation Guide for TargetLink 1.3, Germany, dSPACE GmbH, p. 748 (2001)
- (3) Nakamura, G.: Understanding Design of Experiment, Tokyo, Kindaigakusya, p. 82-108 (1997)

Authors



Shintaro MINOWA



Toshiaki HIROTA



Hiroki YOKOYAMA



Tetsuya ISHIGURO



Satoru OHTSUKA