International Journal of Automotive Technology, Vol. 7, No. 3, pp. 345−351 (2006)

INTEGRATED DEVELOPMENT ENVIRONMENT FROM MODELING TO IMPLEMENTATION FOR AUTOMOTIVE REAL-TIME EMBEDDED CONTROL SYSTEMS

J. MA[1)], J. YOUN[1)], M. SHIN[1)], I. HWANG[1)] and M. SUNWOO[2)*]

[1)]Department of Automotive Engineering, Graduate School, Hanyang University, Seoul 133-791, Korea
[2)]Department of Automotive Engineering, Hanyang University, Seoul 133-791, Korea

(Received 25 October 2005; Revised 10 February 2006)

ABSTRACT−Software-In-the-Loop Simulation (SILS) and Rapid Control Prototyping (RCP) are proposed as an integrated development environment to support the development process from system design to implementation. SILS is an environment used to simulate control systems with temporal behavior. RCP offers seamless phase shift from design to implementation based on automatic code generation. There are several toolsets that support control system design and analysis. A few of these tools generate the control software automatically. However, most of these design toolsets do not cover temporal behavior which appears after implementation. In earlier toolsets, the design and the implementation of a control system are considered as two separate processes which mean the conventional development process is not connected strictly. SILS/RCP environments work under an identical platform and use the same representation for system modeling. An integrated SILS/RCP environment makes it possible to design controllers under conditions similar to real execution during off-line simulation and to realize controllers in the early design phase. SILS/RCP environments integrate the design and implementation phases which reduce the time-to-market and provide greater performance-assured design. The establishment of SILS/RCP and the practical design approaches are presented.

KEY WORDS : SILS (Software-in-the-Loop Simulation), RCP (Rapid Control Prototyping), Co-simulation, Delay effect, CACSD (Computer Aided Control System Design)

1. INTRODUCTION

Embedded control systems generally have stringent real-time requirements. The controllers have to satisfy not only control specifications, but also temporal requirements. In other words, they should ensure functional correctness (the correct result is produced) and temporal correctness (the result is produced at the correct time) (Gu *et al.*, 2004; Kopetz, 1997). However, most Computer Aided Control System Design (CACSD) tools do not support simulation of the temporal behavior of control systems.

Control systems are normally implemented using microcontroller units (MCUs). In many cases, computer based control theories assume equidistant sampling intervals and negligible or constant control delays which are the latency between the sampling of inputs to the controller and the generation of outputs (Henriksson *et al.*, 2002). These assumptions are not realistic and can not be achieved in practice. Therefore, as the undeterministic delays increase, the predictability of control systems decreases. The undeterministic temporal behavior may reduce the system stability. As a result, the demands for simulation environments considering temporal behavior have increased.

There were two approaches for the design of CACSD tools. One is a control engineering perspective. The other is a software and computer engineering perspective. From control engineering perspective, tools are designed in order to support functional simulation and mathematical analysis. A few of these tools can automatically generate software to implement specified control algorithms. From a software and computer engineering perspective, a number of tools are developed to perform mathematical analysis of temporal behavior. A few of these tools have been developed to support automatic software assembly that merges the system codes with a real-time kernel or a real-time operating system (RTOS). In the 1990's, an effort to integrate the two different environments began. ControlH and MetaH were introduced by Steve Vestal at Honeywell Technology Center. (Vestal, 1994) ControlH was a language designed to support dynamical system modeling and control algorithm specification (Englehart and Jackson, 1994). MetaH

*Corresponding author. e-mail: msunwoo@hanyang.ac.kr was designed to support specification and analysis of real-time, secure, fault-tolerant, and multi-processor computer system architectures (Mitchell and Gauthier, 1986). This toolset offered the analysis of functional and temporal behavior and automatic code generation. However, ControlH and MetaH used two different platforms, so that control specifications and real-time specifications need to be described by each language in different environments. In order to design control systems using these environments, users needed to be familiar with each language and environment.

There have been several attempts to overcome the problem using different platforms for the system analysis including temporal behavior. They are RT/CS Co-design (Eker and Cervin, 1999), DRTSS (Storch and Liu, 1996), STRESS (Audsley *et al.*, 1994), HaRTS (Zhu, 1994), AIRES (Gu *et al.*, 2004), and TRUETIME (Henriksson *et al.*, 2002; Cervin *et al.*, 2003), which make it possible to simulate the control algorithm in consideration of real-time characteristics in the same platform. Other studies have focused on automatic implementation combining control algorithms with a real-time kernel or a RTOS. These include RT-UML (UML Notation Guide, 1997), AIDA (Törngren and Redell, 2000), MIRCOS (Rebeschieß, 1999), and Giotto (Henzinger *et al.*, 2003), which support the real-time control system analysis and implementation. Nevertheless, in the previous studies, the design and the implementation do not work under the identical platform.

In this paper, Software-In-the-Loop Simulation (SILS) and Rapid Control Prototyping (RCP) are proposed as a simulation environment and an implementation environment. SILS and RCP integrate separated environments into seamlessly connected development environments based on one platform. The simulation result of SILS holds temporal behavior, so that the result is affected by the functional behavior of the control logic as well as the temporal behavior according to real-time properties. The control system designed by SILS environment can be immediately transformed into executable binary code through the proposed RCP environment.

In section 2, the concept and the implementation of SILS/RCP environments are described. In section 3, the feasibility and the effectiveness is considered with a case study.

2. SILS/RCP TOOLBOX

2.1. The Concept of SILS/RCP

The design process of the automotive Electronic Control Unit (ECU) requires multidisciplinary approaches which are related to the control engineering side and the software engineering side. The traditional development process can be represented as a V-shape model (Henzinger

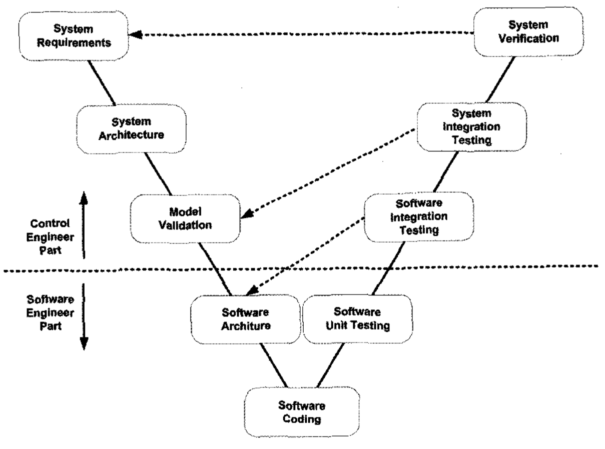


Figure 1. System development V.

et al., 2003).

In the system V development diagram shown in Figure 1 (Toeppe *et al.*, 1999), the whole process is clearly divided into two parts which are control engineering and software engineering. Control engineers are in charge of acquiring system requirements, analyzing system architecture, and setting up the control logic. Software engineers play a role in forming the control algorithms into the software that will be embedded in MCUs. However, there are potential problems caused by unrealistic assumptions, differences between continuous and discrete time-domain, unconsidered real-time properties, and so on. As the result of that, the controller performance may not be consistent with the simulation results, so that the design process must be iterative. In addition, software engineers can not apply the changes to the realization until control engineers finish modifying the algorithms completely.

In order to reduce the time-to-market and support more performance-assured controller design, SILS/RCP environments are developed. SILS/RCP environments work under Matlab/Simulink® which is widely used in the CACSD toolset. SILS/RCP environments have several blocks that consider the temporal behavior of control systems such as the real-time kernel, the task representation, and the network induced effect. The blocks for the real-time kernel describe the scheduler's behaviors. The task is used for the scheduling unit, and time delays of tasks are also considered for realistic simulation results. SILS/RCP environments support the simulation and realization of network based control systems for Controller Area Network (CAN) and Local Interconnect Network (LIN) which are most popularly applied to the automobile.

In SILS, control engineers are able to achieve a simulation that is more similar to the execution on MCUs from the early design phase. RCP makes it easy to implement the control logic so that control engineers can

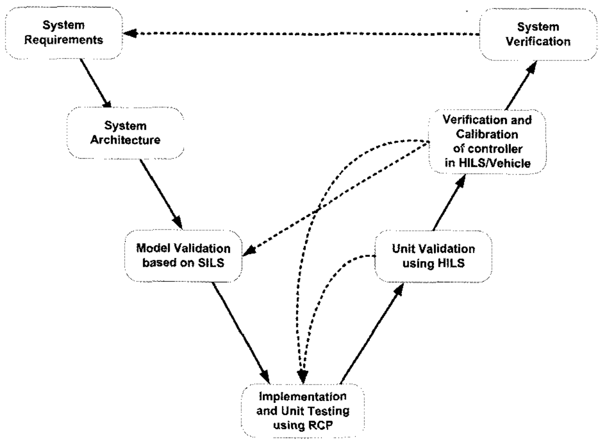

System Requirements
System Verification
System Architecture
Verification and Calibration of controller in HILS/Vehicle
Model Validation based on SILS
Unit Validation using HILS
Implementation and Unit Testing using RCP

Figure 2. Modified system development V.

validate the performance of algorithms on MCUs, even though the design of whole systems may not be completed. The goal of our research is to build a development framework for real-time embedded control systems that is composed of a timing analysis environment (Response-time Analysis Tool: RAT) (Choi *et al.*, 2004), SILS, RCP, and Hardware-In-the-Loop Simulation (HILS). By using the development framework, a modified development V-process is proposed as shown in Figure 2 (Lee *et al.*, 2004a; Lee *et al.*, 2004b; Song *et al.*, 2003).

In the modified system development V, the design process is simplified and software engineers are not required because the software design and the implementation steps are integrated. Control engineers can develop the control law and verify the performance by themselves. Since the block representations are equivalent to the software realization using RCP, the reuse of the verified algorithm blocks may mean the reuse of the reliable software. In many cases, the error of software is caused by error-prone hand-coding. The automatic code generation may be the one of the solutions for reliable system implementation.

SILS/RCP environments contribute toward seamless development process from design phases to implementation phases.

2.2. The Structure of SILS/RCP

SILS/RCP toolbox is shown in Figure 3 and 4.

In Figure 3, each subsystem is used for the real-time kernel configuration, the task representation, the system delay, and the in-vehicle network simulation.

The each block of RCP toolbox is related to the specific hardware modules of the MCU. According to the required hardware modules which are digital I/O, Analog-To-Digital (ATD) converter, Timer, CAN, etc., control engineers should replace the inputs and the outputs of control blocks with the corresponding RCP blocks before

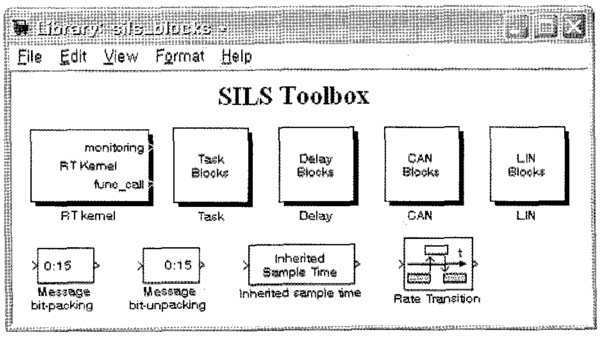


Figure 3. SILS toolbox.

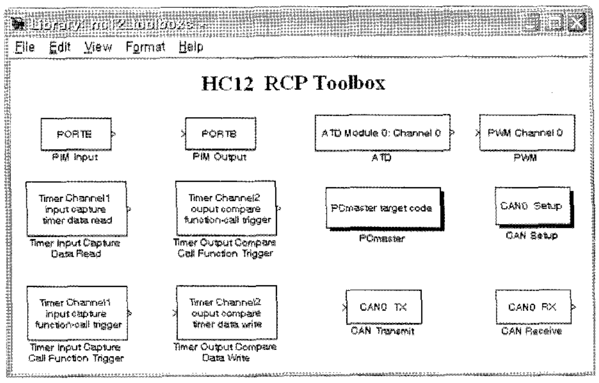

File Edit View Format Help
HC12 RCP Toolbox
PORTE
PIM Input
PORTB
PIM Output
ATD Module 0: Channel 0
ATD
PWM Channel 0
PWM
Timer Channel1 input capture timer data read
Timer Input Capture Data Read
Timer Channel2 ouput compare function-call trigger
Timer Output Compare Call Function Trigger
PCmaster target code
PCmaster
CAN0 Setup
CAN Setup
Timer Channel1 input capture function-call trigger
Timer Input Capture Call Function Trigger
Timer Channel2 ouput compare timer data write
Timer Output Compare Data Write
CAN0 TX
CAN Transmit
CAN0 RX
CAN Receive

Figure 4. RCP toolbox.

the implementation through RCP.

2.2.1. Real-time Kernel

In Simulink®, the execution of models is managed by internal state transitions, as the sampling rate is fixed or variable. However, in this scheme, it is not easy to express the real execution on the MCU scheduled by a real-time kernel or a RTOS. In order to support the simulation in which the task is the unit for the execution, the real-time kernel is used in SILS. The real-time kernel is following the static cyclic scheduling policy. The attributes configured in SILS for the scheduler are succeeded by RCP, so that the scheduling properties are maintained after the implantation.

Figure 5 shows the block of the Real-Time Kernel, and Figure 6 shows the attributes of the Real-Time Kernel. The scheduler is called in each *Hyper period.* After the

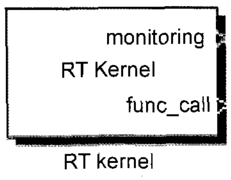
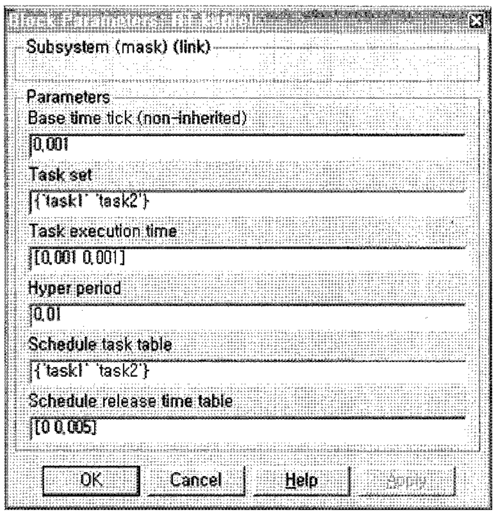


Figure 5. Real-Time Kernel based on static cyclic scheduling policy.



Subsystem (mask) (link)

Parameters

Base time tick (non-inherited): 0.001

Task set: {'task1' 'task2'}

Task execution time: [0.001 0.001]

Hyper period: 0.01

Schedule task table: {'task1' 'task2'}

Schedule release time table: [0 0.005]

Figure 6. Real-time Kernel configuration.

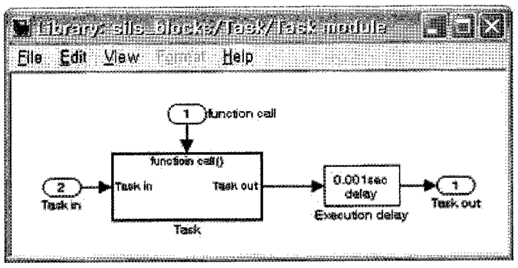


Figure 8. Subsytem of task module.

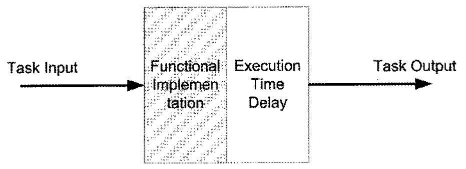


Figure 9. Concept of task execution.

release time defined in the *Schedule release time table*, the scheduler activates the corresponding task listed in the *Schedule task table*. The task release is fulfilled by the *func_call* signal.

2.2.2. Task modeling

In order to facilitate modularization, the concept of the task is adopted. The task is treated as a unit of function that is scheduled and executed by the system because SILS is scheduled based on the static cyclic scheduling policy (Liu, 2000). The task is defined as follows.

• *Task – An encapsulated sequence of operations that executes independently of other tasks* (Douglass, 1999).

The task model in SILS/RCP is shown in Figure 7. Figure 8 is the subsystem of the *Task Module* in Figure 7. The left block, *Task*, is related to the function of the task. The *Execution delay* block is the time delay for the execution time of the task.

The execution of the unit task consists of two parts: computation and output generation. The computations of the task are accomplished at the beginning of the task as shown Figure 9. The task output is generated after some delay defined by the *Execution Delay.*

The execution time of the task is governed by the following definition.

• *Task execution time – The amount of time required to complete the execution of the task when it is executed alone and has all the resources it requires* (Liu, 2000).

In order to measure the execution time of the unit task, the generated code for the unit task from RCP is used, since interrupt sources and preemptions are not allowed.

2.2.3. Network modeling

SILS/RCP environments provide the simulation and the implementation of CAN and LIN network protocol.

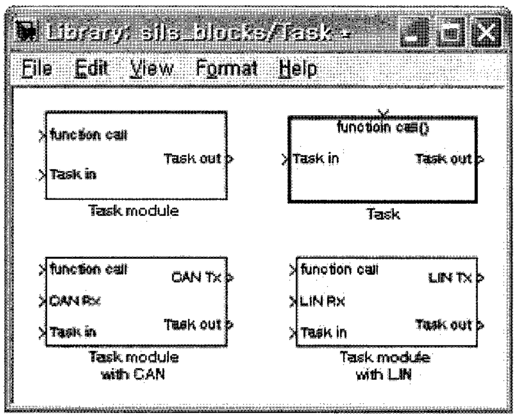


Figure 7. Task representation in SILS/RCP.

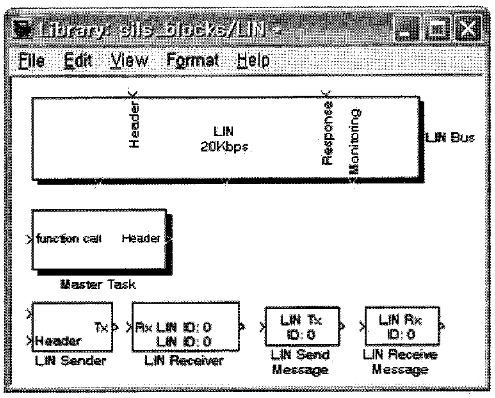


Figure 10. Network model of LIN protocol.

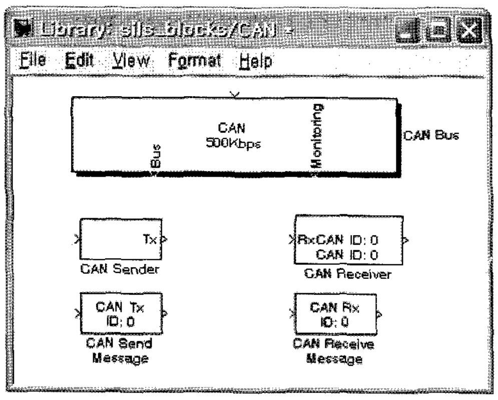


Figure 11. Network model of CAN protocol.

When control engineers design the distributed control system based on in-vehicle networks, SILS/RCP environments support the analysis of effects which are caused by network induced bus delays, communication rates, and so on. This allows a more realistic simulation can be performed from the early design phase. The simulation result from SILS makes it easier to evaluate the performance from a timing problem perspective.

2.2.4. Implementation using RCP

RCP is an implementation environment that generates the executable code from the designed model in SILS. Because SILS and RCP use the identical platform and same block representations based on Matlab/Simulink®, RCP can be realized using Real-Time Workshop Embedded Codder presented by Mathworks®. Embedded Codder creates only general C code, so additional work is required to connect Embedded Codder environment to the specific hardware. In this research, MC9SDP256B produced by freescale is selected as the target MCU. In order to link Embedded Codder to the target, HC(S)12 HAL[1] which is developed and distributed by ACE Lab[2] is used (Hodge *et al.*, 2004).

Since RCP provides the implementation environment, the performance verification on the MCU is available from the early design phase. As a result, the process from the design phase to the implementation phase is simplified.

3. CASE STUDY: ELECTRONIC THROTTLE CONTROLLER DESIGN

In order to validate the feasibility and the effectiveness of SILS/RCP environments, the design process of an Electronic Throttle Controller (ETC) is described.

3.1. ETC Modeling in SILS

An ETC model for SILS is shown in Figure 12. Figure 13 shows the controller of the ETC. In this case study, a PID controller is used which is divided into three tasks: *Sensing_Task*, *Computation_Task*, and *Actuation_Task*. In the *Sensing_Task*, the throttle angle is computed from the sensing value of the ATD converter. In the *Computation_Task*, the control input is calculated from the differences between the current throttle angle and the reference angle. In the *Actuation_Task*, the PWM signal is generated according to the control input of the *Computation_Task*.

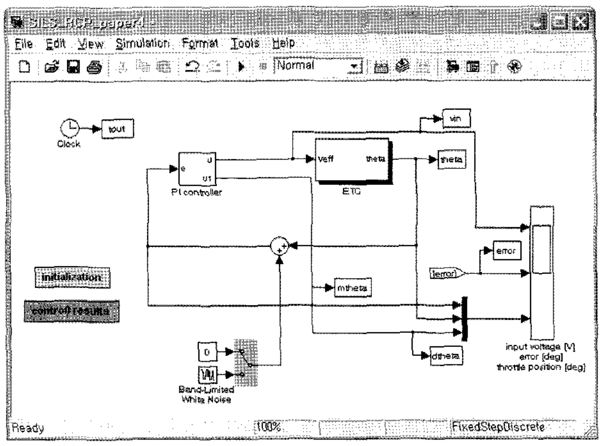


Figure 12. ETC model for SILS.

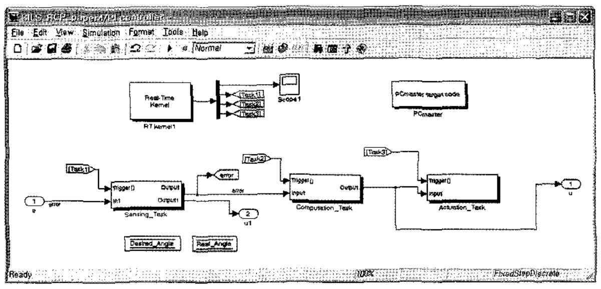


Figure 13. PID Controller of ETC for SILS.

3.2. Simulation and Implementation Result

The result from a simulation that does not consider temporal behaviors of the target system is shown in Figure 14.

The model used to generate the data in Figure 14 is transformed to the block diagram shown in Figure 12. The control gains are fixed in order to compare the previous result to the result given by SILS. The scheduling attributes which are the execution time and the release time are configured. The execution times of the *Sensing_Task*, *Computation_Taks*, and *Actuatioin_Task*

[1]HC(S)12 Hardware Abstract Layer: http://www.acelab.org/HAL

[2]Automotive Control and Electronics Laboratory at Hanyang University in Seoul, Korea http://www.acelab.org

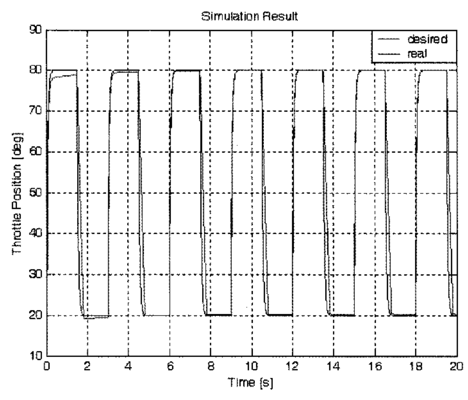


Figure 14. Simulation result without SILS.

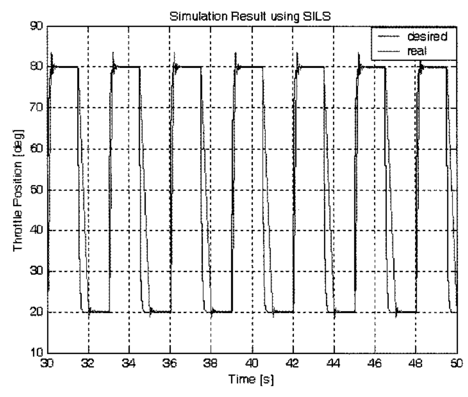


Figure 15. Simulation result with SILS.

are 1 ms, 2 ms, and 1 ms, respectively. These execution times are measured according to the definition of the execution time. The release times of the tasks are 0 ms, 1 ms, and 4 ms.

Figure 15 presents the simulation result in SILS environment. In this figure, overshoot, undershoot, and time delay in falling time are observed which are not seen in the simulation result generated without SILS. After gain tuning under SILS environment, the control performance is improved as can be seen in Figure 16.

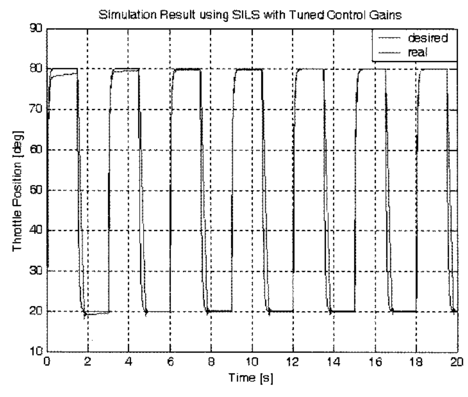


Figure 16. After gain turning with SILS.

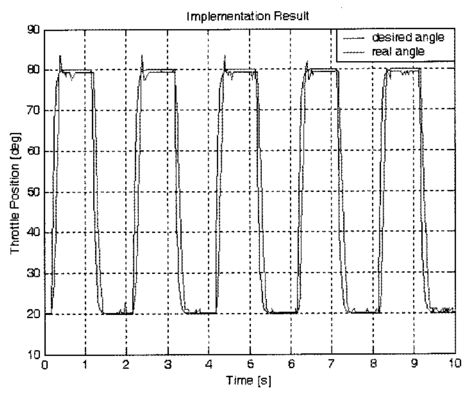


Figure 17. Implementation result using RCP.

Applying the tuned gain, executable code is generated by RCP. In Figure 17, the implemented result is shown.

From the Figures 16 and 17, the implemented result is reasonable except for the overshoot in the rising time. The occurrence of the overshoot is caused by modeling uncertainties and nonlinearity around 80°.

4. CONCLUSION

In this paper, SILS/RCP environments are proposed to support a seamless development process from the design phase to the implementation phase. The feasibility of the environments is validated from the ETC design process.

Control engineers are able to obtain more realistic simulation results using SILS environment from the early design phase. RCP supports an immediate phase shift from the design phase to the implementation phase. The integrated development environment which consists of SILS and RCP can simplify the development process, reduce the time-to-market, and provide greater performance-assured design. Especially, control engineers develop the control algorithms and implement the controller to the MCU using identical platform and same models with SILS/RCP environments.

ACKNOWLEDGEMENT–This research is supported in part by MOST (Ministry of Science and Technology) under the National Research Laboratory program, and part of the project 'Development of Partial Zero Emission Technology for Future Vehicle'. We are grateful for their financial support.

REFERENCES

Audsley, N. C., Burns, A., Richardson, M. F. and Wellings, J. (1994). STRESS: a simulator for hard real-time systems. *Software-Practice and Experience* **24, 6,** 543–564.

Cervin, A., Henriksson, D., Lincoln, B., Eker, J. and Årzén, K. (2003). Analysis and simulation of controller timing. *IEEE Control Systems Magazine* **23, 3,** 16–30.

Choi, J., Shin, M. and Sunwoo, M. (2004). Development of timing analysis tool for distributed real-time control system. *Int. J. Automotive Technology* **5**, **4**, 269–276.

Douglass, B. P. (1999). *Doing Hard Time-Developing Real-Time Systems with UML, Objects. Frameworks, and Patterns*. Addison-Wesley, Massachusetts.

Eker, J. and Cervin, A. (1999). A Matlab toolbox for real-time and control systems co-design. *Proc. 6th Int. Conf. Real-Time Computing Systems and Applications, IEEE Computer Society*, 320–328.

Englehart, M. and Jackson, M. (1994). Control H: a 4th generation language for real-time GN&C applications. *Proc. IEEE Symp. Computer Aided Control System Design*.

Gu, Z., Wang, S., Kim, J. C. and Shin, K. G. (2004). Integrated modelling and analysis of automotive embedded control systems with real-time scheduling. *World Cong. SAE Int.*, 115–122.

Henriksson, D., Cervin, A. and Arzen, K. (2002). Truetime: simulation of control loops under shared computer resources. *15th IFAC World Cong. Automatic Control*, Barcelona, Spain. 417–422.

Henzinger, T. A., Kirsch, C. M., Sanvido, M. A. A. and Pree, W. (2003). From control models to real-time code using giotto. *IEEE Control Systems Magazine*, 50–64.

Hodge, G., Ye, J. and Stuart, W. (2004). Multi-target modeling for embedded software development for automotive applications. *World Cong. SAE Int., In-Vehicle Networks and Software, Electrical Wiring Harnesses, and Electronics and Systems Reliability.* 85–90.

Kopetz, H. (1997). *Real-Time Systems*. Kluwer Academic Publishers, Massachusetts.

Lee, W., Park, S. and Sunwoo, M. (2004a). Towards a seamless development process for automotive engine-control system. *Control Engineering Practice*, **12**, 977–986.

Lee, W., Shin, M. and Sunwoo, M. (2004b). Target-identical rapid control prototyping platform for model-based engine control. *IMechE*, Part D, **218,** 755–765.

Liu, J. W. S. (2000). *Real-Time Systems*. Prentice Hall. New Jersey.

Mitchell, E. E. L. and Gauthier, J. S. (1986). *ACSL: Advanced Continuous Simulation Language-User Guide and Reference Manual*. Mitchell & Gauthier Associates. Concord MA.

Rebeschieß, S. (1999). MIRCOS-Microcontroller-based real time control system toolbox for use with Matlab/Simulink. *Proc. IEEE Int. Symp. CACSD*, Hawaii, USA, 243–248.

Song, J., Lee, W. and Sunwoo, M. (2003). Development of a rapid control prototyping platform for engine control system. *Trans. Korean Society of Automotive Engineers* **11, 1**, 160–165.

Storch, M. F. and Liu, J. W.-S. (1996). DRTSS: A simulation framework for complex real-time systems. *Proc. 2nd IEEE Real-Time Technology and Applications Symp.*, 160–169.

Toeppe, S., Ranville, S., Bostic, D. and Wang, Y. (1999). Practical validation of model based code generation for automotive applications. *Proc. 18th AIAA/IEEE/SAE Digital Avionics System Conf.*, 10.A.3-1-10.A.3-14.

Törngren, M. and Redell, O. (2000). A modelling framework to support the design and analysis of distributed real-time control systems. *Microprocessors and Microsystems*, **24**, 81–93.

UML Notation Guide. Version 1. 1. (1997). Object Management Group, doc. no. ad/97-08-05.http://www.rational.com/uml.

Vestal, S. (1994). Integrating control and software views in a CACE/CASE toolset. *IEEE/IFAC Joint Symp. Computer Aided Control System Design for Control Systems*, Tucson, AZ, USA, 353–358.

Zhu, J. (1994). Design and simulation of hard real-time applications. *Proc. 27th Annual Simulation Symp.*, 217–225.