Proceedings of the 17th World Congress
The International Federation of Automatic Control
Seoul, Korea, July 6-11, 2008

Development of SILS and RCP for OSEK-OS based ECU

Myoungho Sunwoo*

** Department of Automotive Engineering, Hanyang University, Seoul Korea (Tel: +82-2-2220-0453; e-mail: msunwoo@hanyang.ac.kr).*

Abstract: This paper presents the Matlab/Simulink-based Software-in-the-Loop Simulation (SILS) tool for OSEK-OS based ECU. The SILS tool has the capability for temporal and functional simulations of control systems. The temporal behavior of a control system is mainly dependent on the implemented software and hardware such as the real-time operating system (OSEK-OS), the target CPU, and the communication protocol. The SILS components with temporal attributes are specified as tasks, task executions, real-time schedulers (OSEK-OS scheduler), and real-time networks. Methods to realize these components in graphical block representations are investigated with Matlab/Simulink. Furthermore, in order to achieve a seamless development process from SILS to Rapid Control Prototyping (RCP), the SILS block set is designed to support automatic code generation in C codes without tool changes and block modifications.

1. INTRODUCTION

One of the goals in automotive design is the creation of a seamless development framework from Software-in-the-Loop Simulation (SILS) to Rapid Control Prototyping (RCP). Current computer-aided control system design (CACSD) tools are inadequate for the realization of a seamless process due to their inability for representing certain software architecture and considering temporal behavior.

The use of CACSD software in control systems for supporting design frameworks has increased over the years [1][2][3][4]. Among these tools Matlab/Simulink has become a standard tool for modeling and off-line simulation in control system design. Matlab/Simulink is not only a tool used for system modeling, simulation, and analysis, but it is also a powerful tool for the real-time system development in combination with automatic code generation technology [5].

Matlab/Simulink is used by control engineers to design control algorithms by focusing on the functional requirements [6]. In general, control engineers do not consider how events are timed and how tasks are scheduled. They usually 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 [7]. For example, Simulink determines a fixed and variable time step that a simulation needs to task in order to meet target accuracy requirements. The simulation accuracy is related only to functional performance. However, the performance of a control system is normally affected by implementation software architecture such as the task, operating system, and target execution platforms. These factors are mostly related to how tasks are scheduled, which include task execution delays, scheduling delays, and network transmission delays. Therefore, it is necessary to incorporate such temporal behavior into currently used CACSD tools.

There have been several attempts to incorporate task scheduling into Matlab/Simulink environments such as TRUETIME and AIRES. TRUETIME is a toolbox for the simulation of distributed real-time control systems and makes it possible to simulate the temporal behavior of real-time kernels executing controller tasks using S-functions provided by MathWorks [7]. However, the TRUETIME toolset only supports simulation and is not connected directly to automatic code generation. In addition, TRUETIME does not support all Simulink blocks, which restricts graphical representations for the design of control algorithms. AIRES is used as a co-simulation tool that is designed to model the embedded software and the target execution platform, and to perform real-time analysis to give the system designer early feedback on system timing behavior [6]. However, real-time specifications must be described using a meta-modeling tool called Generic Modeling Environment (GME).

Figure 1(a) shows two types of simulations in a single electronic control unit (ECU). Basically, Simulink considers the execution time of a control algorithm as zero or simple delay, as shown on the left. On the other hand, the control software running on the target ECU has complex time delays determined by the implementation platform such as the scheduling policy, task characteristics and ECU performance. The figure on the right shows an environment for the SILS-based simulation of control algorithms. The control algorithm consists of tasks, and each task is treated as a unit of function, and simulated by a real-time scheduler.

In the case of a distributed control system, the control performance deteriorates due to network-induced delays which can vary widely according to the transmission time of a message and the overhead time of transmission. Figure 1(b) shows differences between Simulink and the SILS environment simulation with multiple nodes. SILS supports the simulation of functions and event timing in network-based control systems.

In this research, the SILS components with time attributes are specified as tasks, task executions, real-time schedulers, and real-time networks. The OSEK-OS is used as a real-time scheduler of this research. CAN and LIN protocols are used as real-time networks. Possible methods for realizing these components in graphical block representations are investigated with Matlab/Simulink. Additionally, in order to construct a seamless development process from SILS to RCP, the SILS block set is designed to support automatic code generation.

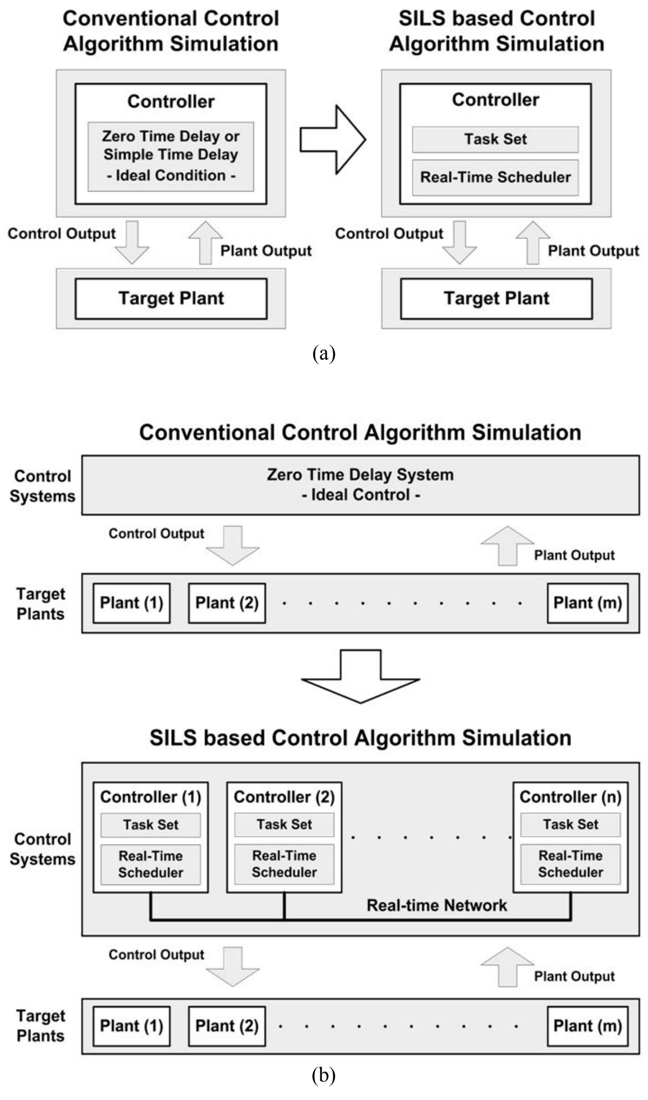


Fig. 1. Comparison of conventional and SILS based control algorithm simulation: (a) single-node and (b) multi-nodes

2. SILS ENVIRONMENT

In order to consider time delays induced by real-time tasks, schedulers, and networks, the SILS provides the following design elements:

- Real-time scheduling kernel which is based on OSEK-OS scheduler
- Task execution model which performs both simulation and code generation
- Real-time network kernel commonly used in the automotive industry

2.1 Modeling of real-time scheduler

Over the past decade, automotive manufacturers have increased the deployment of embedded microcontrollers and software to enhance the quality, the safety, and the fuel efficiency of automobiles. They have applied software architecture with a real-time operating system (RTOS) to provide control of the overall system performance and to improve reusability. In many cases, the RTOS introduces a fixed priority scheduling algorithm to provide design flexibility. OSEK-OS, which is a standard RTOS interface for automotive electronics, also specifies this scheduling policy.

The real-time scheduler of the SILS supports scheduling policy of OSEK-OS which is fixed priority scheduling algorithm.

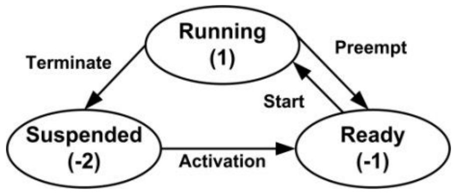


Fig. 2. State transition diagram of a task for fixed priority-based scheduler

2.2 OSEK-OS scheduler model

The fixed priority-based scheduling algorithm is used for the scheduler model of OSEK-OS. The term “priority-based scheduling algorithm” refers to a large class of scheduling algorithms that never intentionally leave any resource idle [9]. The scheduling algorithm assigns a static and unique priority to each task. A set of priorities for task scheduling is selected to make a system schedulable and also to guarantee all deadlines, considering the resource, precedence, and synchronization requirements of all tasks.

This scheduling mechanism operates based on events. The scheduler responds to a set of events, where time is simply one particular type of event [8]. Each task in the system is activated by the arrival of a triggering event, and is ready to run. At any scheduling decision time, the task with the highest priority is scheduled and executed on the available processors. If the running task can be preempted, scheduling decisions are made whenever some task becomes ready for execution or the executed task is completed. On the other hand, if a nonpreemptable task is running, the scheduling decision is only made when the task is terminated.

The task executed by the fixed priority algorithm has an additional state: READY. A task is started either in the SUSPENDED or READY state, depending on the configuration of the task. The scheduler always assigns the CPU time to the highest priority task which is in the READY state at the scheduling decision time. The state of the task changes from READY to RUNNING. If the task is preemptable and higher priority tasks are in the READY state, the original task is preempted, and resumed when all higher priority tasks are completed. When the CPU time assigned to the task in the RUNNING state is equal to its execution time, it is suspended by the scheduling kernel.

The SILS considers two types of event triggers for the task execution: EVENT and TIMER. EVENT is defined as an immediately processed trigger source. It is usually applied to synchronize tasks. A TIMER is triggered when the corresponding timer of the scheduler reaches a preset time value. It is used to schedule a periodically executing or delayed task. The scheduling kernel establishes and monitors the trigger of events and processes the execution and state transition of tasks.

2.3 Task execution modeling

A task is a component independently executed by the real-time kernel. Each task is assumed to take a specified nonzero amount of time to execute. The properties of a task are defined by a set of attributes which consists of preemptive, priority, name and execution time. The execution time can be constantly updated by the user during simulation. Other attributes are normally kept constant. However, the execution time can be changed by calls to kernel primitives when the task is executing.

In the SILS, all data for control are communicated through Simulink input and output ports. When a task is triggered, the task computes output data using input data from the input ports based on internal logic. After the task which is executing uses the CPU time, the data produced is passed to the output ports as soon as the task is triggered by the scheduling kernel.

The execution of the task processed by a priority-based scheduler can be preempted by higher priority tasks or external interrupts. Design specifications such as precedence relationships and synchronizations between tasks intentionally allow task executions to be discontinued or delayed. Furthermore, due to nonpreemptable or noninterruptable constraints of the running task, the highest priority task or interrupt service can be blocked (see Figure 3).

As the result, task execution is modeled as two independently executed subsystems (Figure 4). The first subsystem, called a computation subsystem, is executed when a task is invoked. It calculates output data, which is not available until the task is terminated. In order to update output ports, the second subsystem, the output subsystem, is invoked. The invocation of this subsystem is implemented using a function-call trigger provided from Simulink. A function-call subsystem is converted into C functions during automatic code generation [10].

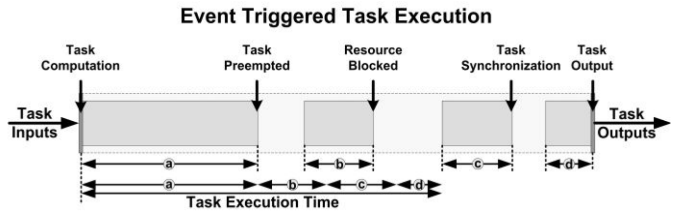


Fig. 3. Task execution model for priority based scheduling

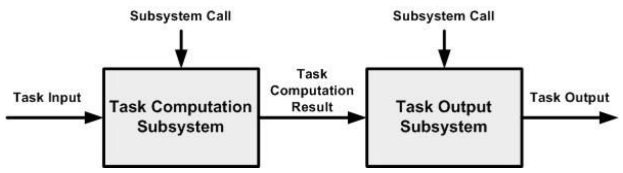


Fig. 4. Task execution Simulink block representation

2.4 Modeling a real-time network

In networked control systems, various delays of variable length occur from sharing a common network medium. These delays are called network-induced delays [11]. Network-induced delays can vary widely according to the transmission time of messages and the overhead time, both of which are dependent on the applied network protocol. The SILS and 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 in the automotive industry.

2.5 CAN protocol modeling

CAN is referred to as a multi-master network, since all network nodes can act as a master or a slave. During one message frame, only one node acts as a master and the other nodes act as slaves. The multi-master capability of CAN uses a carrier sense multiple access and collision detection (CSMA/CD) arbitration protocol that determines which node sends its message frame to the bus via bitwise arbitration [12].

Each message has a unique identifier represented as an 11-bit number. The identifier is used for two purposes: message filtering upon reception and priority assigning of a message [13].

1) A CAN Controller in each node selects the highest priority message among messages ready to be sent, and moves the message into the assigned transmit buffer to be sent onto the CAN bus.
2) The CAN bus schedules the messages in all transmission buffers in a priority-driven manner. The highest priority

message is transmitted to CAN Controllers. The transmission delay is also defined as in equation (2).

3) The message filtered by the CAN Controller is only stored in the reception buffer.

A CAN message is able to hold up to 8 data bytes. For each message, the total bit size of the message is determined by adding 47 bits and stuff bits to the bit size of each message data. The communication overhead of 47 bits consists of the start of frame (SOF), arbitration, remote transmission request (RTR), control, cyclic redundant check (CRC), acknowledgment, end of frame (EOF), and possible stuff bits. The maximum number of stuff bits which can be inserted to synchronize communication is defined by equation (1). Equation (2) shows the maximum time (C) for CAN message transmission [9].

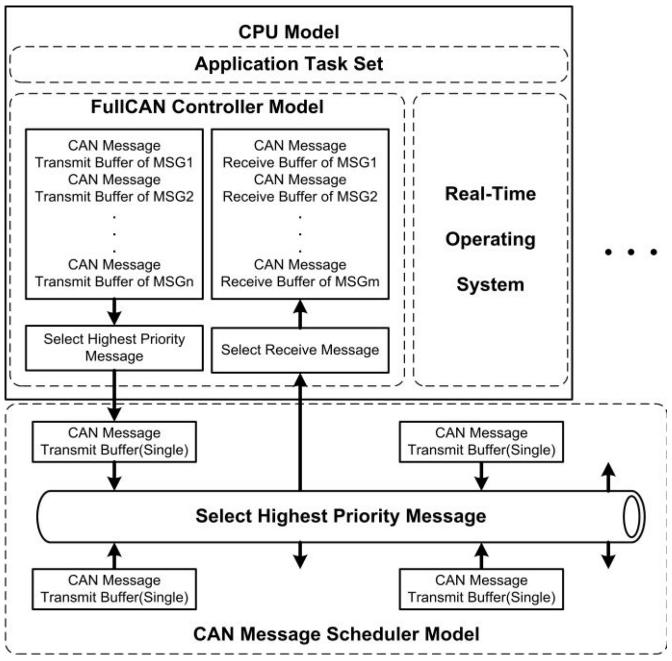

CPU Model
Application Task Set
FullCAN Controller Model
CAN Message Transmit Buffer of MSG1
CAN Message Transmit Buffer of MSG2
CAN Message Transmit Buffer of MSGn
CAN Message Receive Buffer of MSG1
CAN Message Receive Buffer of MSG2
CAN Message Receive Buffer of MSGm
Real-Time Operating System
Select Highest Priority Message
Select Receive Message
CAN Message Transmit Buffer(Single)
Select Highest Priority Message
CAN Message Scheduler Model

Fig. 5. CAN communication modeling

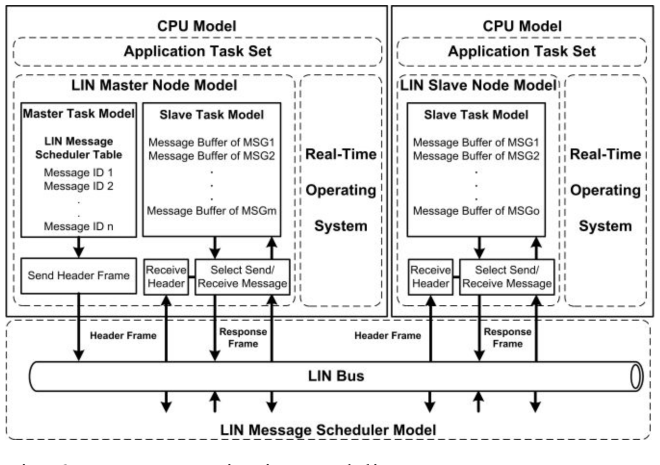


Fig. 6. LIN communication modeling

$$(Stuffbits)_{\max} = \left\lfloor \frac{34+8s-1}{4} \right\rfloor \tag{1}$$

$$C = \left(8s + 47 + \left\lfloor \frac{34+8s-1}{4} \right\rfloor \right) \tau_{bit} \tag{2}$$

where, s is size of data field and τ_{bit} is the transmission rate.

2.6 LIN protocol modeling

The LIN protocol has a single master with multiple slaves. Therefore, a LIN network consists of one master task and several slave tasks [15]. A master node contains the master task as well as a slave task. All other nodes only contain a slave task only. The master task decides when and which frame shall be transferred on the bus. The slave tasks provide the data transmitted by each frame.

The LIN protocol is designed to use a static cyclic scheduling algorithm for transmitting header frames. The message scheduling table is constructed based on all message periods and transmission times. The header contains an identifier which is uniquely assigned to each message. The slave task designated to provide the response associated with the identifier transmits the response frame. (see Figure 6)

1) The master task is periodically called by the scheduling kernel, and transmits header frames based on the schedule table. The transmitted header frame is passed from the LIN bus to slave tasks with a transmission delay according to equation (3).
2) All slave tasks in nodes connected to the LIN bus decide to send or receive the response frame associated with the identifier in the received header frame.
3) The slave task designated by the identifier as the transmitting node sends the response data onto the LIN bus with a transmission delay according to equation (4). Then, slave tasks in the receiving node copy the response data to a message buffer for each task.

The transmission time of a header frame is calculated using equation (3), which consists of the four time parameters and the transmission rate (τ_{bit}).

$$T_{header} = 34\tau_{bit} + P_{INTER} + P_{SYNBRK} + P_{SYNDEL} + P_{ID} \tag{3}$$

where,
P_{INTER} : Delay for preparing a header frame transmission
P_{SYNBRK} : Additional time for synchronization of synch break field
P_{SYNDEL} : Additional time for synchronization of delimiter field
P_{ID} : Delay for preparing identifier field

A response frame can transmit up to 8 bytes of data, and contains one byte checksum field to verify the received data on the slave tasks. The equation to calculate the transmission time of a response frame is formulated with two time parameters: transmission rate and data length (s) as follows:

$$T_{response} = 10(s+1)\tau_{bit} + P_{INFRAME} + P_{INTERBYTE} \times s \qquad (4)$$

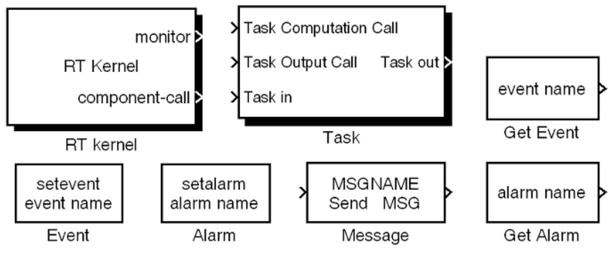
where,

$P_{INFRAME}$: Inter-frame response space

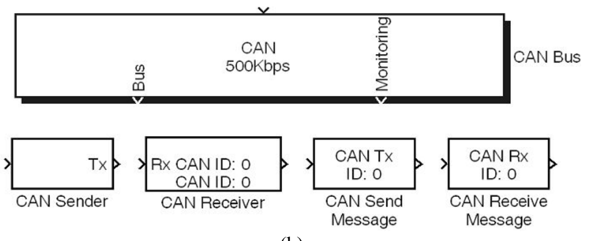
$P_{INTERBYTE}$: Inter-byte space

3. THE SILS AND RCP BLOCK-SET

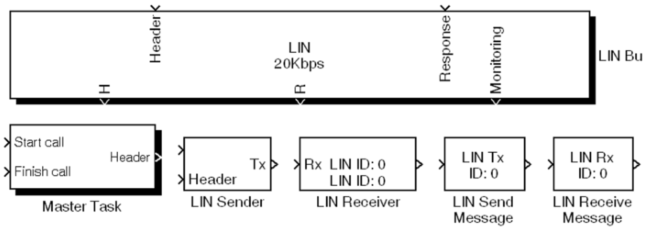
Figure 7 shows the SILS block set which is designed using Level-2 M-file and C-file S-functions provided by Matlab/Simulink [10]. An S-function not only supports the simulation of the custom code, but also automatically generates the implementation code specified to a target platform. The attributes configured in the SILS are followed by RCP, so that the scheduling properties are maintained after implementation. The real-time scheduling kernel during the simulation phase is directly mapped to OSEK-OS [16]. All designed Simulink blocks are compatible with the RCP platform created during a previous study [17]. As a result, control engineers can develop the control algorithms and implement the controller to the target ECU in an identical design environment.



(a)



(b)



(c)

Fig. 7. Simulink block-set: (a) RTOS, (b) CAN, and (c) LIN

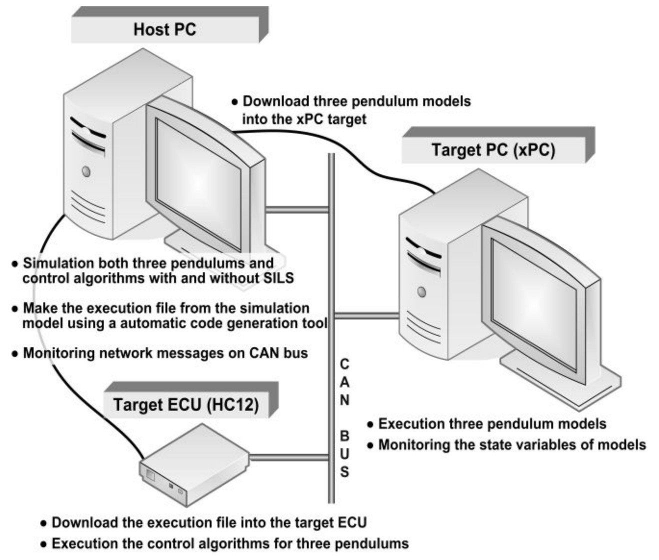

Host PC
• Download three pendulum models into the xPC target
Target PC (xPC)
• Simulation both three pendulums and control algorithms with and without SILS
• Make the execution file from the simulation model using a automatic code generation tool
• Monitoring network messages on CAN bus
CAN BUS
Target ECU (HC12)
• Execution three pendulum models
• Monitoring the state variables of models
• Download the execution file into the target ECU
• Execution the control algorithms for three pendulums

Fig. 8. HILS system for SILS environment evaluation

4. CASE STUDY FOR THE SILS EVALUATION

Figure 8 shows the evaluation system diagram to assess the feasibility of the proposed SILS environment. The implementation code which is generated from the SILS/RCP environment is effectively tested and verified by using PC-based Hardware-in-the-Loop Simulation (HILS). The HILS provides a test and evaluation environment regardless of the modeling uncertainties, disturbances, and noises of real systems. Therefore, the test result is mainly affected by the control algorithm and implementation platform.

In this case, the control system is configured as a distributed control system using smart sensors and smart actuators, which exchange the control information through a network bus. The main electronic control unit has three control algorithms, which are independently executed, in order to control three inverted pendulums (see Figure 9). The angle of three pendulums and the position of three carts are controlled by the horizontally concentrated force which is calculated by LQR and PID control logics.

The priority based scheduler, OSEK-OS, and the network bus, CAN, are configured as Table I and II. There exist two tasks in each control loop. The first task handles the sensor input and executes the state estimation, and the other task generates the control output based on designed control algorithms. All control information between ECU and smart nodes are transferred in the form of network messages through the CAN.

The integral of the absolute value of the input (IAI) instead of the error is used for performance evaluation of the control system [18]. The mathematical formula for the IAI is as follows:

$$\mathrm{IAI} = \int_{t_0}^{t_f} |F|\,dt,\ or\ \sum_{k=k_0}^{k_f} |F_k|$$

where, $t_0\,(k_0)$ and $t_f\,(k_f)$ are the initial and final times of the evaluation period in continuous (discrete) time, and F is the input force for the control of a inverted pendulum system.
We evaluate control performances according to the assignment of priorities of ECU's tasks and the sampling period of the smart sensors. Table I and II show the configuration of ECU's tasks and CAN messages for the simulation and the implementation. Figure 10(a) shows the block diagram of the whole system which is formed by three sensor nodes, three actuator nodes, the main ECU, the CAN kernel, and three inverted pendulums. Figure 10(b) shows block diagram in the main ECU including the scheduler block, task blocks and CAN interface blocks. The simulation model is constructed using the SILS block-set in Figure 7. The identical Simulink model is used at the implementation phase.

TABLE I
TASK INFORMATION

Task name	Period [ms]	exe. time [ms]	Preemption	Priority	
				Case 1	Case 2
Pend1_In_T	10	0.7	yes	13	13
Pend1_Out_T	10	0.8	yes	8	12
Pend2_In_T	10	0.7	yes	12	11
Pend2_Out_T	10	0.8	yes	9	10
Pend3_In_T	10	0.7	yes	11	9
Pend3_Out_T	10	0.8	yes	10	8

TABLE II
CAN INFORMATION: 500KBPS

Message name	ID	Data Length [bytes]	Data Type	Period [ms]
Cart1_pos_msg	1	8	Double	10
Pend1_ang_msg	2	8	Double	10
Cart1_input_msg	3	8	Double	10
Cart2_pos_msg	4	8	Double	10
Pend2_ang_msg	5	8	Double	10
Cart2_input_msg	6	8	Double	10
Cart3_pos_msg	7	8	Double	10
Pend3_ang_msg	8	8	Double	10
Cart3_input_msg	9	8	Double	10

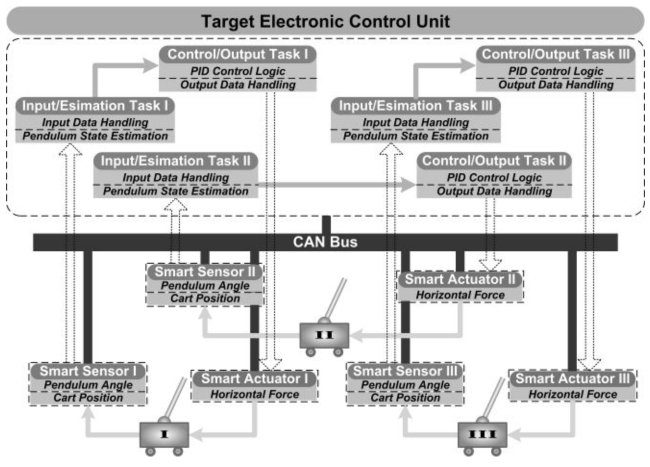


Fig. 9. Configuration of the evaluation system with three pendulums

Figure 11(a) shows performance evaluation results of the simulation and the implementation configured as Case 1 and 2 in table I. The evaluation systems are repeatedly simulated and tested by changing the realization time of the initialization tasks. Control inputs of Case 1 are smaller than those of Case 2 for both the simulation and the implementation. In all cases, at implementation, the amplitude of control inputs is similar with the amplitude of control inputs in the simulation. Additionally, in order to analyze the effect of the sensing period at the sensor nodes, the evaluation system is simulated with various sensing periods: 2ms, 3ms, 4ms, and 6ms. Figure 11(b) shows that the SILS simulation results are similar to the implementation results for all tested sensing periods.
The results show that the proposed SILS environment provides a realistic simulation result which is similar to the implementation result. In the early design phase, control system designers can evaluate the control performance by considering temporal factors, such as task execution delay and real-time operating system and real-time network.

5. CONCLUSION

In this paper, we have proposed a SILS environment of specified graphical blocks such as tasks, task executions, real-time scheduler (OSEK-OS scheduler), and In-vehicle networks. In addition, in order to achieve a seamless development process from SILS to RCP, the SILS block set is designed to support automatic code generation in C codes without tool changes and block modifications.

The control engineer can use the SILS environment to evaluate the software-induced performance degradation of the complex time-delayed system by the CPU and the network during the simulation phase. Also, this tool can be used by the software engineer to analyze the performance of designed control systems by changing the software design factors, such as real-time scheduling of the CPU and network, task assignment in one node, and allocation in multiple nodes. The SILS environment developed here will help bridge the gap between control and software engineering in the design and development processes.

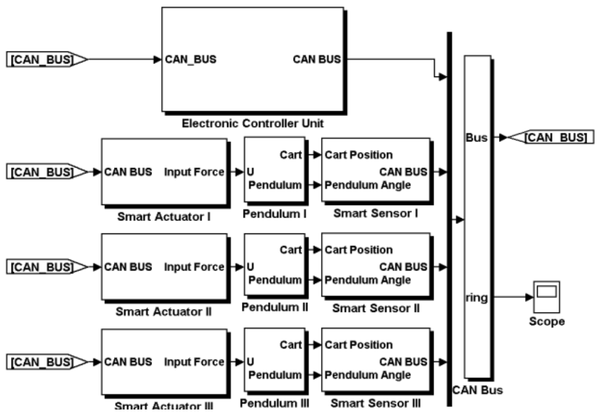
ACKNOWLEDGMENTS

This work was supported by the Korea Science and Engineering Foundation (KOSEF) through the National Research Lab. Program funded by the Ministry of Science and Technology (No. M10400000339-06J0000-33910).

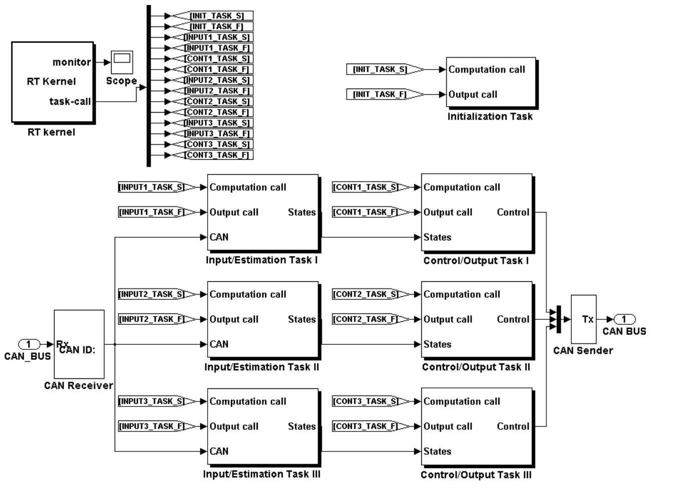
REFERENCES

[1] W. Lee, S. Park, M. Sunwoo, "Towards a seamless development process for automotive engine-control system", Control Engineering Practice, Vol. 12, pp.977~986, 2004
[2] M. Shin, W. Lee, M. Sunwoo, "Implementation- conscious Rapid Control Prototyping Platform for Advanced Model-based Engine Control", SAE Congress paper, 2003-01-0355

[3] S. Toeppe, S. Ranville, D. Bostic, Y. Wang, "Practical Validation of Model Based Code Generation for Automotive Applications", IEEE, 1999
[4] G. Hodge, J. Ye, W. Stuart, "Multi-Target Modeling for Embedded Software Development for Automotive Applications", 2004 World Congress SAE International, 2004
[5] MathWorks, http://www.mathworks.com
[6] Z. Gu, S. Wang, J. Kim, K. Shin, "Integrated Modeling and Analysis of Automotive Embedded control systems with Real-Time Scheduling", SAE Congress paper, 2004-01-0279
[7] D. Henriksson, A. Cervin, K. Arzen, "TrueTime: Simulation of Control Loops Under Shared Computer Resources", 15th IFAC World Congress on Automatic Control, 2002
[8] T. Rolina, N. Tracey, "Why Switch to an OSEK RTOS and How to Address the Associated Challenges", SAE Congress paper, 2005-01-0312
[9] J. Liu, "Real-Time Systems", Prentice Hall, 2000
[10] MathWorks, Writing S-Functions Ver.4, 2004
[11] W. Kwon, S. Choi, " Real-Time Distributed Software-In-the-Loop Simulation for Distributed Control Systems", IEEE International Symposium on Computer Aided Control System Design, 1999
[12] T. Gerke, C, Schanze, "Development and Verification of In-Vehicle Networks in a Virtual Environment", SAE Congress paper, 2005-01-1534
[13] K. Tindell, A. Burns, "Guaranteeing message latencies on control area network (CAN)", Technical report, Department of computer science. University of York. England, 1994
[14] Freescale Semiconductor, http://www.freescale.com
[15] J. Youn, M. Shin, W. Lee, M. Sunwoo, "A Study on Timing Model and Analysis of LIN Protocol", 2003 spring conference proceeding of the KSAE, pp.952~957, 2003
[16] Motorola, "OSEK/VDX Operating System Ver. 2.1", 2000
[17] J. Ma, J. Youn, M. Shin, M. Sunwoo, "SILS/ RCP: Integrated Model based System Development Tool for automotive embedded control system design", 2004 spring conference proceeding of the KSAE, pp.143~149, 2004
[18] G. F. Franklin, J. D. Powell, and A. Emami-Naeini, Feedback Control of Dynamic Systems, 3rd ed. Reading, MA: Addison-Wesley, 1994.

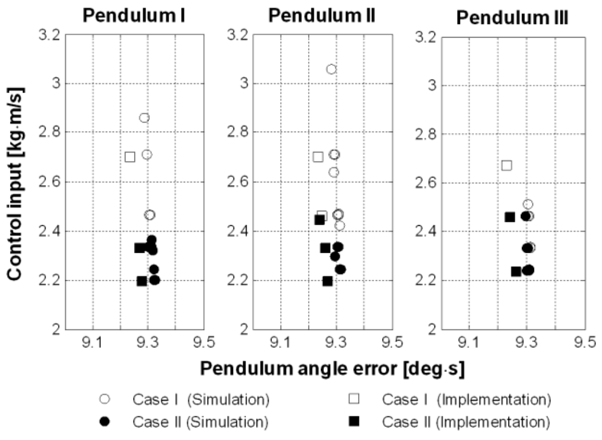


(a)

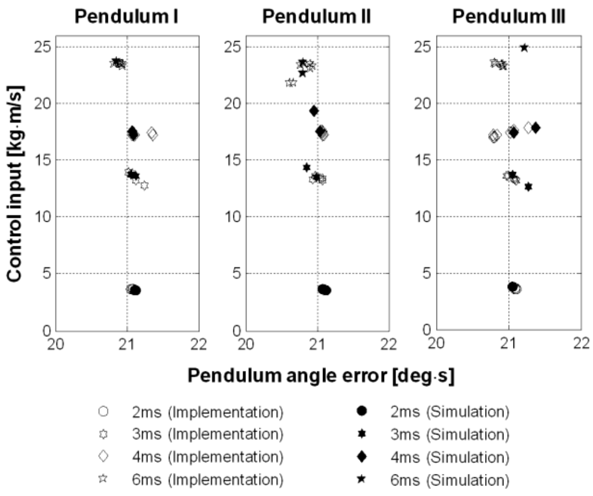


(b)

Fig. 10. Simulink models of the evaluation system: (a) System block diagram and (b) Block diagram in ECU



(a)



(b)

Fig. 11. (a) Control performance versus priority assignment and (b) Control performance versus sampling period