 Digital Object Identifier : 10.1109/TENCON.2010.5686017

Design and Implementation of Hard-wired Tracking Control System

Yasuaki Tezuka*
* Dept. Knowledge-based Information Engineering
Toyohashi University of Technology
Tempaku, Toyohashi 441-8580, Japan

Shuichi Ichikawa†
† Dept. Electrical and Electronic Information Engineering
Toyohashi University of Technology
Tempaku, Toyohashi 441-8580, Japan
E-mail: ichikawa@ieee.org

Yoshiyuki Noda‡
‡ Dept. Mechanical Engineering
Toyohashi University of Technology
Tempaku, Toyohashi 441-8580, Japan
E-mail: y-noda@tut.jp

***Abstract*—Many of recent control systems are required to finish complex calculations in a very short period. Hard-wired implementation of control logic is a prospective solution to these computationally demanding applications. This study presents an implementation of tracking control system, which was first designed with Simulink, converted into VHDL with HDL Coder, and then implemented with an Altera Stratix III FPGA device. The derived performance of the hard-wired predictive control circuit was 15 M steps/s, which is approximately 20 times faster than the corresponding software implementation on Intel Atom Z530 (1.6 GHz). The hard-wired implementation fulfills the requirement for real-time prediction (1.1 M steps/s), which could not be satisfied by the software implementation.**

***Keywords*—Custom Circuit; Vibration Control; Predictive Control; MATLAB; Simulink**

I. Introduction

With the recent advances in control theory, control performance has been drastically improved in a wide range of applications. In exchange for such advances, recent control applications became computationally demanding; that is, such an application requires a considerable amount of computation for each control output. In a highly responsive system, the requirements for computational performance becomes even higher.

In the research and development of control systems, Simulink [1] is widely used to design the models of time-varying systems. Simulink is integrated with MATLAB [2], which provides a high-level language and an interactive graphical environment for model-based design, simulation, and implementation. Real-Time Workshop [3] is optional software to convert Simulink models into C language programs, which can be ported to various processors and operating systems. The ported control programs are usually executed with DSPs or embedded processors, though the derived performance is not always sufficient for recent real-time control applications.

Recently, the design and implementation of hard-wired control system are attempted for higher performance. Field Programmable Gate Array (FPGA) technology is regarded best suited for this purpose. FPGA devices have been used in a wide variety of application-specific systems, and are rapidly evolving in both logic scale and performance. FPGA is a reconfigurable device, whose flexibility is naturally suited to replace a software-based system.

The problem of this approach is that it is not generally easy for control system designers to design hardware with Hardware Description Language (HDL) and to use CAD software instead of their familiar tools. It is hence desired to provide an easy way for control system designers to convert a Simulink model into the corresponding FPGA design.

Simulink HDL Coder [4] is optional software to generate cycle-accurate HDL codes (Verilog HDL or VHDL) from Simulink models. The HDL codes generated by HDL Coder are stated to be portable and synthesizable with most popular synthesis tools; HDL Coder is a desirable tool for control system designers to design hard-wired control system with Simulink. Meanwhile, HDL Coder is a relatively new product that imposes many restrictions upon designers. It is not evident whether a practical system can be converted by HDL Coder. Even if the conversion is possible, the logic scale and the performance of the generated circuit are not proven to be acceptable for practical applications.

The first purpose of this study is to examine the feasibility of the above-mentioned design method for a practical control application: a tracking control system with vibration control. The second purpose is to examine the logic scale and the performance of the hard-wired control circuit implemented by this method with a common FPGA device.

In signal processing applications, there are a few software products that convert a Simulink model into HDL descriptions: e.g., System Generator for DSP [5] from Xilinx, DSP Builder/Simulink [6] from Altera, and Synplify DSP [7] from Synopsys. A major problem of these products is that they are aimed at signal processing applications. Since the options and the toolboxes of Simulink are different for each application, it is not straight-forward to use the software for DSP applications to design control systems.

Each of System Builder, DSP Builder, and Synplify DSP provides users with its own blockset that supports various functions for digital signal processing; users have to construct a Simulink model with the provided blockset to generate HDL descriptions with these tools. The tools from FPGA vendors are specially tailored for their own FPGA devices,

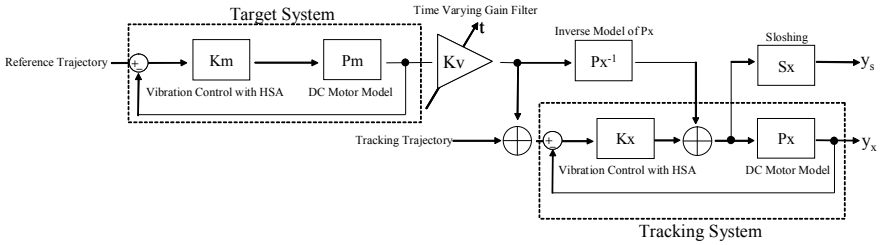

Fig. 2. Tracking control system [8] [9].

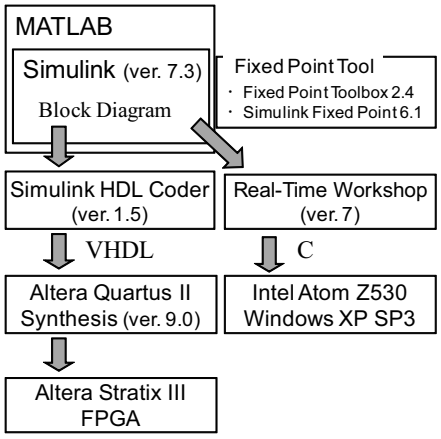

Fig. 1. Tools and design flow.

and are expected to generate an effective design in exchange for portability of the design. System Builder, DSP Builder, and Synplify DSP are the products of third parties, and generally have some problems with portability and compatibility to Simulink models.

In this study, HDL Coder [4] was adopted to convert Simulink model into HDL codes. HDL Coder is a product of Mathworks, which is the vendor of MATLAB and Simulink. HDL Coder is not specific to DSP applications, and converts a general Simulink model into HDL codes. Although the supported blocks are limited, HDL Coder does *not* force users to use a special blockset. For these reasons, HDL Coder is expected to be applicable to a wider range of applications. Though HDL Coder was adopted in this study, the tools should be selected for each specific case. Other tools might be preferable in other cases.

Figure 1 summarizes the evaluation process and the tools used in this study. First, the target model (continuous model) is constructed and verified with Simulink. The model is then discretized to construct the corresponding discrete model with adequate fixed-point data type. The sampling frequency is set to 512 Hz in all examples of this study. HDL Coder converts this discrete model into the corresponding VHDL code, which is synthesized, placed, and routed with Altera Quartus II 9.0 software. Target device is set to Altera Stratix III FPGA EP3S260 with the optimization option set to Balanced. For performance comparison, the continuous model and the discrete model are also converted into C language programs by Real-Time Workshop.

II. Tracking Control System with Vibration Control

Noda et al. [8] [9] [10] discussed a design of self-transfer-type automatic pouring system in casting industry. This system synchronizes the movement of the ladle to the movement of the mold line, while suppressing the residual vibration and the sloshing (liquid vibration) of the molten metal in the ladle. Sloshing suppression is important in an automatic pouring system, since the sloshing causes the overflow of molten metal or the contamination that leads to the degradation of the product quality. Hence, vibration control is essential in the tracking control system of this application.

Figure 2 illustrates the block diagram of the corresponding control system. To fulfill the requirements of tracking to the target system, a two-degrees-of-freedom control system is applied to the tracking system. The blocks Km and Kx in Figure 2 are the feedback controllers based on Hybrid Shaped Approach (HSA) [11] [12]. Pm and Px are the transfer functions of motors, while Px^{-1} is the inverse model of Px. The position of the target system is applied to Px^{-1} for the feed-forward control of the tracking system. Sx is the sloshing model of the second order system to estimate the vibration in liquid container. Kv is a time varying gain filter, which suppresses the sloshing at the beginning of tracking to the target system [8].

This tracking control system has two degrees of freedom: feed-forward control and feedback control. Each controller is designed to fulfill the device requirements (maximal velocity and maximal acceleration), while the overall system may fail these requirements since both controllers are activated simultaneously. To avoid the violation, a simple predictive control is adopted. This system predicts the future velocity and acceleration from the current information by using state space model. If the requirements are to be satisfied until the end of tracking, the system starts tracking. If any violation is predicted, the system waits until the violation is avoided.

It is not essential to implement the whole tracking system (Fig. 2) with an FPGA device, though it is possible. Most of the system operates at a relatively low frequency (sampling

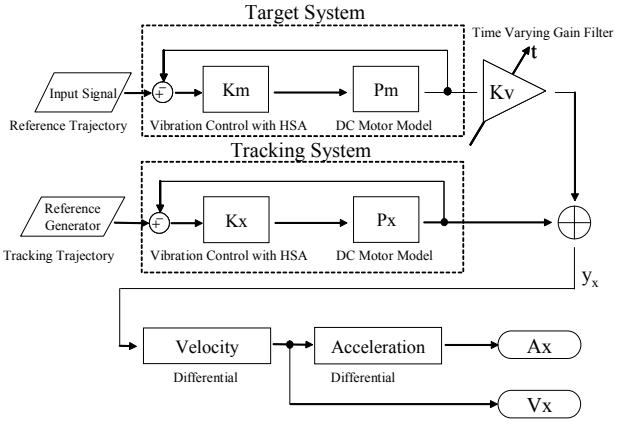

Fig. 3. y_x prediction model.

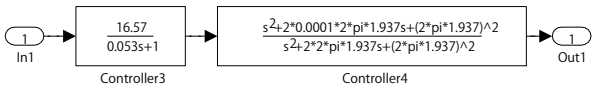

Fig. 4. Km block.

rate), which can be well maintained by software. The computationally demanding part of this system is the predictive control, where the future behavior must be predicted in a single sampling period. If the computational performance of the system increases by hard-wired implementation, we can finish the prediction in a shorter time period or can broaden the horizon of prediction in the same time period. Thus, this study focuses on the hard-wired implementation of predictive control part.

Figure 3 is the block diagram of predictive control part of the system. The Px^{-1} block in Fig. 2 is necessary for feed-forward control of the tracking motor, while it is cancelled out by Px in the prediction system (Fig. 3). The Sx block can be also omitted in the prediction system, since the sloshing is not necessary to predict the coordinate y_x of the tracking system. In the following discussion, the input waveform is assumed to be a sine wave of 0.23 Hz as in the previous studies [8] [9].

The blocks Km and Kx are the HSA controllers, each of which consists of a low-pass filter and a notch filter. Figure 4 illustrates the transfer functions of two filters in Km block, where the filter parameters are fixed as constant values. Since the logic scale and delay time of digital filter are generally reduced by fixing filter parameters as constants, Km and Kx were optimized with a given set of parameters (*hardware specialization*). More details on this topic are found in our previous work [13], which summarizes the quantitative evaluation results of hardware specialization techniques in digital filter implementation.

Pm and Px are motor models, each of which consists of a transfer function and an integrator. Figure 5 illustrates the block diagram of Pm. Time varying gain filter block (Kv in Fig. 3) increases its gain smoothly from 0 to 1 with a quintic

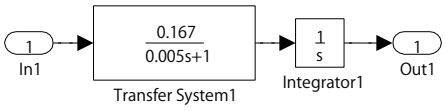

Fig. 5. Pm block.

function of time (t). Kv was implemented as described in the previous study [8], which describes every details of Kv.

Though the predictive control part outputs the coordinate (y_x), the purpose of the prediction is to check the predicted velocity and acceleration against the device constraints. Therefore, the predicted coordinate y_x has to be differentiated to calculate the velocity Vx and the acceleration Ax (Fig. 3). In general, numerical differentiation is prone to be affected by external noise or loss of significance, and should be avoided if possible. In this particular case, our simulations demonstrated that the errors of three-point centered difference are insignificant if the width of data format is sufficient. In case that the output of numerical differentiation is unstable, Vx and Ax might also be calculated by state space models in exchange for the increase of logic scale of the circuit.

III. Design and Evaluation of Predictive Control Part

Figure 6 is the Simulink model that corresponds to the predictive control part (Fig. 3). The tracking system shown in Fig. 2 processes the actual position of target system, which is derived from encoder or laser sensor. Meanwhile, the predictive control part cannot depend on the external input; the future target position is thus internally estimated (target reference trajectory). The Reference Generator (RG) of Figure 6 generates this transfer reference trajectory by integrating the maximum velocity $v_R = 0.3\ [m/s]$ and the maximum acceleration $a_R = 1.0\ [m/s^2]$. RG block was designed as shown in the previous study [9], and the reference trajectory was set to a sine wave of 0.23 Hz. The integrator at the input of Kv outputs the elapse time from the beginning.

The predictive control system shown in Fig. 6 is a continuous model, where data and time are both continuous. To implement a hard-wired control system, this continuous model has to be discretized. Discretization itself is not difficult because Simulink provides various tools for discretization. The actual problem is that HDL Coder and Real-Time Workshop support only a limited variation of blocks; i.e., HDL Coder and Real-Time Workshop cannot convert the discrete model discretized by Simulink as it is. Thus, the discretized model has to be modified to use the supported set of blocks only. Since HDL Coder and Real-Time Workshop support different sets of blocks, two equivalent (but different) discrete models have to be prepared for each of HDL Coder and Real-Time Workshop.

A. Performance of Software Implementation

Figure 7 displays the discrete model for Real-Time Workshop, which is convertible to C program. Data type option was

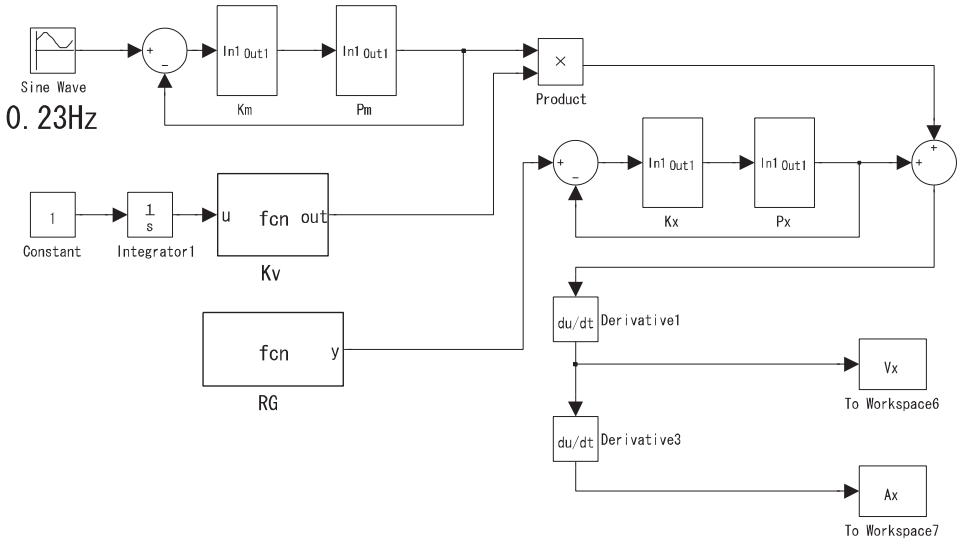

Fig. 6. The Simulink model of predictive control part.

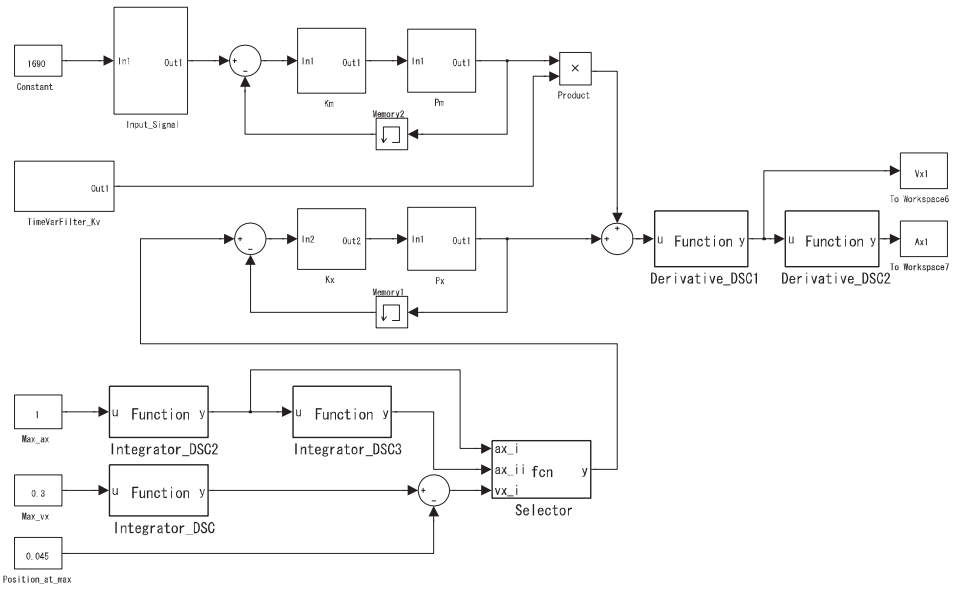

Fig. 7. The discrete model of predictive control part for Real-Time Workshop.

set to automatic in conversion, and this discrete model was verified to be fully functional with a discrete solver. Major differences from the continuous model are listed below.

- Numerical differentiator and integrator were replaced by the corresponding MATLAB embedded function block, which was described procedurally.
- A memory block was inserted in each feedback loop.
- Each filter of continuous-time signal was replaced by the corresponding IIR digital filter. [13]

The above-mentioned continuous model (Fig. 6) and discrete model (Fig. 7) were actually converted into the corresponding C codes by Real-Time Workshop on MATLAB R2008a. System target was set to GRT (general real-time), and data type option was set to automatic. The solver for continuous model was set to ode4 (Runge-Kutta), while the discrete solver was used for discrete model. Time step was fixed to 1/512 [s] for both models.

The C programs (continuous and discrete) were compiled with LCC compiler, and then executed on Intel Atom Z530 processor (1.6 GHz clock) with 1 GB main memory and Windows XP SP3. Though Atom Z530 is not a high-performance processor, it is far above the average of embedded processors as of 2010. The execution time was measured with `timeit` command as the average of 100 trials. Figure 8 plots the

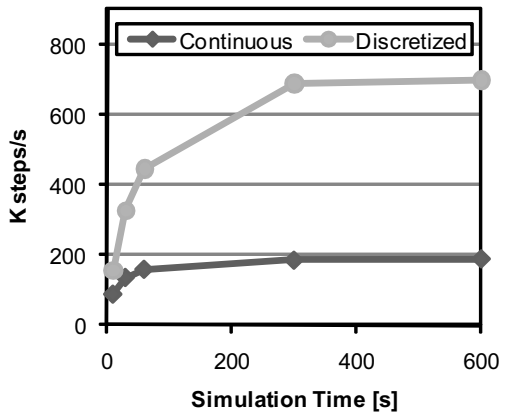

Fig. 8. The performance of software implementations.

performance for various number of simulation steps. When the number of simulated steps is small, the performance is low on account of initial overhead. For the simulation of 300 [s] or longer, the overhead becomes negligible and the performance saturates. In the following discussion, the performance of 600 [s] is adopted as the peak performance of software implementation; i.e., 698 [K steps/s] for the discrete model and 189 [K steps/s] for the continuous model.

B. Performance of Hard-wired Implementation

Figure 9 illustrates the discrete model for HDL Coder, whose major differences from the continuous model are listed below.

- Numerical differentiator was replaced by a simple Simulink block diagram of three-point centered difference. Numerical integrator was replaced by a Simulink block diagram of summation.
- A unit delay block was inserted in each feedback loop.
- Each filter of continuous-time signal was replaced by the corresponding IIR digital filter. [13]

This discrete model was converted into VHDL codes by HDL Coder, where data type was set to fixed-point number (1 bit for sign, 15 bit for integer, and 24 bit for fraction). The generated VHDL codes were processed by Altera Quartus II 9.0 for the target device Stratix III (EP3S260). The locations of I/O pins were not designated.

Table I summarizes the evaluation results. The numbers in parentheses designate the resource usages in the target device. The circuit requires a considerable amount of multiplier resources with few logic resources and no memory. This circuit is expected to fit into a middle class device of Stratix III FPGA or a low-end device of Stratix IV. Elapsed time shown in Table I is the time required to generate the corresponding FPGA design with an Athlon 64 X2 4400+ (2GB RAM, Windows XP3).

C. Performance Comparison

In a hard-wired predictive control unit, the prediction of a simulation step is calculated in one clock cycle. The peak

TABLE I
THE EVALUATION RESULTS OF HARD-WIRED CONTROL CIRCUIT ON EP3S260F1157C2 DEVICE.

ALUTs	8683	(4%)
Logic registers	432	(<1%)
I/O Pins	100	(13%)
DSP 18-bit elements	312	(41%)
Block memory	0	(0%)
Clock [MHz] (85–0°C)	15.14–16.38	
Elapsed Time [min]	83	

performance of hard-wired unit is hence estimated to be 15.14–16.38 [M steps/s]; this is 21.7–23.5 times higher than that of the discrete model software and 80.1–86.7 times higher than that of the continuous model software. Considering that the design of this circuit was automatically generated from the Simulink model with minimal modification, the derived performance looks satisfactory.

In this tracking system, the maximal time for rendezvous can be predetermined. Since the target trajectory is periodic (a sine wave of 0.23 Hz), we can find the worst-case time by examining the rendezvous time exhaustively over one cycle [8]. As a result, every tracking is known to finish in a single cycle (4.3 seconds) under the conditions of this study. Therefore, it is sufficient for the predictive control part to simulate 2200 steps (4.3 seconds of 512 Hz sampling) in one sampling period (1/512 [s]); i.e., the required performance for real-time predictive control is approximately 1.1 M steps/s.

The peak performance of hard-wired predictive control is 15–16 [M steps/s], which well satisfies the above requirement. Meanwhile, the peak performance of discrete software is no more than 0.698 [M steps/s]. The effective performance is 0.155 [M steps/s] for the simulation of 10 [s], and it will be even worse for the simulation of 4.3 [s] (one cycle). It looks very difficult to build a real-time predictive control unit with software implementation.

IV. CONCLUSION

In this study, a practical control system developed with MATLAB/Simulink was actually converted with Simulink HDL Coder into the corresponding VHDL description, which was implemented and evaluated with a common FPGA device. The derived hard-wired control circuit operated at 15 MHz clock on an Altera Stratix III FPGA, and achieved approximately 20 times higher performance than the corresponding software implementation on Intel Atom Z530 processor (1.6 GHz clock).

In this study, input waveform and many parameters were fixed (e.g., filter parameters), which simplified the design of hard-wired control logic considerably. There are many items to be examined in future studies.

For an example, the initial coordinate and velocity of tracking system were set to zero in this study according to the previous study [9]. The predictive control part thus simulates the tracking system based on the current states of the target system, and verifies whether the device requirements are met until the tracking system synchronizes with the target system.

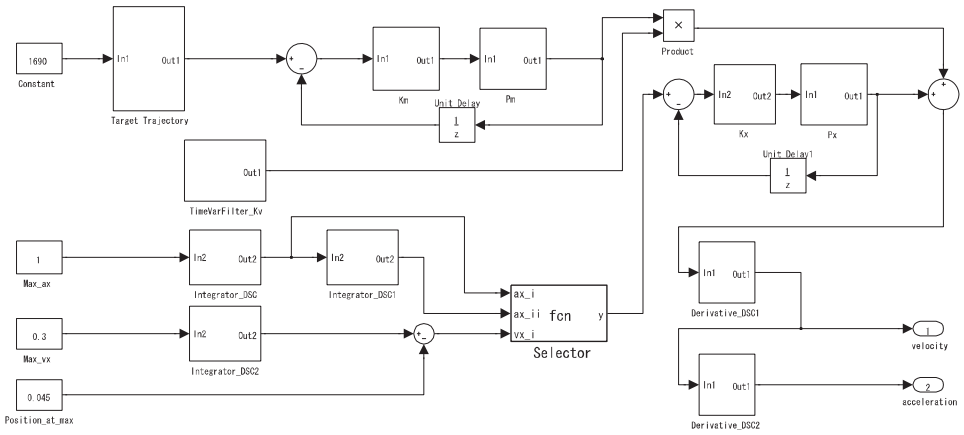

Fig. 9. The discrete model of predictive control part for HDL Coder.

If all requirements are met, the tracking system actually starts; if any requirement is violated, the tracking system stops starting and waits for the next sampling period.

Although the above-mentioned control is a very simple example of predictive control, general predictive control is more complex. For example, it is possible to control the output (e.g., output voltage to motor) for each sampling period by predicting near future from the current information of the sensors in tracking/target systems. In such a case, the resulting predictive control circuit becomes more complex and larger in logic scale.

In a more realistic situation, all necessary data cannot be derived from sensors. Some data may be affected by external noise, some data may not be directly observed, or the number of sensors may be limited by cost restrictions. In such cases, the internal state has to be estimated by other means: e.g., state observer, Kalman filter, etc. It is left for future studies to examine the feasibility of our approach for such general cases.

Data type is another important topic for future studies. Though fixed-point arithmetic was adopted in this study, it is inevitable to adopt floating-point arithmetic for some applications. Although our current tools (HDL Coder and Quartus) do not support floating-point arithmetic, it might be supported in the future versions. The implementation of hard-wired control logic with floating-point arithmetic should be seriously considered for practical applications.

ACKNOWLEDGMENT

This work was partially supported by JSPS KAKENHI (19500042).

REFERENCES

[1] MathWorks, "Simulink - simulation and model-based design," http://www.mathworks.com/products/simulink/.
[2] ——, "MATLAB - the language of technical computing," http://www.mathworks.com/products/matlab/.
[3] ——, "Real-Time Workshop," http://www.mathworks.com/products/rtw/.
[4] ——, "Simulink HDL Coder," http://www.mathworks.com/products/slhdlcoder/.
[5] Xilinx, "System Generator for DSP," http://japan.xilinx.com/tools/sysgen.htm.
[6] Altera, "DSP Builder/Simulink," http://www.altera.co.jp/technology/dsp/dsp-builder/dsp-simulink.html.
[7] Synopsys, "Synplify DSP," http://www.synopsys.co.jp/products/synplifydsp/.
[8] Y. Noda, K. Yano, T. Miyoshi, and K. Terashima, "Tracking control of vibrating systems to moving object with considering sloshing suppression," *SICE Transactions*, vol. 40, no. 4, pp. 424–433, 2004.
[9] Y. Noda, K. Yano, and K. Terashima, "Automatic detection to moving target object and tracking control with considering constraints of velocity and acceleration and vibration damping," *SICE Transactions*, vol. 42, no. 3, pp. 265–274, 2006.
[10] ——, "Tracking control to moving object of liquid container transfer with vibration damping," in *Proc. 2002 IEEE International Conference on Control Applications*, vol. 1, 2002, pp. 582–587.
[11] K. Yano, T. Toda, and K. Terashima, "Sloshing suppression control of automatic pouring robot by hybrid shape approach," in *Proc. 40th IEEE Conference on Decision and Control 2001*, 2001, pp. 1328–1333, vol.2.
[12] K. Yano, S. Higashikawa, and K. Terashima, "Liquid container transfer control on 3D transfer path by hybrid shaped approach," in *Proc. 2001 IEEE International Conference on Control Applications (CCA 2001)*, 2001, pp. 1168–1173.
[13] Y. Tezuka, S. Ichikawa, and Y. Noda, "Evaluation of the hardware specialization techniques for vibration control applications," in *Proc. 2010 IEEE Symposium on Industrial Electronics & Applications (ISIEA 2010)*, 2010, (to appear).