An FPGA-based multiple-axis motion control chip

Article *in* IEEE Transactions on Industrial Electronics · April 2009

DOI: 10.1109/TIE.2008.2004671 · Source: IEEE Xplore

An FPGA-Based Multiple-Axis Motion Control Chip

Jung Uk Cho, Quy Ngoc Le, and Jae Wook Jeon, *Member, IEEE*

***Abstract*—This paper presents the design and implementation of a multiple-axis motion control chip using a field-programmable gate array (FPGA). This multiple-axis motion control chip is designed to control a multiple-axis motion system such as a robotic arm manipulator or a computer numerical control machine. The proposed motion control chip has many functions. These include velocity profile generation, interpolation calculation, inverse kinematics calculation, proportional–integral–derivative control, feedback count, pulse integration, data conversion, clock generation, and external interfacing. These functions are designed using the VHSIC hardware description language and implemented on an FPGA according to the electronic design automation design methodology. This allows for a highly sampled, accurate, flexible, compact, low-power, and low-cost motion control system. The detailed design of the proposed motion control chip is presented. A multiple-axis motion control system using this chip is implemented, and its performance is measured. The multiple-axis motion control system is implemented on a platform consisting of a chip-based multiple-axis motion controller, analog ac servo drivers, a selective compliant assembly robot arm robot, and a host personal computer.**

***Index Terms*—Field-programmable gate arrays, manipulators, motion control, multiple-axis motion, VHSIC hardware description language (VHDL).**

I. Introduction

MOTION control systems play a major role in the control of different types of industrial automation devices such as robotic arm manipulators, computer numerical control (CNC) machines, and assembly machines [1], [2]. Most motion control systems use a motion controller that is based on a digital signal processor (DSP) and a microprocessor, and the systems include software motion libraries and additional circuits such as interface and memory [2], [3]. A multiple-axis motion control system needs many functions in order to quickly and accurately perform the complex tasks required for industrial robots and automation systems [4], [5]. These include velocity profile generation, interpolation, inverse kinematics, and proportional–integral–derivative (PID) control. Conventional numerical approaches for these functions have several difficult aspects. Complicated multiplicative, divisional, and trigonometric function calls are needed to perform repetitive calculations. To meet the desired accuracy and speed, a floating-point arithmetic processor and a sufficiently wide data path are usually required. This requirement leads to a complex hardware design, expensive computation, increased system size, high cost, and high power consumption [6]. In the development of very large scale integration technology, the field-programmable gate array (FPGA) has been widely used to implement digital systems because of its simplicity, programmability, short design cycle, fast time-to-market, low power consumption, and high density [7], [8]. An FPGA provides a compromise between an application-specific integrated circuit (ASIC) and a general-purpose processor [9]. In contrast, DSPs and personal computers (PCs) have pipelined architectures. The computing time of an FPGA-based controller can be relatively short regardless of the complexity of the control algorithm because of its parallel processing architecture. A motion controller can be implemented using a single FPGA chip. Therefore, a compact system with low power and a simple circuit is possible. The price of the motion control system can be reduced with ASIC manufacturing.

The design and implementation of a high-performance ac servo system using the FPGA was presented by Takahashi and Goetz [10]. Shao *et al.* [1] presented a new motion control hardware architecture using the FPGA for position and velocity control, and a DSP was used for dynamic compensation, inverse kinematics, and trajectory generation. The design and implementation of an FPGA-based controller that employs 3-D dynamic interpolation for dynamic process and system compensation was presented by Oldknow and Yellowley [11]. It was applied to a two-axis test stand (X–Y table) in the experiment. An FPGA-based motion controller was developed by Yau *et al.* [12] in order to realize the real-time nonuniform rational B-spline (NURBS) interpolator and CNC controller in an FPGA. The NURBS interpolation algorithm and the infinite impulse response filter algorithm were implemented to control an X–Y table using an FPGA-based motion controller. Tzou and Kuo [9] presented the design and implementation of a motor control IC for the permanent magnet ac (PMAC) servo. A DSP was used for system initialization and parameter setting. Park and Oh [6] presented the hardware realization of inverse kinematics for robot manipulators. The system was applied only to straight line interpolation. A servo control IC for the X–Y table was developed by Kung *et al.* [8]. This system used a soft-core embedded processor to perform motion trajectory and position control. There are new methodologies for FPGA-based design. Lienhardt *et al.* [13] proposed an observer technique and the use of the Saber cosimulation tool to validate the design. Lin-Shi *et al.* [14] presented a hybrid control for a

Manuscript received October 2, 2007; revised August 7, 2008. First published September 12, 2008; current version published February 27, 2009. This work was supported in part by the Ministry of Education and Human Resources Development (MOE), in part by the Ministry of Commerce, Industry and Energy (MOCIE), and in part by the Ministry of Labor (MOLAB), through the fostering project of the Laboratory of Excellency project.

J. U. Cho was with the School of Information and Communication Engineering, Sungkyunkwan University, Suwon 440-746, Korea. He is now with the Department of Computer Science and Engineering, University of California at San Diego, La Jolla, CA 92093-0404 USA (e-mail: jucho@cs.ucsd.edu).

Q. N. Le and J. W. Jeon are with the School of Information and Communication Engineering, Sungkyunkwan University, Suwon 440-746, Korea (e-mail: quy1001@ece.skku.ac.kr; jwjeon@yurim.skku.ac.kr).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TIE.2008.2004671

permanent-magnet synchronous motor, implemented with a CPU-core inside an FPGA device. Chan *et al.* [4] showed a new way to make a PID controller using distributed arithmetic (DA). This DA-based PID controller considers the optimal resource utilization and power consumption when implementing the controller in an FPGA. Naouar *et al.* [15] reviewed various methods for implementing a current controller in an FPGA that show preeminently fast computation times. Ishii *et al.* [16] proposed an FPGA implementation that contains position synchronization and force control for a remotely operated system. Since a high sampling rate is achieved, the bandwidth of the force estimator and the performance of the control system are improved. In the research listed to date, FPGAs were used to realize only the logic circuits of motion control systems or particular functions of the motion control systems such as interpolation, velocity profile generator, PID controller, and inverse kinematics calculator. This paper proposes a multiple-axis motion control chip incorporating the generation of various velocity profiles, interpolation, inverse kinematics, PID control, feedback count, pulse integration, data conversion, clock generation, and external interface. These functions are required for a general multiple-axis motion control system. They are designed using VHSIC hardware description language (VHDL) and implemented on an FPGA.

The proposed multiple-axis motion control chip consists of nine modules: velocity profile generator, interpolation calculator, inverse kinematics calculator, PID controller, feedback counter, pulse integrator, data converter, clock generator, and external interface. The velocity profile generator module can efficiently generate velocity profiles with the desired acceleration and deceleration characteristics. The interpolation calculation module makes the robotic arm move to the desired location without any sharp jerks. The inverse kinematics calculation module can deduce the angles of the joints corresponding to a given location on the tip of the robotic arm. The PID control module controls the position and velocity of the motors. The feedback counter module calculates the current motor position. The pulse integrator calculates the integrated pulse during sampling. The data converter module converts the output of the circuit into the driving signals for servo drivers of several types. The clock generator module provides various clock signals for several modules in the system. The external interface module communicates with the host computer through the serial cable. These modules of the multiple-axis motion control chip are designed using VHDL and implemented on an FPGA according to the electronic design automation design methodology. This allows for a highly sampled, accurate, flexible, compact, low-power, and low-cost motion control system. A multiple-axis motion control system with this chip is implemented, and its performance is measured. This system is developed to control a multiple-axis motion system such as a robotic arm manipulator or a CNC machine. The multiple-axis motion control system is implemented on a platform consisting of a chip-based multiple-axis motion controller, analog ac servo drivers, a selective compliant assembly robot arm (SCARA) robot, and a host PC.

This paper is organized as follows. In Section II, a technique that can efficiently generate velocity profiles with the desired acceleration and deceleration characteristics is explained briefly. The design of this velocity profile generation module is described. In Section III, a technique that can efficiently perform the interpolation and the design of the interpolation calculation module are explained. In Section IV, a technique that can efficiently calculate the inverse kinematics of motors for multiple axes is explained briefly, and then, the design of this inverse kinematics calculation module is described. In Section V, the functions of each internal module of the proposed multiple-axis motion control chip are explained in detail. In Section VI, the proposed multiple-axis motion control chip is implemented with an FPGA. A chip-based multiple-axis motion control system is built using the proposed multiple-axis motion control chip, and it is used to control a robotic arm manipulator. Various velocity profiles and interpolations generated by the proposed multiple-axis motion control chip are applied to the multiple-axis motion control system. The corresponding results are shown and measured. Our conclusion is presented in Section VII.

II. Velocity Profile Generation

A. Technique to Generate Velocity Profiles With the Desired Characteristics

Let us consider a single-axis control system of which the maximum velocity and the sampling time are $V_{\max}$ and T_{s}, respectively. The acceleration time, constant velocity time, and deceleration time are T_{a}, T_{c}, and T_{d}, respectively. The motor velocity should increase and decrease following the acceleration and deceleration values f_{a} and f_{d}. These are functions of time, so

$$V(t) = \begin{cases} V_{\max} f_{\mathrm{a}}(t/T_{\mathrm{a}}), & \text{for } 0 < t \le T_{\mathrm{a}} \\ V_{\max}, & \text{for } T_{\mathrm{a}} < t \le T_{\mathrm{c}} \\ V_{\max} f_{\mathrm{d}}\left((t-T_{\mathrm{c}})/T_{\mathrm{d}}\right), & \text{for } T_{\mathrm{c}} < t \le T_{\mathrm{d}}. \end{cases} \tag{1}$$

Velocity is integrated in (1) during each sampling interval; to get the distance, the motor must move in each sampling interval, as shown in (2) at the bottom of the page. If setting variable $u = t/T_{\mathrm{a}}$, then

$$\begin{cases} u = t/T_{\mathrm{a}} \rightarrow dt = T_{\mathrm{a}} du, & \text{for acceleration time} \\ u = (t-T_{\mathrm{c}})/T_{\mathrm{d}} \rightarrow dt = T_{\mathrm{d}} du, & \text{for deceleration time.} \end{cases} \tag{3}$$

$$\delta P(kT_{\mathrm{s}}) = \begin{cases} \int_{(k-1)T_{\mathrm{s}}}^{kT_{\mathrm{s}}} V_{\max} f_{\mathrm{a}}(t/T_{\mathrm{a}}) dt, & \text{for } 0 < t \le T_{\mathrm{a}} \\ \int_{T_{\mathrm{a}}+(k-1)T_{\mathrm{s}}}^{T_{\mathrm{a}}+kT_{\mathrm{s}}} V_{\max} dt, & \text{for } T_{\mathrm{a}} < t \le T_{\mathrm{c}} \\ \int_{T_{\mathrm{c}}+(k-1)T_{\mathrm{s}}}^{T_{\mathrm{c}}+kT_{\mathrm{s}}} V_{\max} f_{\mathrm{d}}\left((t-T_{\mathrm{c}})/T_{\mathrm{d}}\right) dt, & \text{for } T_{\mathrm{c}} < t \le T_{\mathrm{d}} \end{cases} \tag{2}$$

Substituting (3) into (2), $\delta P(kT_\mathrm{s})$ is expressed as (4), shown at the bottom of the page, where

$$\begin{cases} {}^a\gamma_k = n_\mathrm{a}\int_{(k-1)/n_\mathrm{a}}^{k/n_\mathrm{a}} f_\mathrm{a}(u)du, & \text{for } 0 < u \le 1 \\ {}^d\gamma_k = n_\mathrm{d}\int_{(k-1)/n_\mathrm{d}}^{k/n_\mathrm{d}} f_\mathrm{d}(u)du, & \text{for } 0 < u \le 1 \\ n_\mathrm{a} = T_\mathrm{a}/T_\mathrm{s} \\ n_\mathrm{d} = T_\mathrm{d}/T_\mathrm{s}. \end{cases} \tag{5}$$

These equations prove the concept of this method of generating the velocity profile. The coefficients ${}^a\gamma_k$ and ${}^d\gamma_k$ can be precomputed and stored with a given set of input parameters consisting of V_max, T_a, T_d, T_s, the total desired move distance S, the desired acceleration characteristic f_a, and the desired deceleration characteristic f_d. When the motor is running, the distance that the motor must move in one sampling interval is determined by (4) with only one multiplication.

With this input set of parameters, two parameters remain to be computed. They are the interval over which the motor moves with constant velocity T_c and the maximum motor velocity V_m. V_m is usually smaller than the desired input velocity V_max. To compute the time interval during which the motor moves at constant velocity, the distances that are covered while the motor is accelerating (S_a) and decelerating (S_d) are

$$\begin{cases} S_\mathrm{a} = \int_0^{T_\mathrm{a}} V_\mathrm{max} f_\mathrm{a}(t/T_\mathrm{a})dt, & \text{for } 0 < t \le T_\mathrm{a} \\ S_\mathrm{d} = \int_{T_\mathrm{c}}^{T_\mathrm{c}+T_\mathrm{d}} V_\mathrm{max} f_\mathrm{d}\left((t-T_\mathrm{c})/T_\mathrm{d}\right)dt, & \text{for } T_\mathrm{c} < t \le T_\mathrm{d}. \end{cases} \tag{6}$$

Substituting (3) into (6)

$$\begin{cases} S_\mathrm{a} = V_\mathrm{max}T_\mathrm{s}n_\mathrm{a}\int_0^1 f_\mathrm{a}(u)du = \alpha_\mathrm{a}n_\mathrm{a}V_\mathrm{max}T_\mathrm{s}, & \text{for } 0<u\le 1 \\ S_\mathrm{d} = V_\mathrm{max}T_\mathrm{s}n_\mathrm{d}\int_0^1 f_\mathrm{d}(u)du = \alpha_\mathrm{d}n_\mathrm{d}V_\mathrm{max}T_\mathrm{s}, & \text{for } 0<u\le 1 \end{cases} \tag{7}$$

where

$$\begin{cases} \alpha_\mathrm{a} = \int_0^1 f_\mathrm{a}(u)du, & \text{for } 0 < u \le 1 \\ \alpha_\mathrm{d} = \int_0^1 f_\mathrm{d}(u)du, & \text{for } 0 < u \le 1. \end{cases} \tag{8}$$

Equation (8) can be computed with a given set of input parameters, and then, the interval during which the motor moves with constant velocity T_c is

$$T_\mathrm{c} = T_\mathrm{s}N, \qquad \text{where } N = \left[\frac{S}{V_\mathrm{max}T_\mathrm{s}} - \alpha_\mathrm{a}n_\mathrm{a} - \alpha_\mathrm{d}n_\mathrm{d}\right]. \tag{9}$$

Then, the actual maximum velocity V_m of the motor is recomputed by

$$V_\mathrm{m}T_\mathrm{s} = \begin{cases} \frac{S}{\alpha_\mathrm{a}n_\mathrm{a}+\alpha_\mathrm{d}n_\mathrm{d}}, & \text{if } N \le 0 \\ \frac{S}{N+\alpha_\mathrm{a}n_\mathrm{a}+\alpha_\mathrm{d}n_\mathrm{d}}, & \text{if } N > 0. \end{cases} \tag{10}$$

In conclusion, when the motor is running, the distance that the motors move in each sampling interval is as follows. If $N \le 0$,

$$\delta P(kT_\mathrm{s}) = \begin{cases} {}^a\gamma_k V_\mathrm{m}T_\mathrm{s}, & 1 \le k \le n_\mathrm{a} \\ {}^d\gamma_{(k-n_\mathrm{a})} V_\mathrm{m}T_\mathrm{s}, & n_\mathrm{a}+1 \le k \le n_\mathrm{a}+n_\mathrm{d}. \end{cases} \tag{11}$$

If $N > 0$,

$$\delta P(kT_\mathrm{s}) = \begin{cases} {}^a\gamma_k V_\mathrm{m}T_\mathrm{s}, & 1 \le k \le n_\mathrm{a} \\ V_\mathrm{m}T_\mathrm{s}, & n_\mathrm{a}+1 \le k \le n_\mathrm{a}+N \\ {}^d\gamma_{(k-n_\mathrm{a}-N)} V_\mathrm{m}T_\mathrm{s}, & n_\mathrm{a}+N+1 \le k \le n_\mathrm{a}+N+n_\mathrm{d}. \end{cases} \tag{12}$$

This velocity-profile-generating technique was implemented to generate various velocity profiles [17]–[19].

B. Velocity Profile Generation Module

The velocity profile generation module in the proposed multiple-axis motion control chip can efficiently generate any desired velocity profile. Fig. 1 shows the architecture of the velocity profile generation module. Its inputs, namely, V_max and S, represent the maximum velocity and the desired distance to move, respectively. The other input, namely, C_v, represents the intervals and characteristics of acceleration and deceleration. Its outputs, namely, p and *dir*, represent the command pulse and the rotation direction of the motor, respectively. The parameters *clk* and *cpg_clk* represent the input clock and the clock for the command pulse generator block, respectively. The other parameters, namely, *reset*, *enable*, and *latch*, represent the control signals [20].

The sign calculator block calculates the rotation direction of the motor *dir* and the desired distance magnitude $|S|$ from the desired distance S. The peak velocity generator block calculates the velocity after calculating the acceleration V_m, the distance during the deceleration S_d, and the initial and final values of the counter block. The velocity after the acceleration V_m is calculated as described in (9) and (10). The division operations in the peak velocity generator block are implemented using a pipelined divider intellectual property (IP) core, supplied by the Xilinx Company. The distance during the deceleration is calculated as $S_\mathrm{d} = \alpha_\mathrm{d}n_\mathrm{d}V_\mathrm{m}T_\mathrm{s}$ from (7). The initial and final values of the counter block are determined from the desired acceleration and deceleration characteristics. The counter block generates the address of the rate factor generator block. The address is increased and decreased during acceleration and deceleration, respectively, and it remains constant during a uniform velocity interval. The rate factor generator block stores the product of each desired characteristic coefficient γ_k and 128 in order to avoid a floating-point calculation and sends each product value to the position increment block. The position increment block

$$\delta P(kT_\mathrm{s}) = \begin{cases} V_\mathrm{max}T_\mathrm{s}n_\mathrm{a}\int_{(k-1)/n_\mathrm{a}}^{k/n_\mathrm{a}} f_\mathrm{a}(u)du = {}^a\gamma_k V_\mathrm{m}T_\mathrm{s}, & \text{for } 0 < u \le 1 \\ V_\mathrm{max}T_\mathrm{s}, & \text{for } T_\mathrm{a} < t \le T_\mathrm{c} \\ V_\mathrm{max}T_\mathrm{s}n_\mathrm{d}\int_{(k-1)/n_\mathrm{d}}^{k/n_\mathrm{d}} f_\mathrm{d}(u)du = {}^d\gamma_k V_\mathrm{m}T_\mathrm{s}, & \text{for } 0 < u \le 1 \end{cases} \tag{4}$$

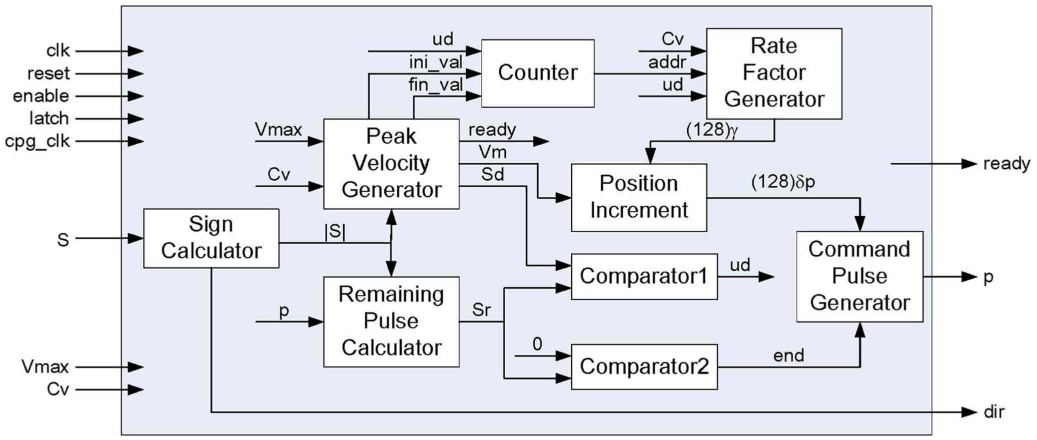


Fig. 1. Velocity profile generation module. It consists of a sign calculator, a peak velocity generator, a counter, a rate factor generator, a position increment, a remaining pulse calculator, two comparators, and a command pulse generator.

calculates the position increment during each sampling time, as described in (11) and (12). The output of the position increment block is the product of a real position increment and 128, because the output of the rate factor generator block is the product of a coefficient and 128. The remaining pulse calculator block calculates the remaining distance S_{r} by decrementing $|S|$ by each command pulse p. The *comparator1* block compares this remaining distance with the distance during deceleration S_{d} in order to detect when the deceleration starts. The *comparator2* block compares this remaining distance with the zero value in order to detect when the motor stops. The command pulse generator block produces the command pulse p. This block is similar to the digital differential analyzer (DDA) and performs a division by 128 in order to compensate for the multiplication by 128 in the rate factor generator block [21].

III. Interpolation Calculation

A. Technique to Calculate Interpolations

A motion control system on a robotic arm manipulator uses a specific control method to guide the robotic arm to the desired locations. One of the requirements of such a control system is that it designates a smooth path for the arm to move from one location to another, in order to avoid any sudden jerky motion in the arm that might cause objects to slip out of its grasp or damage the object or the arm itself. At the same time, the path needs to be short. Therefore, the path for the arm is initially designed in terms of the number of points over which the arm is to move. Then, interpolation is used to design a smooth curve that will contain all of the points.

Generally, linear interpolation takes two points, for example, (x_i, y_i) and (x_j, y_j), and the interpolation formula is given by

$$y_p = \frac{y_j - y_i}{x_j - x_i}(x_p - x_i) + y_i. \tag{13}$$

Circular interpolation takes the center coordinates and the radius of a circle, given the point (x_i, y_i) and the radius r, and the interpolation formula is given by

$$(x_p - x_i)^2 + (y_p - y_i)^2 = r^2 \tag{14}$$

$$x_p = \cos(\theta)\cdot r + x_i \qquad y_p = \sin(\theta)\cdot r + y_i. \tag{15}$$

The goal of cubic spline interpolation is to get an interpolation formula that is smooth in the first derivative and continuous in the second derivative, both within an interval and at its boundaries. Given the points (x_i, y_i), $i = 1, \ldots, N$, attention is focused on one particular interval, between x_j and x_{j+1}. Cubic spline interpolation in that interval gives the interpolation formula

$$y = Ay_j + By_{j+1} + Cy''_j + Dy''_{j+1} \tag{16}$$

where

$$A \equiv \frac{x_{j+1} - x}{x_{j+1} - x_j} \qquad B \equiv \frac{x - x_j}{x_{j+1} - x_j} \tag{17}$$

$$C \equiv \frac{1}{6}(A^3 - A)(x_{j+1} - x_j)^2$$

$$D \equiv \frac{1}{6}(B^3 - B)(x_{j+1} - x_j)^2. \tag{18}$$

Note that the dependence on the independent variable x in (16) and (17) is entirely through the linear x-dependence of A and B and (through A and B) the cubic x-dependence of C and D. We take derivatives of (16) with respect to x, using the definitions of A, B, C, and D to compute dA/dx, dB/dx, dC/dx, and dD/dx. The result is

$$\frac{dy}{dx} = \frac{y_{j+1} - y_j}{x_{j+1} - x_j} - \frac{3A^2 - 1}{6}(x_{j+1} - x_j)y''_j + \frac{3B^2 - 1}{6}(x_{j+1} - x_j)y''_{j+1} \tag{19}$$

for the first derivative, and

$$\frac{d^2y}{dx^2} = Ay''_j + By''_{j+1} \tag{20}$$

for the second derivative. The second derivative will be continuous across the boundary between the two intervals (x_{j-1}, x_j) and (x_j, x_{j+1}). The key idea of a cubic spline is to require this continuity and to use it to get equations for the second derivatives y''_i.

The required equations are obtained by setting (19) evaluated for $x = x_j$ in the interval (x_{j-1}, x_j) equal to the same equation

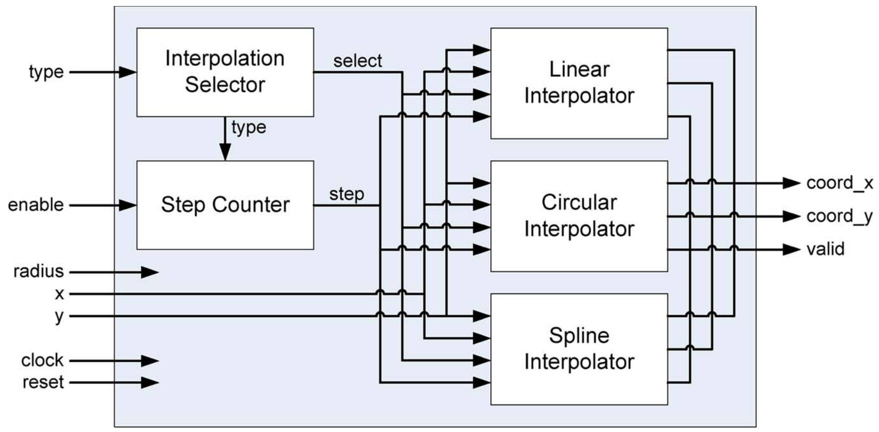


Fig. 2. Interpolation calculation module. It consists of an interpolation selector, a step counter, a linear interpolator, a circular interpolator, and a spline interpolator.

evaluated for $x = x_j$ but in the interval (x_j, x_{j+1}). With some rearrangement, this gives (for $j = 2, \ldots, N-1$)

$$\frac{x_j - x_{j-1}}{6} y''_{j-1} + \frac{x_{j+1} - x_{j-1}}{3} y''_j + \frac{x_{j+1} - x_j}{6} y''_{j+1} = \frac{y_{j+1} - y_j}{x_{j+1} - x_j} - \frac{y_j - y_{j-1}}{x_j - x_{j-1}}. \quad (21)$$

There are $N-2$ linear equations for the N unknowns y''_i, $i = 1, \ldots, N$. Therefore, there is a two-parameter family of possible solutions.

For a unique solution, we need to specify two further conditions, typically taken as boundary conditions at x_1 and x_N. The most common ways of doing this are either set one or both of y''_1 and y''_N equal to zero, giving the so-called natural cubic spline that has a zero second derivative on one or both of its boundaries, or set either of y''_1 and y''_N to the value calculated from (19) so as to make the first derivative of the interpolating function have a specified value on either or both boundaries.

One reason that cubic splines are particularly practical is that the set of (21), along with the two additional boundary conditions, is not only linear but also tridiagonal. Each y''_j is coupled only to its nearest neighbors at $j \pm 1$. Therefore, the equations can be solved in $O(N)$ operations by the tridiagonal algorithm [22].

B. Interpolation Calculation Module

The interpolation calculation module in the proposed multiple-axis motion control chip is based on the efficient interpolation technique which is explained in previous section. The interpolation calculation module consists of an interpolation selector, a step counter, a linear interpolator, a circular interpolator, and a spline interpolator (Fig. 2). Its inputs, namely, x, y, *radius*, and *type*, represent the target coordinates and radius for the interpolation, and the type of interpolation, respectively. The interpolation selector block generates the *select* signal in order to select the type of interpolation depending on the *type* signal. The step counter generates the *step* signal used to generate the coordinates of each interpolation depending on the *type* signal. x and y correspond to the coordinates of several points depending on the type of interpolation. *radius* corresponds to the radius of the circle when circular interpolation is used. For example, linear interpolation requires the coordinates of two points, circular interpolation requires the center coordinates and the radius of a circle, and spline interpolation requires the coordinates of five points. After all of the input values are loaded in the module, the *enable* signal is asserted in order to initiate the interpolation calculation. The selected interpolator block is operated according to the *step* signal in order to calculate the commanded interpolation. The results of the interpolation calculation, which consist of the coordinates of the smooth curve that contains all the points, are generated for the smooth path for the arm. The *valid* signal is able to obtain the valid values of the outputs *coord_x* and *coord_y*. These are the coordinates of each interpolation after the completion of the interpolation calculation. The values are multiplied by 128 to avoid floating-point calculations with a decimal fraction and to improve the accuracy of the operation. The output of the interpolation is used in the inverse kinematics calculation module; thus, all of the coordinates are transformed into the pulses that are required to rotate the robotic arm by the appropriate angle.

The interpolation technique includes many mathematical functions such as multiplication, division, and trigonometric functions. Since VHDL does not support these functions, it is necessary to develop these functions using VHDL. The multiplier and pipelined divider IP cores, supplied by the Xilinx Company, are used to design multiplication and division operations, respectively. The sine and cosine values are found using lookup tables. The value in degrees is selected according to the input values. The variables used in the interpolation equation are multiplied by 2048 (2^{11}) to avoid floating-point calculations with a decimal fraction and to improve the accuracy of the calculations. The multiplication by 2048 is compensated by 11 right shift operations in each coordinate calculator. Although the right shift operation is made 11 times, the total processing time is much less compared with a division by 2048, and the same result is achieved.

The linear interpolator block consists of a gradient calculator and a linear coordinate calculator (Fig. 3). The gradient calculator calculates the *gradient* value of the line that consists of two points. The linear coordinate calculator calculates all

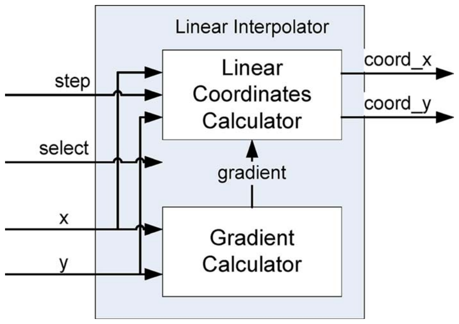


Fig. 3. Linear interpolation block. It consists of a gradient calculator and a linear coordinate calculator.

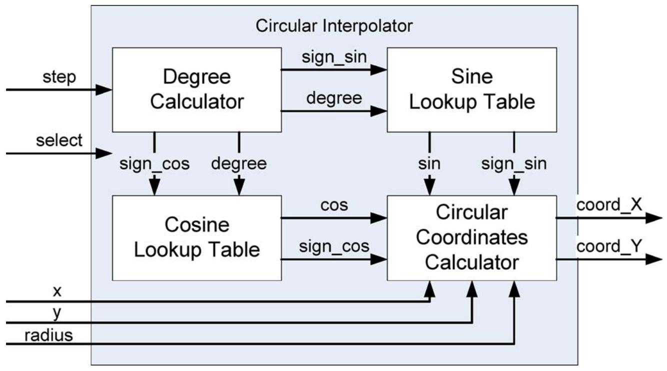


Fig. 4. Circular interpolation block. It consists of a degree calculator, a sine lookup table, a cosine lookup table, and a circular coordinate calculator.

coordinates, namely, *coord_x* and *coord_y*, of the line interpolation. The circular interpolator block consists of a degree calculator, sine and cosine lookup tables, and a circular coordinate calculator (Fig. 4). The degree calculator calculates the *degree* value of the circle depending on the *step* value. The sine and cosine lookup tables generate the sine value *sin*, the sign of the sine value *sign_sin*, the cosine value *cos*, and the sign of the cosine value *sign_cos*, respectively. The value in degrees is selected according to the input values. The circular coordinate calculator calculates all coordinates, namely, *coord_x* and *coord_y*, of the circular interpolation. The spline interpolator block consists of a lower boundary calculator, an upper boundary calculator, and a spline coordinate calculator (Fig. 5). The lower boundary condition is set either to be natural or to be a specified first derivate in the lower boundary calculator. The upper boundary condition is either natural or a specified first derivative in the upper boundary calculator. All coordinates, namely, *coord_x* and *coord_y*, of the spline interpolation are evaluated using the spline coordinate calculator.

IV. Inverse Kinematics Calculation

A. *Technique to Calculate Inverse Kinematics*

The use of inverse kinematics to determine the mechanism motion is common in mechanical engineering, particularly in robot research. In a two-joint robotic arm manipulator, given the angles of the joints, the kinematics equations give the location of the tip of the arm. Inverse kinematics refers to the reverse process. Given a desired location for the tip of the robotic arm,

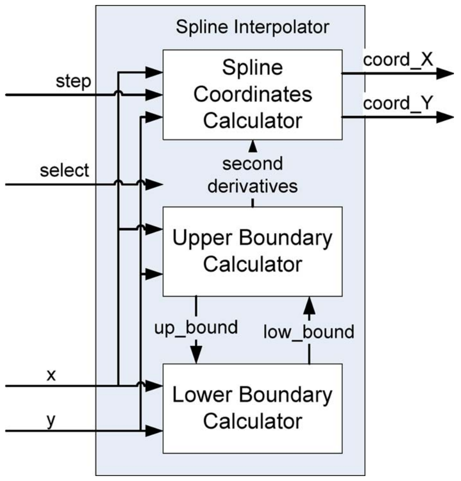


Fig. 5. Spline interpolation block. It consists of a lower boundary calculator, an upper boundary calculator, and a spline coordinate calculator.

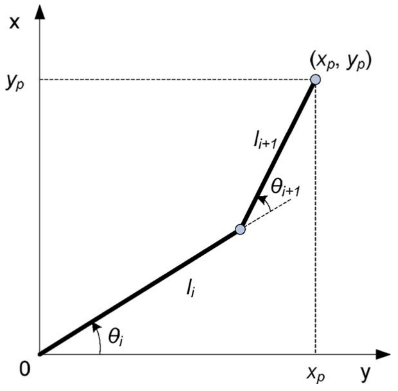


Fig. 6. Two-joint robotic arm manipulator with two angles. It illustrates the problem of the inverse kinematics calculations.

the problem is to find the angles of the joints to locate the tip of the arm at the desired location.

Let the first angle θ_i be the angle between the first arm and the ground. Let the second angle θ_{i+1} be the angle between the second arm and the first arm (Fig. 6). Let the length of the first arm be l_i and that of the second arm be l_{i+1}.

Let us assume that the first joint has limited freedom to rotate and that it can rotate between 0° and 90°. Similarly, let us assume that the second joint has limited freedom to rotate and can rotate between 0° and 180°. (This assumption eliminates the need to handle special cases that will obscure this discussion.) Hence, $0 \le \theta_i \le \pi/2$, and $0 \le \theta_{i+1} \le \pi$.

Now, for every combination of θ_i and θ_{i+1}, the values of the x and y coordinates are deduced using forward kinematics formulas. The following equations show how data are generated for all combinations of the θ_i and θ_{i+1} values. The x and y coordinates are determined as

$$x_p = l_i \cos\theta_i + l_{i+1}\cos(\theta_i + \theta_{i+1}) \tag{22}$$

$$y_p = l_i \sin\theta_i + l_{i+1}\sin(\theta_i + \theta_{i+1}). \tag{23}$$

In contrast, for every combination of x and y coordinates, θ_i and θ_{i+1} values are deduced using inverse kinematics formulas.

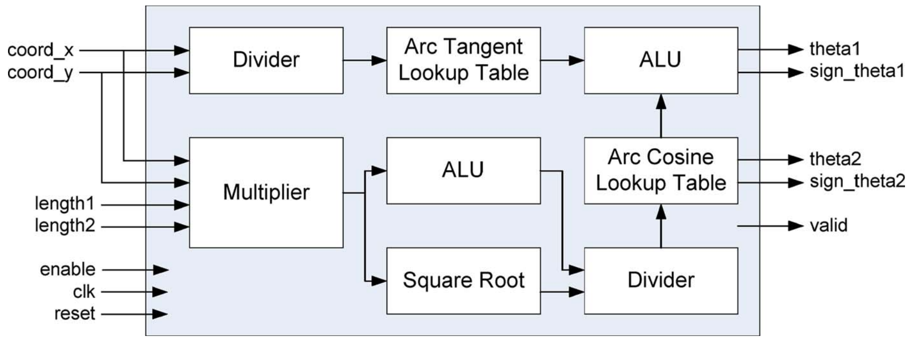


Fig. 7. Inverse kinematics calculation module. It consists of dividers, arithmetic logic units (ALUs), an arc tangent lookup table, an arc cosine lookup table, a square root, and a multiplier.

The following equations show how data are generated for all combinations of the x and y coordinates [23], [24]:

$$\theta_i = \tan^{-1}\left(\frac{y_p}{x_p}\right) - \cos^{-1}\left(\frac{x_p^2+y_p^2+l_i^2-l_{i+1}^2}{2l_i\sqrt{x_p^2+y_p^2}}\right) \tag{24}$$

$$\theta_{i+1} = \cos^{-1}\left(\frac{x_p^2+y_p^2-l_i^2-l_{i+1}^2}{2l_il_{i+1}}\right). \tag{25}$$

B. Inverse Kinematics Calculation Module

The inverse kinematics calculation module in the proposed multiple-axis motion control chip is based on the efficient inverse kinematics technique which is explained in the previous section. In our implementation, the drawing operation requires the cooperation of two links. Therefore, from (24) and (25), consider when $i = 1$, the input parameters of the inverse kinematic module are x_p, y_p, l_1, and l_2. These parameters are used for the computation of the output joint angles θ_1 and θ_2. Fig. 7 shows the architecture of the inverse kinematics calculation module. Its inputs, namely, *coord_x*, *coord_y*, l_1 (*length1*), and l_2 (*length2*), represent the x and y coordinates and the lengths of each link of the robotic arm, respectively. *coord_x* and *coord_y* are the coordinates of each interpolation obtained from the interpolation calculator module. These variables are multiplied by 128 to avoid floating-point calculations with a decimal fraction and to improve the accuracy of the operation. This multiplication by 128 is compensated by seven right shift operations.

After the input values are loaded in the module, the *enable* signal is asserted in order to start calculating the inverse kinematics. Its outputs, namely, θ_1 (*theta1*) and θ_2 (*theta2*), represent the angles of the joints of the robotic arm. θ_1 and θ_2 are multiplied by 20 to avoid floating-point calculations with a decimal fraction and to improve the accuracy of the calculations. This multiplication by 20 is compensated in the pulse integrator module. The angle value is converted to the pulse value depending on the pulse per degree of each axis of the robotic arm. The ranges of each angle are $-\pi/2 \leq \theta_1 \leq \pi/2$ and $0 \leq \theta_2 \leq \pi$. The sign of θ_1 *sign_theta1* is needed, because θ_1 can be either negative or positive. The sign of θ_2 *sign_theta2* is not needed, because the θ_2 can only be positive. The *valid* signal is able to obtain the valid values of the outputs *theta1*, *theta2*, and *sign_theta1* after the completion of the inverse kinematics calculation.

The inverse kinematics technique also includes many of the mathematical functions that are used in the interpolation technique. These include multiplication, division, trigonometric functions, and square root. The multiplier and pipelined divider IP cores, supplied by the Xilinx Company, are used to design the multiplication and division operations, respectively. The square root is designed with the coordinate rotational digital computer IP core supplied by the Xilinx Company. The arc tangent and arc cosine are designed with lookup tables that contain the values of the arc tangent and arc cosine. The value in degrees is selected according to the input value. The variables used in the arc tangent equation are multiplied by 2048 (2^{11}) to avoid floating-point calculations with a decimal fraction and to improve the accuracy of the calculations. The variables used in the arc cosine equation are multiplied by 16 384 (2^{14}) for the same reason.

V. Proposed Multiple-Axis Motion Control Chip

The proposed multiple-axis motion control consists of many modules in addition to those mentioned. These modules include a feedback counter module used to calculate the current motor position, a pulse integrator module used to calculate the integrated pulse during the sampling time, a PID controller module used to control the motor position, a data converter module used to convert the output of the circuit into the driving signals for several types of servo drivers, a clock generator module used to provide several kinds of clock signals, and an external interface module used to communicate with the host computer through the serial cable. Fig. 8 shows the architecture of the proposed multiple-axis motion control chip.

The feedback counter module calculates the current motor position, namely, *count1* and *count2*, from the encoder signals, namely, *encoder1* and *encoder2*, using 32-b up and down counters. Since the encoder signals may contain glitch-type noises, a digital delay based on the two-out-of-three voting technique is included to eliminate noise (Fig. 9). The encoder signal consists of two phase signals, namely, *phase_a* and *phase_b*, for each encoder. The clock of this digital delay filter is dependent on the maximum output frequency of the encoder. The pulse integrator module produces the integrated pulse, *integrated pulse*, during sampling time depending on the commanded

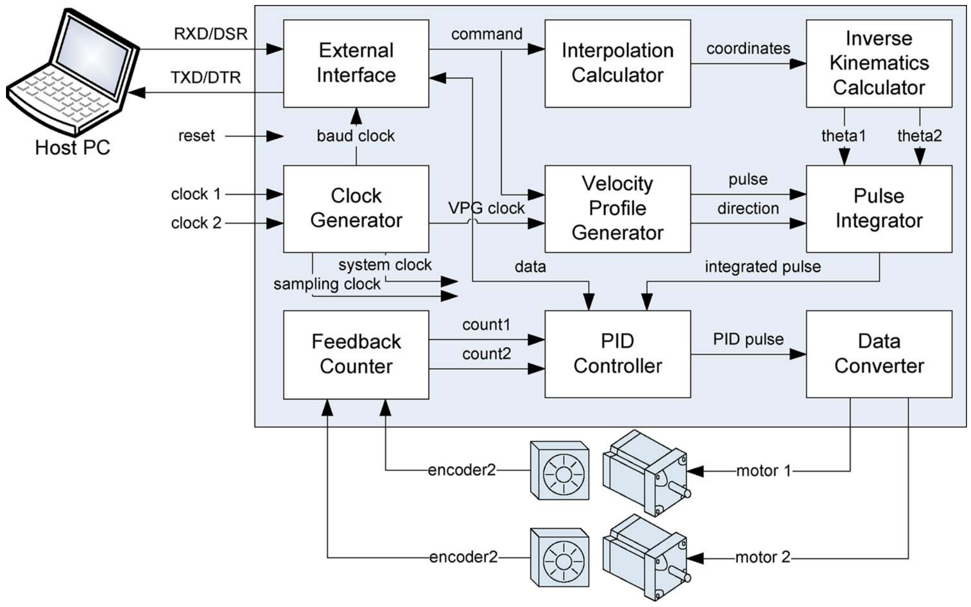


Fig. 8. Proposed multiple-axis motion control chip. It consists of an external interface, an interpolation calculator, an inverse kinematics calculator, a clock generator, a velocity profile generator, a pulse integrator, a feedback counter, a PID controller, and a data converter.

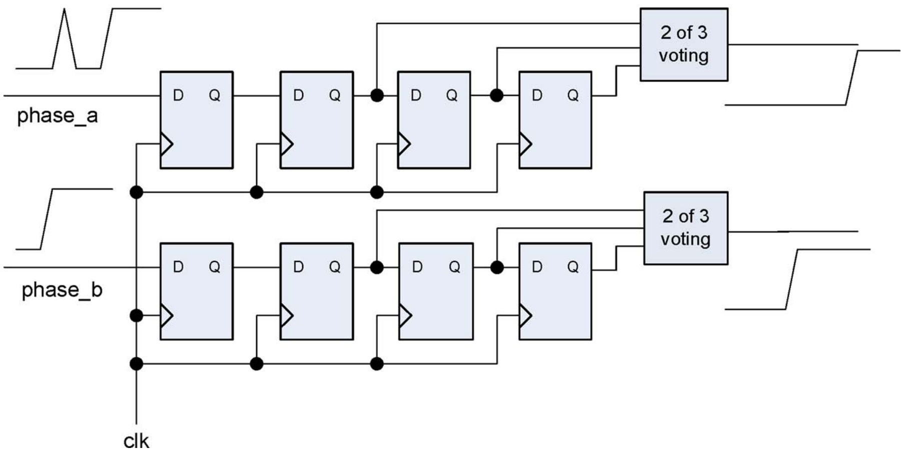


Fig. 9. Digital delay filter. It consists of D flip-flops and two-of-three voting machines.

pulse, *pulse*, and rotation direction, *direction*, from the velocity profile generator and the commanded angle of the first axis θ_1 and angle of the second axis θ_2 of the robotic arm from the inverse kinematics calculator module. The integrated pulse is increased and decreased depending on the counterclockwise and clockwise rotation directions, respectively. The integrated pulse is increased and decreased depending on the angle value that is converted to the pulse value depending on the pulse per degree of each axis of the robotic arm. This module is similar to a DDA. The PID controller module performs the PID control of the motors. It generates a PID control output, *PID pulse*, with current motor position, namely, *count1* and *count2*, obtained from the feedback counter module and the integrated pulse, *integrated pulse*, obtained from the pulse integrator module depending on the PID gain values of the PID control obtained from the host PC through the external interface module. The data converter module converts the PID control output, *PID pulse*, into driving signals, namely, *motor1* and *motor2*, for servo amplifiers of server types. It can produce the pulse and direction for two-channel pulse-based servo amplifiers. It can also produce a 12-b digital value output for digital servo amplifiers and the digital-to-analog converter (DAC) of analog servo amplifiers. The clock generator module provides the sampling clock, *sampling clock*, the system clock, *system clock*, the clock for the velocity profile generator module, *VPG clock*, the clock for the external interface module, *baud clock*, and some timing signals for the proposed multiple-axis motion control chip. The system input clock frequency is 10 MHz for the system clock and the sampling clock and 24.576 MHz for the VPG clock and the external interface clock. The operation clock of this motion control chip can be raised to 36.313 MHz. Since the external interface module contains a serial communication function, the proposed multiple-axis motion control chip can communicate with the host PC through the serial cable. The desired command and the gain values of the PID control are transferred from the host PC to the multiple-axis motion control chip. The pulse command and the motor position are transferred from the multiple-axis motion control chip to the host PC.

TABLE I
SUMMARY OF THE DEVICE UTILIZATION CHARACTERISTICS FOR THE MULTIPLE-AXIS MOTION CONTROL CHIP

Device Utilization Summary				
Logic Utilization				
Number of Slice Flip Flops:	16,031	out of	67,584	23%
Number of 4 input LUTs:	41,490	out of	67,584	61%
Logic Distribution				
Number of occupied Slices:	27,661	out of	33,792	81%
Total Number 4 input LUTs	44,364	out of	67,584	65%
Number of bonded IOBs:	86	out of	824	10%
Number of MULT18X18s:	16	out of	144	11%
Number of GCLKs:	4	out of	16	25%
Total equivalent gate count for design:				546,151

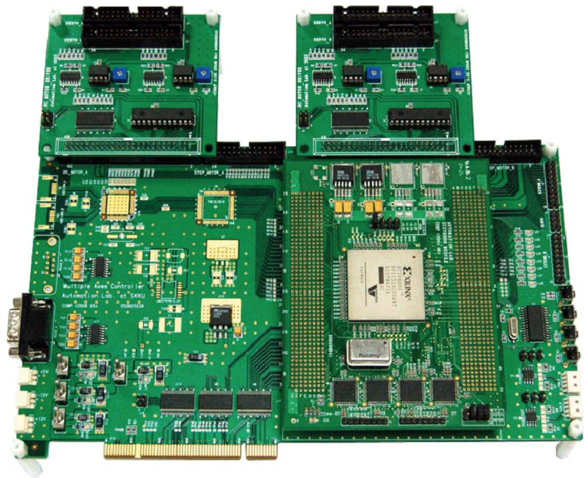


Fig. 10. Proposed multiple-axis motion controller employing a multiple-axis motion control chip. It consists of a motion control board that has an FPGA, a UART, and a level shifter, and a servo driver board that has a DAC, an Op-Amp, and a level shifter.

VI. EXPERIMENT

The proposed multiple-axis motion control chip is designed using VHDL and implemented in a Xilinx FPGA, XC2V6000-FF1152, that has 6M system gates, 76 032 logic cells, and 1104 I/O pins [25]. A summary of the device utilization characteristics for the circuit of the multiple-axis motion control chip is given in Table I. The chip-based multiple-axis motion controller is made with this FPGA-implemented multiple-axis motion control circuit and a few ICs (Fig. 10). This controller consists of a motion control board that has the FPGA, a universal asynchronous receiver/transmitter (UART), a level shifter, and a servo driver board that has a DAC, an operational amplifier (Op-Amp), and a level shifter. This system can control a maximum of four axes simultaneously since one servo driver board (submodule) is used for two servo drivers. The number of chips required for other robots depends on the degrees of freedom (DOF) of the robot and the resources of the implemented FPGA. One DOF normally corresponds to one axis or one motor. In our implementation, the axes are controlled by independent processes running simultaneously. When an additional axis is required, one control component must be added in the VHDL program. If additional control components are added, more FPGA resources are required. Thus, the number of controlled axes is restricted by the FPGA resources. The motion control chip implemented in this paper can control robots with up to four axes. In our implementation of a SCARA robot with four DOFs, only two axes are controlled for testing the proposed design.

The multiple-axis motion control system shown in Fig. 11 consists of a chip-based multiple-axis motion controller, ac servo drivers, CSA12-0xxS1R10R FARA servo, a SCARA robot, SM2 FARA robot, and a host PC [26]. The SCARA robot is widely adopted in industrial environments, and it is also widely used by researchers in order to show the effectiveness of newly devised controllers [1], [5]. Fig. 12 shows the experiment with a SCARA robot in which the multiple axes are driven by analog ac servo drivers. A pen is attached at the end effector of the SCARA robot. This SCARA robot has four DOFs. Four brushless dc (BLDC) servo motors are used for the four-axis manipulator. These motors have an encoder with a resolution of 4096 pulses/revolution. The length of the first link is 350 mm, and the length of the second link is 300 mm. Two reducer gears are mounted on the first and second axes, with a reduction ratio of 1:50 [27]. These mechanical structure parameters are considered when designing this motion control chip.

There are two typical operations in robotic applications; it can be single-axis motion or moving end effector of robot arm following a given trajectory. With single-axis motion, the motor can be controlled by following a given velocity profile, and the input parameters include maximum velocity $V_{\max}$, acceleration time T_{a}, deceleration time T_{d}, the shape of the velocity profile, and the motion distance S. The velocity profile generator block produces the velocity profile, and the PID controller block controls the motor according to the generated velocity profile. When moving the end effector of a robot arm, the required information includes the end point coordinates for linear interpolation, the center point coordinate and radius for circular interpolation, and the coordinates of several trajectory points for spline interpolation. The interpolation block will generate a smooth trajectory based on input information. The robot arm will be controlled by the PID controller block to follow the given trajectory. The host PC sends input commands according to both operations and receives the feedback information through the serial cable.

In the single-axis control experiment, in order to make a motor move the desired distance, the velocity profiles of the symmetric and asymmetric characteristics (Table II) are generated by the proposed multiple-axis motion control chip. Fig. 13 shows the experimental results when these velocity profiles are applied to control one axis in the proposed multiple-axis control system. Fig. 13(a) has a symmetrical smooth velocity profile. Fig. 13(b) has an asymmetrical smooth velocity profile. In the symmetrical smooth velocity profile, the acceleration and deceleration characteristics are the same S-shaped. In the asymmetrical velocity profile, the acceleration characteristic is bell shaped, and the deceleration characteristic is S-shaped. In both cases, the acceleration and deceleration times are 320 ms. The desired distance to move is 100 000 pulses. Both profiles are popular in single-axis applications; the main difference between the two is that the asymmetrical profile can accelerate faster than the symmetrical profile. Fig. 13 contains the velocity commands and its responses of the motor when using the symmetric and asymmetric velocity profiles. In this experiment,

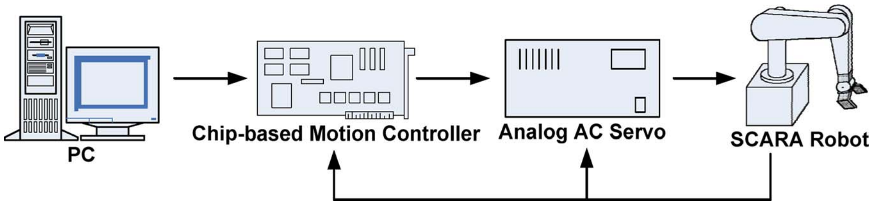


Fig. 11. Proposed multiple-axis motion control system. It consists of a chip-based multiple-axis motion controller, ac servo drivers, CSA12-0xxS1R10R FARA servo, a SCARA robot, SM2 FARA robot, and a host PC.

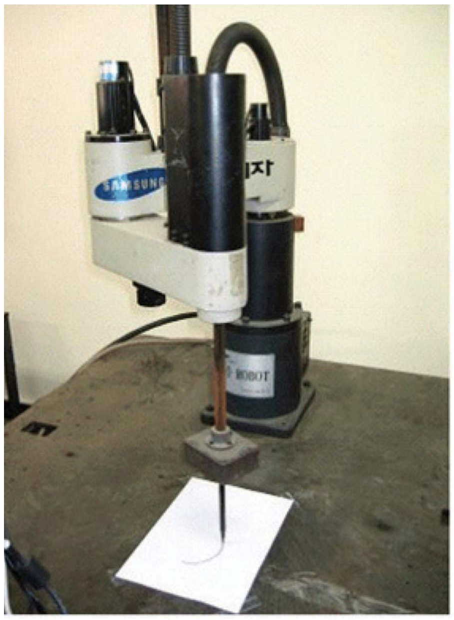


Fig. 12. Experiment with a SCARA robot.

the PID controller uses a relatively small derivative gain and integral gain; thus, the deviation between the command and the response velocity cannot be removed. With the multiple-axis control experiment, in order to make the robotic arm move to the desired position without any sharp jerks, the interpolations of three different characteristics are generated by the proposed multiple-axis motion control chip. In these multiple-axis experiments, the feedback information is collected by the Xilinx ChipScope Pro tool that can check the FPGA on-chip signals online.

The step responses of both axes are shown for verifying the impact of the sampling time on the performance. Fig. 14 shows the step responses of axes 1 and 2 with different proportional gains (Kp) and sampling frequencies. The rise time of each axis will change according to the value of the Kp gain. When a low Kp gain is used, the rise time is longer. On the other hand, the rise time is shorter when high Kp gain is used. With axis 1, when Kp $= 0.125$, the rise time is about 200 ms [Fig. 14(a)], and when Kp $= 0.3$, the rise time is about 50 ms [Fig. 14(b)]. The rise time is shorter by about four times with the latter value. Therefore, high gain can improve the speed of the drawing significantly. However, the high Kp will generate an overshoot of about 10%. This overshoot is undesired in the drawing operation; therefore, the low gain with relatively small overshoot and longer rise time is selected in our implementation. With axis 2 [Fig. 14(c) and (d)], the low gain is used for the same reason. In this experiment, the shortest rise time is about 30 ms [Fig. 14(d)]. This long rise time is dependent on the output power of the ac servo driver and the mass of the robot links. The effect of the sampling frequency is also shown in these step responses. Since the rise time of about 30 ms is substantially longer than 1 ms (corresponding to 1 kHz), the performances are similar when the sampling frequency is 1 or 100 kHz. The responses corresponding to the 1-kHz (dot line) and 100-kHz (dash line) sampling frequencies almost overlap each other. Even if the sampling rate of the proposed design can be 100 kHz, the performance is not different from the performance that is obtained when using a 1-kHz sampling rate because the response of the mechanical system is slow in this experiment. If the proposed motion control chip is applied to a servo driver and a robot that have a faster response, a 100-kHz sampling rate can offer enhanced performance in control velocity and position over a 1-kHz sampling rate.

Fig. 15 shows the result of the experiment in which a line is drawn from one point (300 mm, 300 mm) to another point (400 mm, 400 mm). In this experiment, the drawing is performed in 2 s. Fig. 15(a) shows the line drawn on a piece of paper; the vertical dimension is verified to be equal to 100 mm. Fig. 15(b) and (c) shows the feedback angles of axes 1 and 2, respectively. Fig. 15(d) and (e) shows the angle errors between the reference angle and the feedback angles of axes 1 and 2. The angle error is measured in degrees. The maximum error of both axes is about 0.6°. In this experiment, the errors of both axes are positive, and the feedback angle lags the reference angle during drawing. At the end of the drawing process, the angle error is removed by the integral gain. Axis 2 contains no overshoot, but there is a small overshoot on axis 1. This overshoot affects the quality of the drawing at the end of the drawn line. This experiment shows the results with the 1-kHz (solid line) and the 100-kHz (dash line) sampling frequencies. These two results are so similar that they overlap each other.

Fig. 16 shows the experimental result in which a circle is drawn. This circle has a center at (300 mm, 300 mm) and a radius of 50 mm. This drawing is performed by circular interpolation, and the drawing time is about 3.6 s. Fig. 16(a) shows a circle drawn on paper, and the circle radius of 50 mm can be verified. Fig. 16(b) and (c) shows the feedback angles of axes 1 and 2. Fig. 16(d) and (e) shows the angle errors of axes 1 and 2. The angle error is affected by the rotation direction of the motor, and the feedback angle lags behind the reference angle. When the reference angle is increased, the angle error is positive, and when the reference angle is decreased,

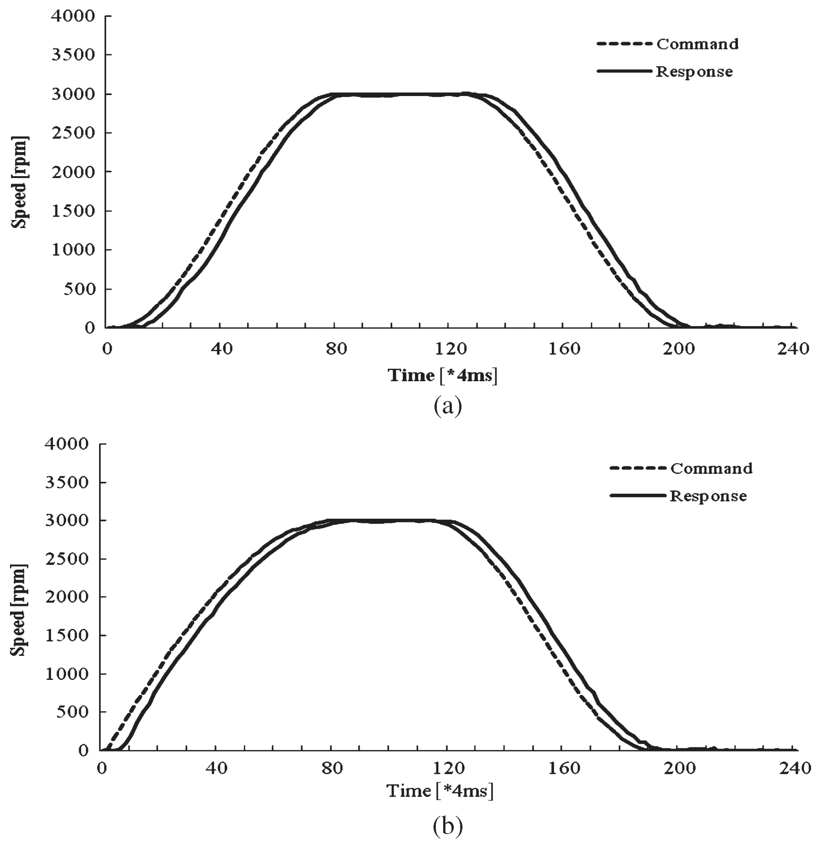


Fig. 13. Velocity profile commands and their responses. It shows the velocity responses of the BLDC servo motor depending on the command of each velocity profile when experiments are performed under proportional control with a gain of 0.375. The acceleration and deceleration times are 320 ms. The desired distance to move is 100 000 pulses. Horizontal unit is 4 ms. (a) Symmetric smooth velocity profile. (b) Asymmetrical smooth velocity profile.

TABLE II
CHARACTERISTICS OF VELOCITY PROFILES

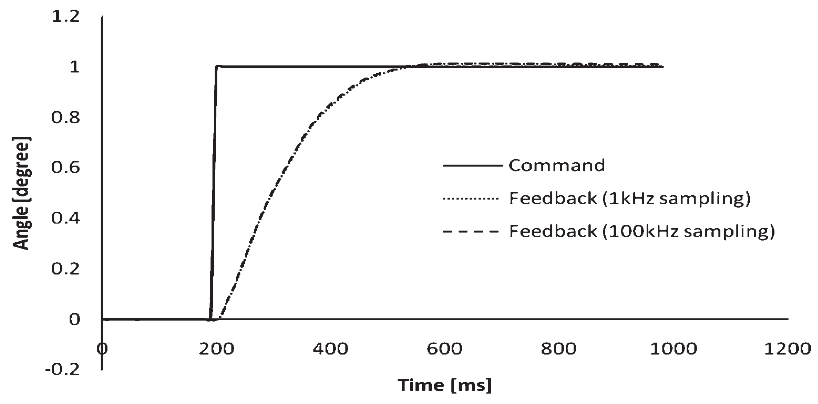
	Velocity profile of symmetric smooth
Acceleration $(1 \le u \le n_a)$	$f_a(u) = \frac{V_m}{2}(\sin(\pi(\frac{u}{n_a} - \frac{1}{2})) + 1)$
Deceleration $(1 \le u \le n_d)$	$f_d(u) = \frac{V_m}{2}(\sin(\pi(\frac{u}{n_d} - \frac{3}{2})) + 1)$
	Velocity profile of asymmetrical smooth
Acceleration $(1 \le u \le n_a)$	$f_a(u) = V_m(\sin(\frac{\pi}{2} \times \frac{u}{n_a}))$
Deceleration $(1 \le u \le n_d)$	$f_d(u) = \frac{V_m}{2}(\sin(\pi(\frac{u}{n_d} - \frac{3}{2})) + 1)$

the angle error is negative. The maximum error of axis 1 is about 0.5°, and that of axis 2 is about 0.3°. This experiment shows the results with both 1- and 100-kHz sampling frequencies. There is a small overshoot with axis 1; this overshoot affects the quality of the drawing at the end of the drawn circle.

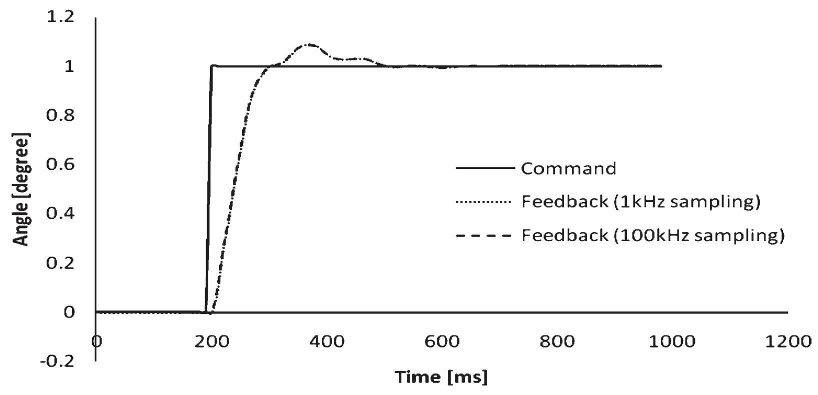
Fig. 17 shows the experimental result drawing a spline curve among five points: (200, 200), (250, 220), (300, 200), (350, 180), and (400, 200); the unit is millimeters. The spline interpolation is used to generate a smooth trajectory based on five point inputs. The drawing time is about 2 s. Fig. 17(a) verifies that the horizontal dimension of the spline curve is 200 mm and that the drawn spline curve is smooth. Fig. 17(b) and (c) shows the feedback angles of axes 1 and 2. Fig. 17(d) and (e) shows the angle errors of axes 1 and 2. In this experiment, the angle error also depends on the direction of the axis. The maximum angle error of axis 1 is about 0.5°, and that of axis 2 is about 0.8°. At the end of the drawing process, the angle error is removed by the integral gain. There is a small overshoot with axis 1; this overshoot affects the quality of the drawing at the end of the drawn shape. This experiment shows the results when the sampling frequency is varied (1 and 100 kHz).

In these experiments, the drawing time is short, and the angle error in the two axes is small. This proves that with parallel processing ability, the proposed design can synchronize multiple axes. The effect of the interpolation calculation with linear, circular, and spline interpolations is verified to be correct. The PID controller block keeps the angle error of each axis low. In addition, the single-axis control ability of the proposed design verifies that it can generate various kinds of velocity profiles efficiently.

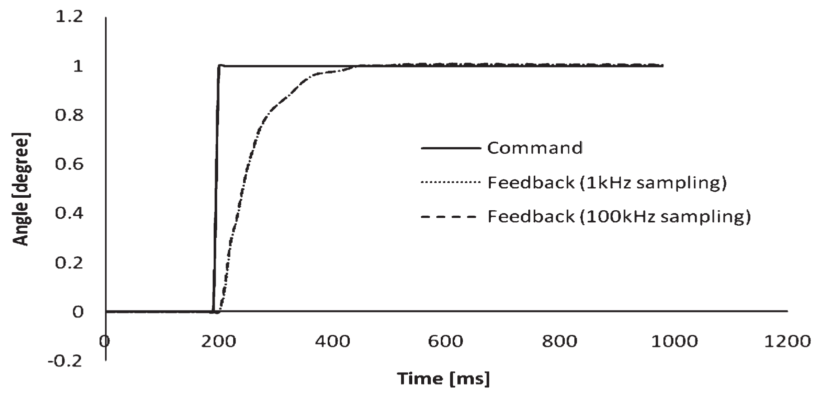
FPGA-based motion controller offers advantages such as high speed, complex functionality, and low power consumption [4]. The power for this FPGA-based multiple-axis motion controller reported by Xilinx Web Power Tools is 0.724 W, where the FPGA has a quiescent power of 0.394 W. The power consumptions of a TI floating-point DSP TMS320C6713 and a TI fixed-point DSP TMS320C6418 are 0.903 and 1.137 W, respectively. The power of the DSP design is estimated using TI power estimation spreadsheet. Table III compares the power



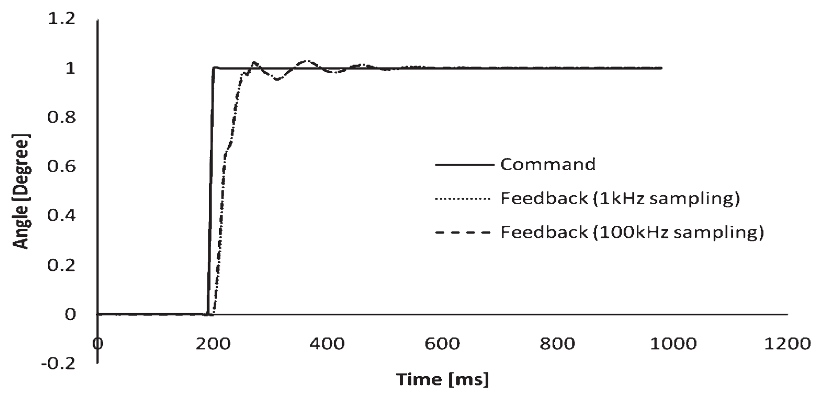
(a)



(b)

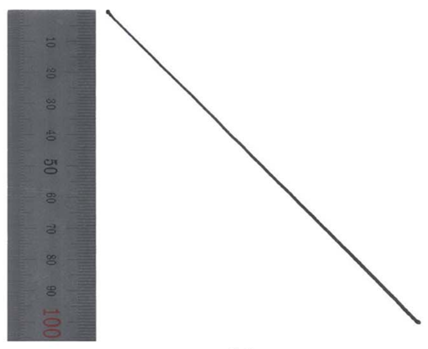


(c)

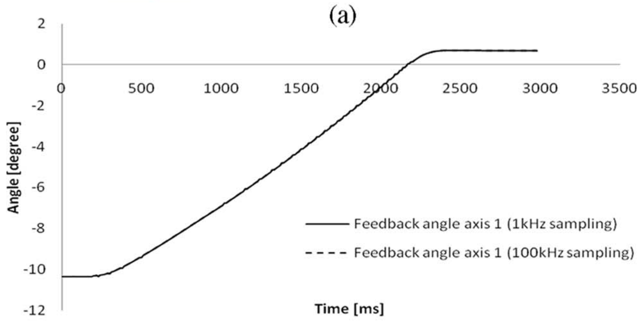


(d)

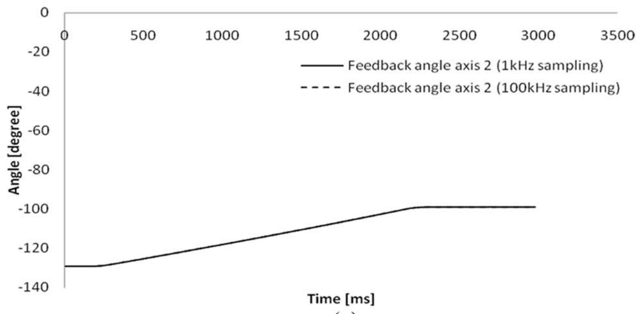
Fig. 14. Step response of both axes with low and high proportional gains. It shows the step response of both axes with different proportional gains and sampling frequencies. The low and high proportional gains of axis 1 are 0.125 and 0.3, respectively. The low and high proportional gains of axis 2 are 0.2 and 0.5, respectively. The 1- and 100-kHz sampling frequencies are applied with these proportional gains. (a) Step response of axis 1 with Kp $= 0.125$. (b) Step response of axis 1 with Kp $= 0.3$. (c) Step response of axis 2 with Kp $= 0.2$. (d) Step response of axis 2 with Kp $= 0.5$.



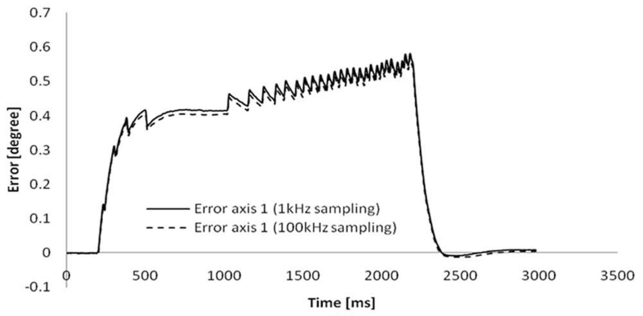
(a)



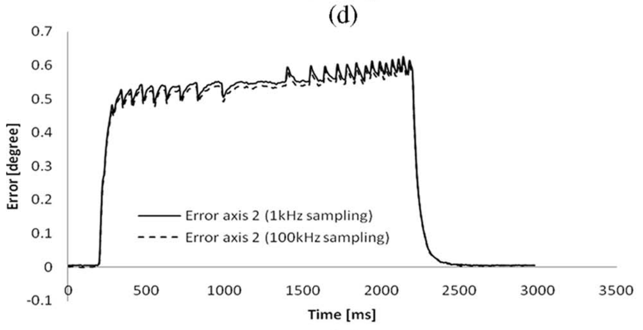
(b)



(c)

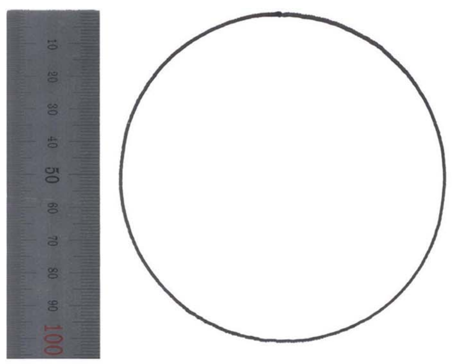


(d)

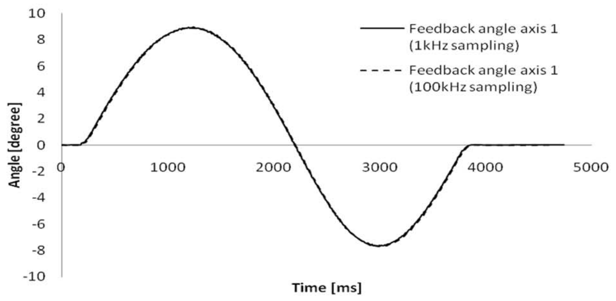


(e)

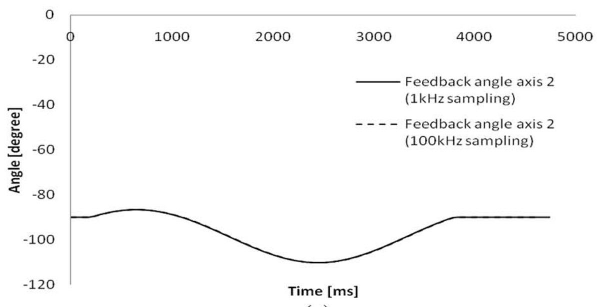
Fig. 15. Experiment with linear interpolation: Drawing a line between two points, namely, (300 mm, 300 mm) and (400 mm, 400 mm). The sampling times are 1 ms and 10 μs. (a) Drawn line. (b) Feedback angle of axis 1. (c) Feedback angle of axis 2. (d) Angle error of axis 1. (e)Angle error of axis 2.



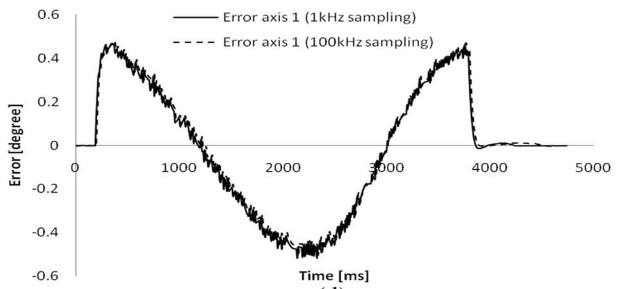
(a)



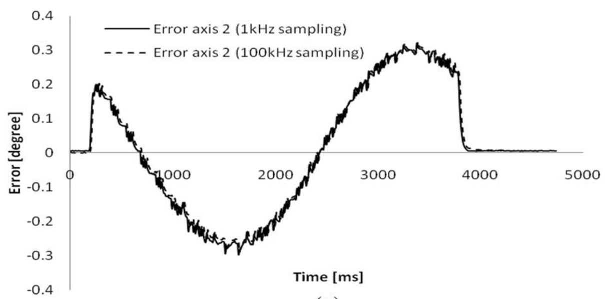
(b)



(c)



(d)



(e)

Fig. 16. Experiment with circular interpolation: Drawing a circle with center at (300 mm, 300 mm) and radius 50 mm. The sampling times are 1 ms and 10 μs. (a) Drawn circle. (b) Feedback angle of axis 1. (c) Feedback angle of axis 2. (d) Angle error of axis 1. (e) Angle error of axis 2.

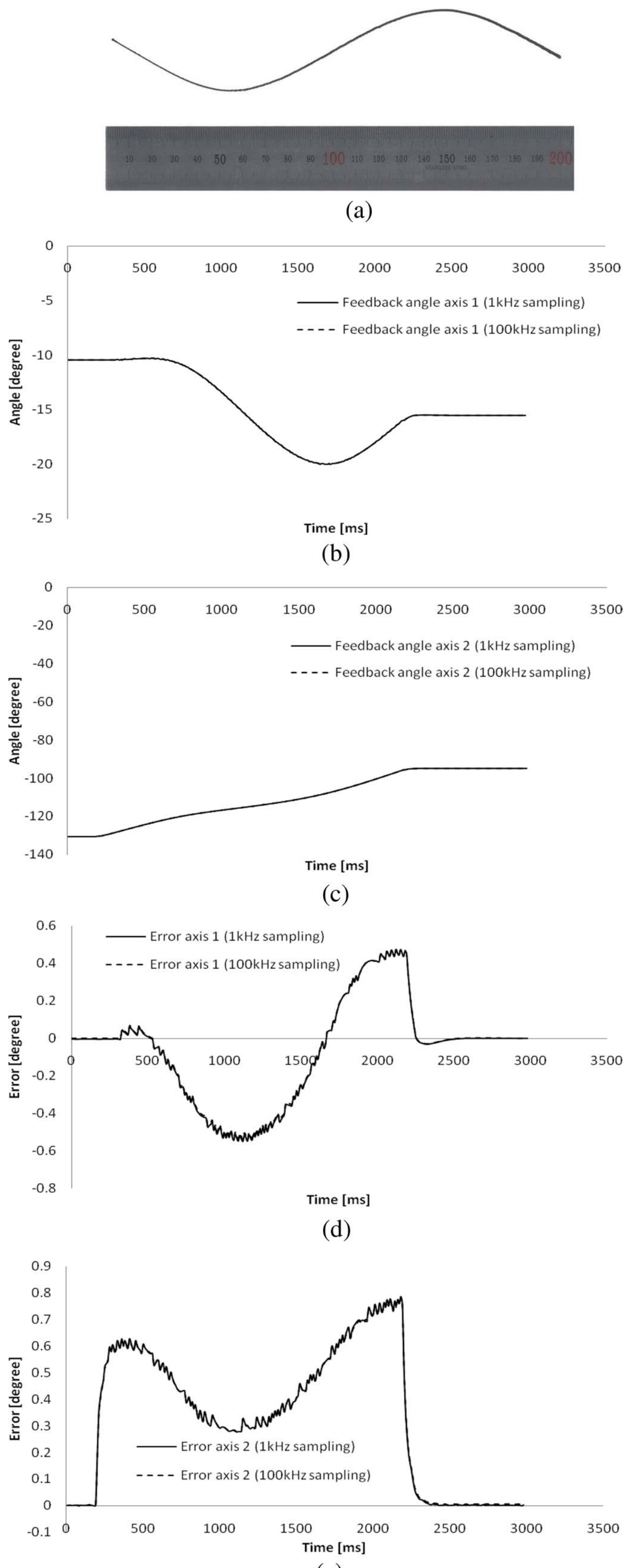


Fig. 17. Experiment with spline interpolation: Drawing a spline among five points: (200, 200), (250, 220), (300, 200), (350, 180), and (400, 200); the unit is millimeters. The sampling times are 1 ms and 10 μs. (a) Drawn spline. (b) Feedback angle of axis 1. (c) Feedback angle of axis 2. (d) Angle error of axis 1. (e) Angle error of axis 2.

TABLE III
COMPARISON OF THE FPGA- AND DSP-BASED MOTION CONTROLLERS

Motion Controller Comparison				
	Sampling Rate (kHz)	Frequency (MHz)	Core Voltage (V)	Power (W)
Virtex-2 FPGA	~100	36	1.5	0.724
TMS320C6713 DSP	~20	300	1.4	1.576
		200	1.2	0.903
TMS320C6418 DSP	~20	600	1.4	1.699
		500	1.2	1.137

consumption of the FPGA- and DSP-based motion controllers. The FPGA-based motion controller delivers a fast sampling rate with lower power consumption.

In a standard DSP-based system, multiple-axis motion controllers can be based on either a single chip or a motion control card. The MC58000 motion control chip series of Performance Motion Devices, Inc., consists of a multiple-axis control IC that supports up to four axes simultaneously, a PID controller, a velocity profile generator, and communication functions. Since this chip does not contain inverse kinematics computations or an interpolation function, it requires an additional device such as a PC for completing the robotic control system. With a Turbo PMAC2-VME motion control card from Data System, Inc., or MMC of Rockwell Samsung Automation, the motion control card supports the control of up to 32 axes simultaneously, and it consists of a velocity profile generator, a PID controller, an inverse kinematics calculator, various interpolation methods, and communication functions. These products support a powerful and complete control system for robotic applications. However, these motion controller cards are system-based controllers where the operation is performed by cooperation of the multiprocessor and external memories.

VII. CONCLUSION

In this paper, a multiple-axis motion control chip was developed to control a multiple-axis motion system such as a robotic arm manipulator or a CNC machine. This chip was designed using VHDL and implemented on an FPGA, thus allowing for a highly sampled, accurate, flexible, compact, low-power, and low-cost motion control system. The proposed multiple-axis motion control chip has many functions. These include velocity profile generation, interpolation calculation, inverse kinematics calculation, PID control, feedback count, pulse integration, data conversion, clock generation, and external interface in order to quickly and accurately perform all of the complex tasks required for industrial robots and automation systems. A multiple-axis motion control system with this chip is implemented on a platform consisting of a chip-based multiple-axis motion controller, analog ac servo drivers, a SCARA robot, and a host PC. The experimental results (such as velocity profiles and interpolations) of this multiple-axis motion control system demonstrate that the proposed multiple-axis motion control chip has the performance that is required to efficiently perform the given task. This paper further demonstrates that the proposed multiple-axis motion control chip can be used for a wide range of applications in industrial automation.

References

[1] X. Shao, D. Sun, and J. K. Mills, "A new motion control hardware architecture with FPGA-based IC design for robotic manipulators," in *Proc. IEEE Int. Conf. Robot. Autom.*, May 2006, pp. 3520–3525.

[2] F.-J. Lin and P.-H. Shen, "Robust fuzzy neural network sliding-mode control for two-axis motion control system," *IEEE Trans. Ind. Electron.*, vol. 53, no. 4, pp. 1209–1225, Aug. 2006.

[3] T. N. Chang, B. Cheng, and P. Sriwilaijaroen, "Motion control firmware for high-speed robotic systems," *IEEE Trans. Ind. Electron.*, vol. 53, no. 5, pp. 1713–1722, Oct. 2006.

[4] Y. F. Chan, M. Moallem, and W. Wang, "Design and implementation of modular FPGA-Based PID controllers," *IEEE Trans. Ind. Electron.*, vol. 54, no. 4, pp. 1898–1906, Aug. 2007.

[5] A. Visioli and G. Legnani, "On the trajectory tracking control of industrial SCARA robot manipulators," *IEEE Trans. Ind. Electron.*, vol. 49, no. 1, pp. 224–232, Feb. 2002.

[6] S.-W. Park and J.-H. Oh, "Hardware realization of inverse kinematics for robot manipulators," *IEEE Trans. Ind. Electron.*, vol. 41, no. 1, pp. 45–50, Feb. 1994.

[7] J. C. G. Pimentel and H. Le-Huy, "A VHDL-based methodology to develop high performance servo drivers," in *Conf. Rec. IEEE IAS Annu. Meeting*, Oct. 2000, vol. 3, pp. 1505–1512.

[8] Y.-S. Kung, K.-H. Tseng, and T.-Y. Tai, "FPGA-based servo control IC for X–Y table," in *Proc. IEEE Int. Conf. Ind. Technol.*, Dec. 2006, pp. 2913–2918.

[9] Y.-Y. Tzou and T.-S. Kuo, "Design and implementation of an FPGA-based motor control IC for permanent magnet AC servo motors," in *Proc. Int. Conf. Ind. Electron., Control, Instrum.*, Nov. 1997, vol. 2, pp. 943–947.

[10] T. Takahashi and J. Goetz, "Implementation of complete AC servo control in a low cost FPGA and subsequent ASSP conversion," in *Proc. IEEE Appl. Power Electron. Conf. Expo.*, Feb. 2004, vol. 1, pp. 565–570.

[11] K. D. Oldknow and I. Yellowley, "FPGA-based servo control and three-dimensional dynamic interpolation," *IEEE/ASME Trans. Mechatronics*, vol. 10, no. 1, pp. 98–110, Feb. 2005.

[12] H.-T. Yau, M.-T. Lin, Y.-T. Chan, and K.-C. Yuan, "Design and implementation of real-time NURBS interpolator using a FPGA-based motion controller," in *Proc. IEEE Int. Conf. Mechatronics*, Jul. 2005, pp. 56–61.

[13] A. M. Lienhardt, G. Gateau, and T. A. Meynard, "Digital sliding-mode observer implementation using FPGA," *IEEE Trans. Ind. Electron.*, vol. 54, no. 4, pp. 1865–1875, Aug. 2007.

[14] X. Lin-Shi, F. Morel, A. M. Llor, B. Allard, and J.-M. Retif, "Implementation of hybrid control for motor drivers," *IEEE Trans. Ind. Electron.*, vol. 54, no. 4, pp. 1946–1952, Aug. 2007.

[15] M.-W. Naouar, E. Monmasson, A. A. Naassani, I. S. Belkhodja, and N. Patin, "FPGA-based current controllers for AC machine drives—A review," *IEEE Trans. Ind. Electron.*, vol. 54, no. 4, pp. 1907–1925, Aug. 2007.

[16] E. Ishii, H. Nishi, and K. Ohnishi, "Improvement of performances in bilateral teleoperation by using FPGA," *IEEE Trans. Ind. Electron.*, vol. 54, no. 4, pp. 1876–1884, Aug. 2007.

[17] J. W. Jeon and Y. Y. Ha, "A generalized approach for the acceleration and deceleration of industrial robots and CNC machine tools," *IEEE Trans. Ind. Electron.*, vol. 47, no. 1, pp. 133–139, Feb. 2000.

[18] J. W. Jeon, "Efficient acceleration and deceleration technique for short distance movement in industrial robots and CNC machine tools," *Electron. Lett.*, vol. 36, no. 8, pp. 766–768, Apr. 2000.

[19] J. W. Jeon and Y. G. Kim, "FPGA based acceleration and deceleration circuit for industrial robots and CNC machine tools," *Mechatronics*, vol. 12, no. 4, pp. 635–642, May 2002.

[20] J. U. Cho and J. W. Jeon, "A motion-control chip to generate velocity profiles of desired characteristics," *ETRI J.*, vol. 27, no. 5, pp. 563–568, Oct. 2005.

[21] Y. Koren, *Computer Control of Manufacturing Systems*. New York: McGraw-Hill, 1984.

[22] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge, U.K.: Cambridge Univ. Press, 1992.

[23] K. S. Fu, R. C. Gonzalez, and C. S. G. Lee, *Robotics: Control, Sensing, Vision, and Intelligence*. New York: McGraw-Hill, 1987.

[24] R. Schilling, *Fundamentals of Robotics: Analysis and Control*. Englewood Cliffs, NJ: Prentice-Hall, 1998.

[25] Xilinx, Inc., *Virtex-II Platform FPGAs: Complete Data Sheet*, Mar. 2005. [Online]. Available: www.xilinx.com

[26] *Servo Driver and Power Controller User's Manual*, Samsung Electron. Co., Ltd., Suwon, Korea, 1993.

[27] *FARA SCARA Robot SM2/SS2 Operator Guide*, Samsung Electron. Co., Ltd., Suwon, Korea, 1993.

Jung Uk Cho received the B.S., M.S., and Ph.D. degrees from the School of Information and Communication Engineering, Sungkyunkwan University, Suwon, Korea, in 2001, 2003, and 2006, respectively.

From 2006 to 2007, he was a Research Instructor with the School of Information and Communication Engineering, Sungkyunkwan University. Since 2008, he has been with the Department of Computer Science and Engineering, University of California at San Diego, La Jolla, as a Postdoctoral Scholar. His research interests include motion control, image processing, and embedded systems.

Quy Ngoc Le was born in Hanoi, Vietnam. He received the B.S. degree from the Faculty of Electronics and Telecommunications, Hanoi University of Technology, Hanoi, in 2003, and the M.S. degree from the School of Information and Communication Engineering, Sungkyunkwan University, Suwon, Korea, in 2006, where he is currently working toward the Ph.D. degree.

His research interests include robust control, power electronics, and motor drives.

Jae Wook Jeon (S'82–M'84) received the B.S. and M.S. degrees in electronics engineering from Seoul National University, Seoul, Korea, in 1984 and 1986, respectively, and the Ph.D. degree in electrical engineering from Purdue University, West Lafayette, IN, in 1990.

From 1990 to 1994, he was a Senior Researcher with Samsung Electronics, Suwon, Korea. Since 1994, he has been with Sungkyunkwan University, Suwon, where he was first an Assistant Professor with the School of Electrical and Computer Engineering and is currently a Professor with the School of Information and Communication Engineering. His research interests include robotics, embedded systems, and factory automation.