Hindawi Publishing Corporation
International Journal of Reconfigurable Computing
Volume 2010, Article ID 727909, 10 pages
doi:10.1155/2010/727909

Research Article

A Reconfigurable System Approach to the Direct Kinematics of a 5 *D.o.f* Robotic Manipulator

Diego F. Sánchez, Daniel M. Muñoz, Carlos H. Llanos, and José M. Motta

Departamento de Engenharia Mecânica, Universidade de Brasília, 70910-900 Brasilia, DF, Brazil

Correspondence should be addressed to Carlos H. Llanos, llanos@unb.br

Received 10 March 2010; Revised 19 October 2010; Accepted 17 December 2010

Academic Editor: Lionel Torres

Hardware acceleration in high performance computer systems has a particular interest for many engineering and scientific applications in which a large number of arithmetic operations and transcendental functions must be computed. In this paper a hardware architecture for computing direct kinematics of robot manipulators with 5 degrees of freedom (5 *D.o.f*) using floating-point arithmetic is presented for 32, 43, and 64 bit-width representations and it is implemented in Field Programmable Gate Arrays (FPGAs). The proposed architecture has been developed using several floating-point libraries for arithmetic and transcendental functions operators, allowing the designer to select (pre-synthesis) a suitable bit-width representation according to the accuracy and dynamic range, as well as the area, elapsed time and power consumption requirements of the application. Synthesis results demonstrate the effectiveness and high performance of the implemented cores on commercial FPGAs. Simulation results have been addressed in order to compute the Mean Square Error (MSE), using the Matlab as statistical estimator, validating the correct behavior of the implemented cores. Additionally, the processing time of the hardware architecture was compared with the same formulation implemented in software, using the PowerPC (FPGA embedded processor), demonstrating that the hardware architecture speeds-up by factor of 1298 the software implementation.

1. Introduction

Currently, there exists a real demand in many robotics applications for higher operational speeds, and the solutions improving performance would have clear benefits in terms of manufacturing efficiency, precision, and processing time. In general, a fully operational robotic system running in real-time requires the repeated execution of a variety of complex algorithms, which involve the use of several transcendental functions and arithmetic operations. Most of these algorithms, if not all, need to be computed within milliseconds (ms) or microseconds (μs) in order to keep several real-time constraints. Such algorithms require massive computing power that surpasses the capabilities of many sequential computers [1].

A classical but important problem in the robotic manipulator is the direct kinematics. The direct kinematics allows the Cartesian location of the end effector to be calculated from measured values of the joint angles [2]. This mathematical formulation frequently needs a large number of arithmetic and trigonometric computations that must ideally be performed in a floating point, because of precision requirement [2]. Design and implementation of floating-point arithmetic operations in FPGAs has a relevant importance in a variety of scientific applications (such as robotics) due to the large dynamic range for representing real numbers, which permits suitable representation of both very small and large numbers in a fixed bit-width with proper precision. Additionally, the software implementations of direct kinematics computations are very expensive in processing time due to the sequential behavior of general purpose processors (GPPs).

Floating-point-based algorithms for arithmetic operators are commonly implemented on software and executed in microprocessors. Typically this solution requires to pay a performance penalty given that the conventional approaches require to perform the data transfer between the ALU (Arithmetic Logic Unit) and the program and

the instruction memories. This problem, well known as von Neumann bottleneck, has been partially overcome by using multicores microprocessors reducing the execution time. Recently, Graphic Processor Units (GPUs) have been used for implementing complex algorithms taking advantage of the parallel floating-point units and increasing in this way their throughput in execution time. Although the GPU-based implementations achieves a noticeable speed-up, it is important to point out the following aspects: (a) the GPU-based solution presents bandwidth bottlenecks when all the source data are accessed from global memory or when simultaneous accesses from different threads to memory have to be addressed, (b) GPUs are not tailored for specific applications and commonly are difficult to fine-tune for executing only the operations required by an algorithm, and (c) these integrated circuits operate at high frequencies leading to large power consumption. This is a drawback for embedded system applications.

Due to the high capacity of parallel processing, FPGAs are now being used to accelerate processes such as digital signal processing, image processing, robotics, encryption and decryption, and communication protocol decoding [3, 4]. Robot tasks with higher operational speeds are one of the applications that require high processing capabilities. Therefore, the FPGA implementation of direct kinematics can be an important solution in order to achieve several real-time constraints.

There are two important aspects that must be considered when parallel processing computations using floating-point operators are implemented in FPGA: (a) the tradeoff between the need of reasonable accuracy and the cost of logic area and (b) the choice of a suitable format such that dynamic range is large enough to guarantee that saturation will not occur for a general-purpose application.

Current advances in VLSI technology raised the density integration fast enough for allowing the designers to develop directly in hardware several algorithms commonly implemented on software, thus, obtaining an expressive processing speed-up [5]. Moreover, computation of direct kinematics has high parallelization capabilities, and then, the performance can be improved by implementing it on FPGAs. In this way, a hardware architecture of direct kinematics taking advantage of these features could be useful in robotics applications that require high-speed movements.

In this paper, an FPGA implementation for computing the direct kinematics of a spheric robot with five degrees of freedom (5 *D.o.f*) is described. The hardware architecture considers a floating-point arithmetic, parameterizable by bit-width, allowing the designer to choose a suitable format according to the available hardware resources, accuracy and dynamic range requirements. The main contributions of this work can be summarized as follows: (a) the proposed architecture makes use of several floating-point arithmetic and trigonometric libraries, allowing the performance to be improved in comparison with previous works implementations, in which the floating-point operations are executed in software using DSPs, GPUs, or CPUs (see Section 2), (b) this work presents an error analysis for different bit-width representations (32, 43, and 64 bits), allowing the designer to analyze the tradeoff between the bit-width representation, which directly affects the cost in logic area and the accuracy requirements. Comparison of the processing time between the hardware architecture and a software implementation, using a PowerPC embedded microprocessor, is also presented, (c) the proposed hardware architecture has been developed taking into account a resource constrained methodology, allowing the arithmetic and trigonometric units to be scheduled between different states in order to achieve the lowest execution time according to the available hardware resources.

Section 2 presents the related works covering hardware implementations of direct and inverse kinematics. Section 3 describes the direct kinematics mathematical formulation. Section 4 describes the IEEE-754 standard for the floating-point number representation and a tradeoff analysis of the FPGA implementation of the floating-point operators required by the direct kinematics. Section 5 presents the FPGA implementations and, before concluding, Section 6 presents the synthesis and simulation results.

2. Related Works

Several previous works have presented hardware architectures for implementing only the servo control loop of robotic manipulators, being in these cases the direct and inverse kinematics computed on software [6–9]. In [6, 7] a hardware structure for controlling a SCARA robotic manipulator is developed, relieving the computational cost of general purpose microprocessor by implementing on FPGAs the servo control loop of the robot manipulator.

In [8, 9] a hardware-software codesign for controlling an articulated robot arm is presented, using a NIOS processor for implementing the inverse kinematics. In the same context, in [10] the case study is a neural controller for 3 *D.o.f* parallel robot for milling. This controller is based on neural model of the inverse dynamics of the manipulator, trained on data collected with the use of a computed torque stabilizing controller, and the authors propose that good candidates for hardware implementation are those fragments of an algorithm that can be calculated using only fixed-point operations, due to the fact that they require less FPGA resources and, therefore, are faster whenever are compared with floating-point modules. The other parts of the algorithm (including those working with floating-point) are good candidates to be implemented in an embedded processor. Additionally, [11] proposes hardware solutions based on an ARM processor and fixed-point FPGA modules for computing the trigonometric and square root functions of inverse kinematics. They are based on existing pipeline arithmetic circuits, which employ the CORDIC (Coordinate Rotation Digital Computer) algorithm.

Taking into account previous works, the hardware/software codesign is a very applied approach. In this direction, [12] implements the direct and inverse kinematics of a multifinger hand system, computing the floating-point arithmetic and trigonometric operations (for instance inverse kinematics) using a high-speed DSP processor and several

hardware peripherals (PWM controllers and interfaces for data communication), connected among them throughout a NIOS embedded processor. On the other hand, [13] shows the implementation of the direct kinematic and force feedback algorithm using both CORDIC (for fixed-point) and Xilinx arithmetic library. This work has measured the error of the direct kinematics with respect to a PC implementation, in which numerical analysis shows that Cartesian position error is always below 0.1 mm and the error in orientation is less than 0.001 deg. Although a 32 bit fixed-point data representation is used, details about the same (for example, bit number for fractional part) are not described.

In [14] a hardware implementation of inverse kinematics and a servo controller for a robot manipulator are proposed, allowing the FPGA to compute the high complex computations, such as the transcendental functions, as well as exploring the parallel capabilities of the inverse kinematics. As previous approaches, all hardware embedded operations are performed in fixed-point.

In [15] a generic controller for a multiple-axis motion system (that can be applied to a manipulator) is implemented, which includes several modules such as velocity profile generator, interpolation calculator, inverse kinematics calculator, PID controller, among others. In this case, the trigonometric operations (such as sin, cos, arc tang and arc cosine) have been also implemented by using lookup tables and a fixed-point arithmetic representation.

Additionally, [16] shows the direct and inverse kinematics computation of a 6 *D.o.f* space manipulator using an ARM processor and FPGA coprocessor. Additionally, it considers the hardware implementation of a pipelined CORDIC library in an FPGA device (using fixed-point representation). The experiment shows that the absolute accuracy of the end-effector is less than 3 mm error. In [17] an optimized inverse kinematics approach is proposed for controlling an arm of a virtual human, in which an FPGA device is used for accelerating the computations. In order to improve the performance, a floating to fixed point conversion is performed; however, it imposes a limitation on the dynamic range of the operations.

In summary, almost all previous works describe a hardware-software codesign for implementing the direct and inverse kinematics, in which critical parts are developed in hardware accelerators, developed in FPGAs. On the other hand, these previous works do not consider explicitly floating-point arithmetic for performing the computations using appropriated arithmetic libraries for FPGA, and this can become very important as required by several engineering applications such as robotic manipulators, in which high accuracy and a large dynamic range are important requirements, apart from attending both good performance and low cost in logic area. In this context, taking the importance and dramatical growth of embedded application for automation, control, and robotics areas [18], the full hardware implementation of a kinematics is very important to be researched, in terms of comparing the cost, performance and precision with respective software implementations, specially developed over FPGA embedded processors. Additionally, most of the previous works do not show the error analysis comparing both hardware and software results of the same kinematics, and only few ones measure the error of the direct/inverse kinematics (for instance, [13, 16]) using only fixed-point representation.

3. Background

3.1. Direct Kinematics. A robot manipulator can be described as a series of links, which connect the end effector to the base, with each link connected to the next by an actuated joint. Attaching a coordinate frame to each link, the relationship between two links can be described with a homogeneous transformation matrix—an A matrix. Therefore, a sequence of these A matrices relates the based to the hand of the manipulator (see (1)) [2].

$$^{R}T_{H} = A_1 A_2 \cdots A_{n-1} A_n. \quad (1)$$

Thus, the direct kinematics problem is summarized by finding a homogeneous matrix transformation T that relates the end effector pose (position and orientation) to the based coordinate frame, knowing the angles and displacement between the links and the geometric parameters of the manipulator [2].

The joint coordinate frames have to be assigned by using a rational convention, associated to a zero position (where all joint variables are set to zero). So, it is assumed that for the assignment of coordinate frames to each link the manipulator has to be moved to its zero position. The zero position of the manipulator is the position where all joint variables are zero. This procedure may be useful to check if the zero positions of the model constructed are the same as those used by the controller, avoiding the need of introducing constant deviations to the joint variables (joint positions). Subsequently the z-axis of each joint should be made coincident with the joint axis. This convention is used by many authors and in many robot controllers [2, 19]. For a prismatic joint, the direction of the z-axis is in the direction of motion, and its sense is away from the joint. For a revolute joint, the sense of the z-axis is towards the positive direction of rotation around the z-axis. The positive direction of rotation of each joint can be easily found by moving the robot and reading the joint positions on the robot controller display. According to the [2, 19], the base coordinate frame (robot reference) may be assigned with axes parallel to the world coordinate frame. The origin of the base frame is coincident with the origin of joint 1 (first joint). This assumes that the axis of the first joint is normal to the $x - y$ plane. This location for the base frame coincides with many manufacturers' defined base frame. Afterwards coordinate frames are attached to the link at its distal joint (joint farthest from the base). A frame is internal to the link it is attached to (there is no movements relative to it), and the succeeding link moves relative to it. Thus, coordinate frame i is at joint $i + 1$, that is, the joint that connects link i to link $i + 1$. The origin of the frame is placed as following: if the joint axes of a link intersect, then the origin of the frame attached to the link is placed at the joint axes intersection; if the joint axes are parallel or do

not intersect, then the frame origin is placed at the distal joint; subsequently, if a frame origin is described relative to another coordinate frame by using more than one direction, then it must be moved to make use of only one direction if possible. Thus, the frame origins will be described using the minimum number of link parameters. The x-axis or the y-axis have their direction according to the convention used to parameterize the transformations between links. At this point the homogeneous transformations between joints must have already been determined. The other axis (x or y) can be determined using the right-hand rule. A coordinate frame can be attached to the end of the final link, within the end-effector or tool, or it may be necessary to locate this coordinate frame at the tool plate and have a separate hand transformation. The z-axis of the frame is in the same direction as the z-axis of the frame assigned to the last joint $(n-1)$. The end-effector or tool frame location and orientation is defined according to the controller conventions. The homogeneous transformations between joints follow the Denavit-Hartemberg convention (D-H), making two consecutive links related through 4 basic transformations that only depend on the link geometry [2]. The transformations are:

(1) a rotation about the z_{n-1} axis by the angle between links (θ_n),

(2) a translation along the z_{n-1} axis of the distance between the links (d_n),

(3) a translation along the x_n axis (rotated x_{n-1} axis) of the length of the link (l_n),

(4) a rotation about the x_n axis of the twist angle (α_n).

$$A_n = R(z,\theta_n)T(0,0,d_n)T(l_n,0,0)R(x,\alpha_n). \quad (2)$$

θ_n, d_n, l_n, and α_n represent the D-H parameters of the link n. Identifying these parameters, the A matrix that relates two consecutive links can be obtained. Multiplying the sequence of the A matrices (see (1)) a homogeneous matrix transformation T is obtained, solving the direct kinematics problem.

3.2. Direct kinematics for a Spherical Robot Manipulator. The robot described in this paper has a spherical topology with 5 degrees of freedom, with two rotational joints in the base followed by a prismatic joint and two rotational joints located in manipulator hand. The first three joints are responsible for the wrist position and the two last ones are responsible for the hand or tool orientation. Figure 1 shows the proposed robot topology.

Figure 2 shows the assigned coordinate frame to each link following the convention D-H for the manipulator depicted in Figure 1. Table 1 presents the D-H parameters.

Table 1 can be used for computing the matrix that relates two consecutive axis frames of the robot, and finally, by multiplying the sequence of the A matrices, a homogeneous matrix transformation T is obtained. The matrix T can be formulated as shown in Equation (3), where the entries

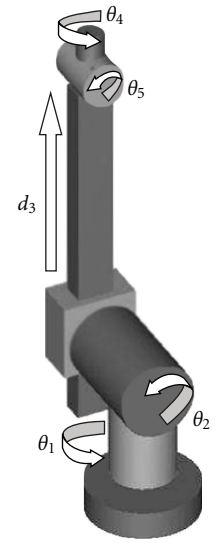

Figure 1: Five degrees of freedom Robot Manipulator.

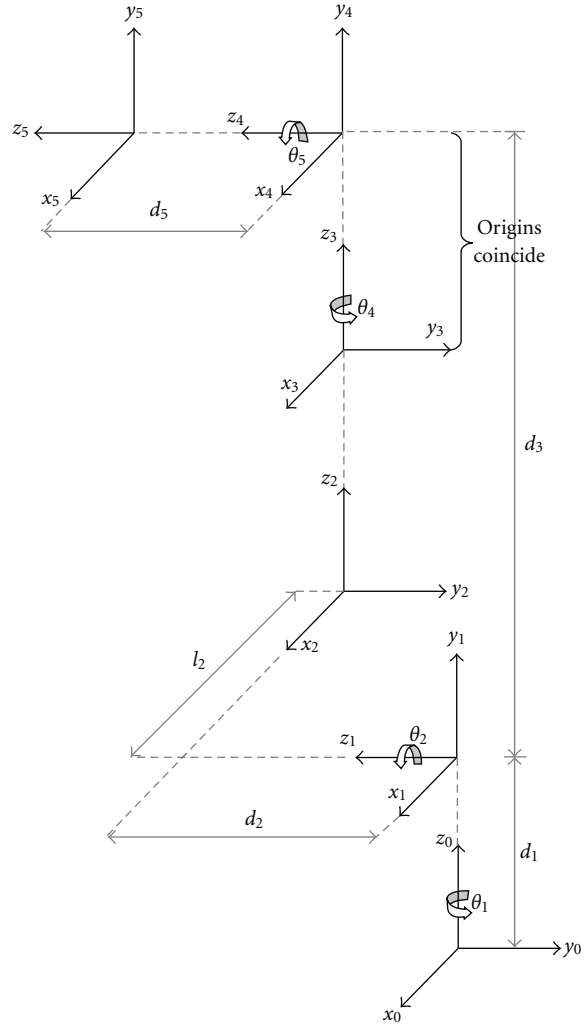

Figure 2: Assignment of coordinate frames.

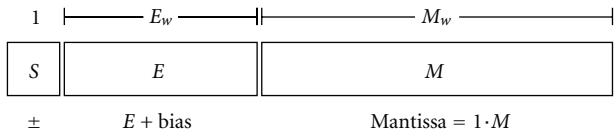

Figure 3: IEEE-754 format.

Table 1: Denavit-Hartenberg parameter of robot manipulator.

Joint variable	Angle θ_n	Displacement d_n	Length l_n	Twist α_n
θ_1	θ_1	d_1 (106 mm)	0	90
θ_2	$\theta_1 + 90$	d_2 (130 mm)	l_2 (0 mm)	−90
d_3	0	d_3	0	0
θ_4	θ_4	0	0	90
θ_5	θ_5	d_5 (0 mm)	0	0

of T follow a notation that represents, for example, y_x as the projection of the y axis of the last coordinate frame in the x axis of the base frame, and p_x is the x component of the origin coordinates of the last frame represented in the base frame.

$$T = \begin{bmatrix} x_x & y_x & z_x & p_x \\ x_y & y_y & z_y & p_y \\ x_z & y_z & z_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{3}$$

By using the D-H notation, the matrix T is summarized by the (4) to (15). In these equations the notation $C_i = \cos(\theta_i)$ and $S_i = \sin(\theta_i)$ was used.

$$x_x = (-C_1S_2C_4 - S_1S_4)C_5 - C_1C_2S_5, \tag{4}$$

$$x_y = (-S_1S_2C_4 + C_1S_4)C_5 - S_1S_2S_5, \tag{5}$$

$$x_z = C_2C_4C_5 - S_2S_5, \tag{6}$$

$$y_x = -(-C_1S_2C_4 - S_1S_4)S_5 - C_1C_2C_5, \tag{7}$$

$$y_y = -(-S_1S_2C_4 + C_1S_4)S_5 - S_1C_2C_5, \tag{8}$$

$$y_z = -C_2C_4S_5 - S_2C_5, \tag{9}$$

$$z_x = -C_1S_2S_4 + S_1C_4, \tag{10}$$

$$z_y = -S_1S_2S_4 - C_1C_4, \tag{11}$$

$$z_z = C_2S_4, \tag{12}$$

$$p_x = -C_1C_2d_3 - C_1S_2l_2 + S_1d_2, \tag{13}$$

$$p_y = -S_1C_2d_3 - S_1S_2l_2 - C_1d_2, \tag{14}$$

$$p_z = -S_2d_3 + l_2C_2 + d_1. \tag{15}$$

It can be observed in the (4) to (15) that some intermediate computations are found in two equations. For example, the intermediate computation $(C_1C_2C_4 - C_1S_2S_4)$ is presented in the (4) and (7). In this way, the intermediate computation should be computed only once, saving processing time.

Although parameter l_2 is equal to zero, it will be taken into account for future calibration purposes.

4. Description and Analysis of the Floating-Point Operators

4.1. IEEE-754 Format. The IEEE-754 format [20] is a floating-point number representation characterized by three components: a sign S, a biased exponent E with E_w bit-width and a mantissa M with M_w bit-width as shown in Figure 3. A zero sign bit denotes a positive number and a one sign bit denotes a negative number. A constant (bias) is added to the exponent in order to make the exponent's range nonnegative and the mantissa represents the magnitude of the number. The standard also includes extensive recommendations for advanced exception handling, additional operations (such as trigonometric functions), and expression evaluation for achieving reproducible results.

This standard allows the user to work not only with the 32-bit single precision and 64-bit double precision, but also with a suitable precision according to the application. This is suitable for supporting variable precision floating-point operations [21].

4.2. An Architectural Approach for Floating-Point Operators. As stated in (4) to (15), the direct kinematics computation of the spherical robot described above requires the computation of add/sub and multiplication arithmetic operators, as well as the sine and cosine transcendental functions. Previous works covering hardware implementations of floating-point transcendental functions on FPGAs are based on CORDIC algorithms using simple and double precision formats [22–24]. In [25] a HOTBM method for computing trigonometric functions in FPGAs is presented, achieving a reduced area and high performance without sacrificing accuracy. However, these works have not received enough attention on the cost in area associated with the precision level, as well as, their respective error analysis.

In this work, the Taylor series expansion has been implemented on FPGAs for computing, in an integrated approach, the sine, cosine, and arctangent functions, using a floating-point arithmetic based on the IEEE-754 standard. Our previous results point out that the Taylor expansion approach has a lower execution time and a lower cost in logic area than the CORDIC-based solution. However, the CORDIC algorithm presents a better performance in terms of precision [26–28]. As will be explained below, this work focuses on solving the resource and timing constrains

(see Section 4); therefore, the choice of using a Taylor expansion approach for computing the floating-point trigonometric operators can be justified. This work considers accuracy as a design criterion and provides an error analysis associated to the bit-width representation and the area cost, as well as, an error analysis associated to the number of iterations of the Taylor series. These results are useful for choosing a suitable format for representing real numbers according to general-purpose applications.

The Taylor series of sine, cosine and arctangent functions, around 0 are given, respectively, as

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots, \tag{16}$$

$$\cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \cdots, \tag{17}$$

$$\arctan(x) = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \cdots. \tag{18}$$

The factors $1/n!$ and $1/n$ are precomputed and stored in a ROM avoiding additional operations. Before the first iteration, the x^2 term is computed and stored. In each iteration one floating-point addition of the accumulated approximation with the ith term is necessary. Additionally, for computing the ith term two floating-point multiplications are needed: (1) for computing either x^2x^{2i-1} or $x^2x^{2(i-1)}$ for sine/arctangent or cosine, respectively, and (2) for computing either $1/(2i+1)!x^{2i+1}$ for sine, $1/(2i+1)x^{2i+1}$ for arctangent and $1/(2i)!x^{2i}$ for cosine.

Figure 4 shows the proposed architecture for the Taylor series expansion. This approach was achieved using only one $\text{FP}_{\text{multiplier}}$, an $\text{FP}_{\text{add/sub}}$, a Finite State Machine (FSM), and three ROMs for storing the precomputed $1/n!$ and $1/n$ factors. The op signal chooses between a sin, cos or atan functions. The FSM synchronizes the $\text{FP}_{\text{multiplier}}$, $\text{FP}_{\text{add/sub}}$ units and alternates the $\text{op}^{+/-}$ signal. At the first iteration, the factor x^2 is computed and stored in a register and fed back to the FSM. Afterward, the precomputed coefficients are selected in order to compute the current term either $x^{2n-1}/(2n-1)!$ for sine, $x^{2n-1}/(2n-1)$ for arctangent or $x^{2n}/2n!$ for cosine. The add/sub unit computes and accumulates both the addition or subtraction operations. After N iterations, the ready signal indicates a valid output.

Table 2 shows the Mean Square Error (MSE) results of the arithmetic and transcendental functions architectures for different bit-width representations (24, 32, 43, and 64 bits), using the Matlab results as statistical estimators. The Taylor architecture uses 5 nonzero powers (5 terms of the series expansion). As expected, the best results were achieved using the double precision format (64 bits); however, one can expect a large cost in logic area.

Table 3 presents the MSE and latency in clock cycles for the Taylor series expansion, using a simple precision format (32 bits). It can be observed that the smaller error achieved by the $\text{FP}_{\text{Taylor}}$ core is over E^{-15}, for five powers (x^{11} polynomial degree). With more than five powers in the series expansion, the error due to the Runge's phenomenon is increased [29] when using polynomial interpolation with polynomials of high degree.

Table 2: MSE of the floating-point operators.

FP-Core	24(6, 17)	32(8, 23)	43(11, 31)	64(11, 52)
Add/sub	$2.27E-7$	$2.76E-11$	$3.24E-15$	$1.26E-17$
Multiplier	$9.53E-4$	$1.53E-7$	$1.49E-11$	$5.87E-16$
Taylor sin/cos	$6.31E-9$	$8.80E-9$	$6.26E-9$	$1.94E-16$
Taylor atan	0.073	0.0014	0.015	0.015

Table 3: MSE and Latency of the FP_{Taylor} architecture.

Powers	$\sin(x)$ $[-\pi/2\ \pi/2]$	$\cos(x)$ $[-\pi/2\ \pi/2]$	$\text{atan}(x)$ $[-100\ 100]$	Elapsed time clock cycles
2	$1.59E-03$	$3,52E-02$	0.00137	11
3	$1.56E-06$	$5,58E-05$	0.00136	15
4	$7.01E-13$	$3.62E-11$	0.00136	19
5	$3.13E-15$	$1.15E-14$	0.00136	23
6	$2.13E-14$	$5.91E-15$	0.00136	27

Table 4: Synthesis results. Virtex2 XC2VP30.

Bit-width (Exp, Man)	Slices (13696)	LUTs (27392)	Mult18 × 18 (136)	Freq (MHz)
32(8, 23)	10528	20017	48	54.83
43(11, 31)	14589	27807	48	48.31
64(11, 52)	34031	64687	180	39.51

Table 5: Synthesis results. Virtex5 XC5VLX110T.

Bit-width (Exp, Man)	Slices (69120)	LUTs (69120)	DSP48Es (64)	Freq (MHz)
32(8, 23)	4051	14457	24	83.61
43(11, 31)	5349	19262	48	66.22
64(11, 52)	7901	98755	111	63.93

5. The FPGA Implementations

The FPGA implementation of the direct kinematics is carried out using Time-Constrained Scheduling (TC) [30], as showed in Figure 5. In fact, the circuit explicitly implements (4) to (15). To do that, some strategies can be used in order to allocate the respective hardware resources, achieving a previously defined cost function, which can represent time or/and resource restrictions. Time constrained and resource constrained scheduling algorithms are well-known techniques for leading with obtaining a suitable time execution and allocating the available hardware resources (for instance, arithmetic operators). For achieving the circuit depicted in Figure 5 (from (4) to (15)), only resource restrictions were imposed, where 4 multipliers and two add/sub units were used, apart from 8 trigonometric units. With the proposed scheduling the lowest time execution was tried to be accomplished. In contrast, a time constrained scheduling would need more arithmetic units for accomplishing

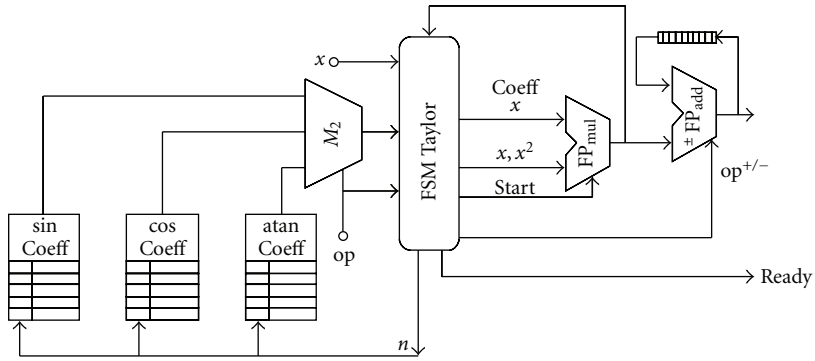

Figure 4: IEEE-754 format.

a predefined restriction in execution time. Anyway, this kind of algorithms does not guarantee an optimal solution [30].

The control of the overall architecture was achieved by a Finite State Machine (FSM), and the overall architecture was described in VHDL using the Xilinx ISE 10.1 development tool, making use of the floating-point arithmetic units. Eight cores for computing transcendental functions were required: four for computing the sin function and the other four for computation the cos function. The necessary addition and subtraction operations were achieved by applying the $FP_{add/sub}$ and $FP_{multiplier}$ cores presented in [28] (see Section 3). For computing the sin and cos functions, a Taylor series expansion was implemented, using a $FP_{add/sub}$ and a $FP_{multiplier}$ cores, with a similar architecture to the presented in [26]. The $FP_{Taylorcos}$ and $FP_{Taylorsin}$ were achieved using 10 and 11 nonzero powers for cosine and sine respectively and an elapsed time of 23 clock cycles for cosine and 26 for sine.

It can be noticed that the orientation and position results are calculated from the following steps: in step 3 the x_z and y_z, in step 4 the parameter p_z, in step 5 the z_z and p_x, in step 6 the p_y parameter is computed, in step 7 the z_x and z_y, in step 10 the x_x and y_x, and finally in step 11 the x_y and y_y (see Figure 5).

In order to compare the performance of the proposed architecture, the direct kinematics was also implemented in software using a C code language running on an FPGA embedded PowerPC processor, using the Xilinx Platform Studio development tool. This approach allows the designer for having a just comparison between both hardware and software developments, which are running in the same environment (namely, the same FPGA with the same clock). To do that, two hardware peripherals were added to the PLB bus of the PowerPC processor: the first one for counting the clock cycles and the second one for accomplishing a RS-232 serial communication to a Matlab environment. The counter is used for measuring the processing time in software by means of enable and stop ports (both addressed by the PLB bus), which are controlled by specific software instructions. Figure 6 shows the connection between the PowerPC processor and the hardware peripherals.

6. Results

6.1. Synthesis Results. Synthesis results are shown in Table 4 for the Xilinx Virtex2 family (chip xc2vp30) using the ISE 10.1 development tool. It can be observed that the cost and performance of the architecture are presented in Figure 5 for different bit-width (exponent and mantissa), including both the simple precision and double precision (IEEE-754 standard). As expected, large bit-width representations are more expensive in terms of area if compared with small bit-width representations. Additionally, the performance is better when using small bit-widths.

It can be seen that the 43 and 64 bit-widths representations exceeded the hardware resources available in the specific Virtex2 FPGA (see Table 4). However, it is important to take into account that the xc2vp30 is not the largest device of the Xilinx Virtex2 FPGA family. Modern FPGAs devices have a large number of configurable logic blocks (CLB), dedicated DSP blocks, among other facilities which are suitable for mapping complex algorithms directly in hardware. Table 5 shows the cost in logic area of the same architecture using a Virtex5 FPGA (chip xc5vlx110t). It is important to take into account that the Virtex5 FPGA family uses embedded DSP48Es blocks, which considers 18×25 bits multipliers. It can be observed that the operational frequency has been increased; however the double precision representations exceed the available number of LUTs. However, as will be explained below, simulation results point out that the single precision format (32 bits) achieves similar error results to large bit-width representations. Therefore, one can conclude that the 32 bit implementation is suitable in terms of hardware resources consumption and error associated for computing the direct kinematics of a 5 *D.o.f* spheric robot.

6.2. Simulation Results. The implemented direct kinematics hardware architecture (see Figure 5) was simulated using the ModelSim 6.3 g simulator tool. A simulation environment to validate the architecture behavior was developed in Matlab. This developed tool created the floating-point test vectors for different bit-widths. The binary floating-point results were analyzed in the simulator environment in order to calculate the Mean Square Error (MSE) of the implemented

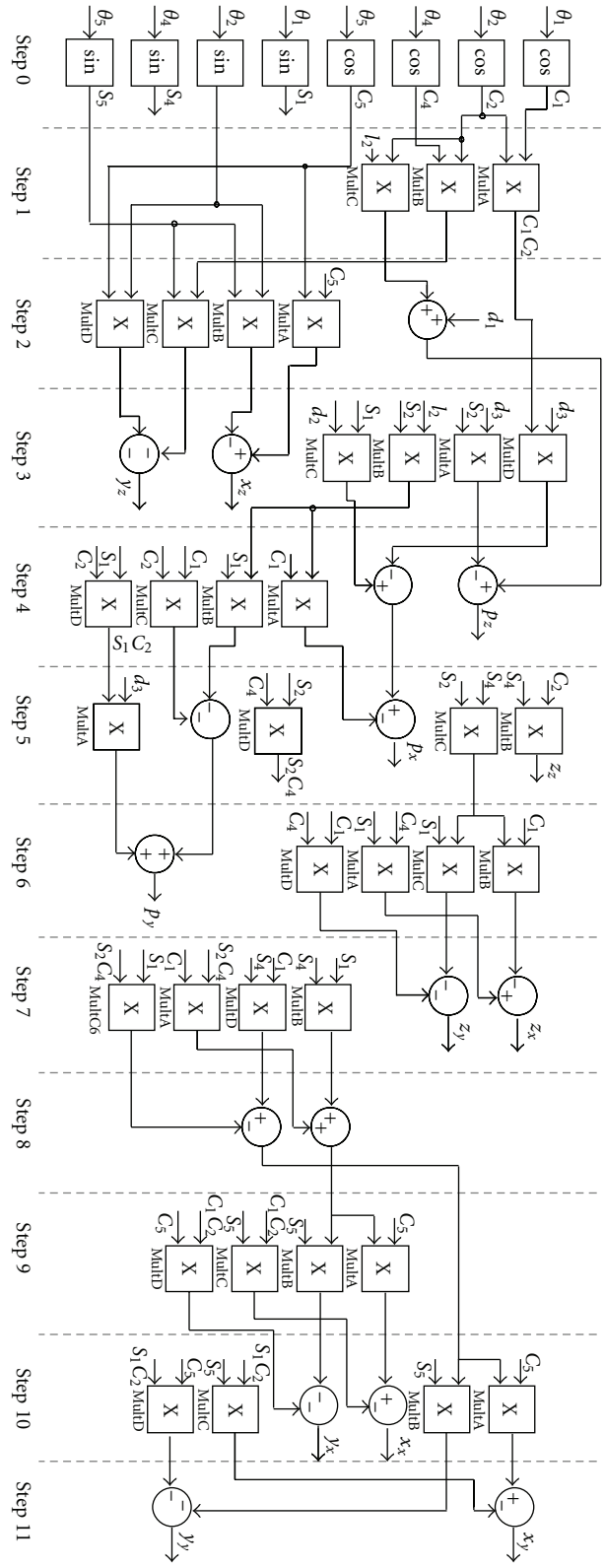

Figure 5: Hardware architecture for computing the direct kinematics using FSM.

Table 6: MSE for different bit-width representation.

Bit-width	axis	x	y	z	p
32(8, 23)	x	$3.61E-12$	$1.44E-11$	$1.75E-14$	$5.42E-06$
	y	$9.08E-13$	$3.55E-12$	$1.06E-14$	$1.33E-06$
	z	$1.44E-11$	$3.63E-12$	$4.46E-12$	$2.07E-10$
43(11, 31)	x	$3.63E-12$	$1.45E-11$	$1.62E-14$	$5.44E-06$
	y	$8.88E-13$	$3.58E-12$	$3.28E-16$	$1.34E-06$
	z	$1.45E-11$	$3.64E-12$	$4.45E-12$	$7.07E-11$
64(11, 52)	x	$3.63E-12$	$1.45E-11$	$1.62E-14$	$5.44E-06$
	y	$8.88E-13$	$3.58E-12$	$3.29E-16$	$1.34E-06$
	z	$1.45E-11$	$3.64E-12$	$4.45E-12$	$7.07E-11$

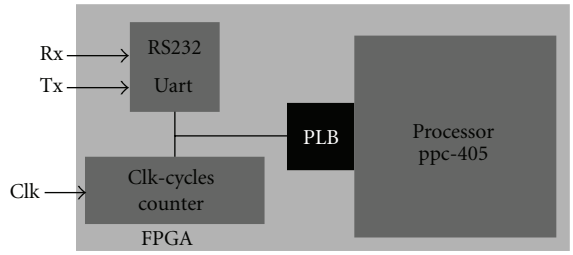

Figure 6: FPGA Architecture for Software Implementation.

Table 7: Latency for each variable.

Stage	Variable	Latency
3	x_z, y_z	43
4	p_z	46
5	z_z, p_x	49
6	p_y	52
7	z_x, z_y	55
10	x_x, y_x	64
11	x_y, y_y	67

architecture, for which the set of Matlab results in double precision arithmetic were used as a statistical estimator.

For computing the MSE of the hardware architecture, 100 input values were used. The input ranges were -90° to 90° for the angles θ_1, θ_4, and θ_5, -35° to 20° for the angle θ_2, and 160 mm to 760 mm for the D_3 parameter.

Table 6 shows the MSE for different bit-width representations. Notice that the magnitude order of MSE does not have significant changes. Simulations using a bit-width representation smaller than a 32 bit-width representation did not present satisfactory results, due to saturation in the floating-point operators.

The software implementation (see Figure 6) was validated using a simulation environment developed in Matlab and the serial RS232 interface for sending the input values and receiving the output values (position, orientation, and time processing in clock cycles). The aim of this simulation is to compare the processing time of the direct kinematics between software and hardware approaches and not to compare the error in the computations.

The elapsed time using the hardware architecture for each variable and each step is shown in Table 7. The overall direct kinematics is computed in 67 clock cycles. According to synthesis results, the maximum operational frequency is around 54 MHz for a single precision format, thus, all the computations are performed in 1.24 μs. The software implementation has a processing time of 1.61036 ms for all computations of the direct kinematics in floating-point using a single precision. For comparison purposes we have only used the obtained operational frequency on the VirtexII FPGA family (54 MHz) due to the fact that this device has a PowerPC embedded processor. It can be observed that the hardware architecture is over 1298 times faster than the software implementation, which means a considerable speed-up in the processing time of the direct kinematics. These elapsed time results can be dependent on the operation scheduling used in the architecture.

7. Conclusion

An FPGA implementation of the direct kinematics of a spherical robot manipulator using floating-point units was presented. The proposed architecture was designed using a Time-Constrained Scheduling, using 8 cores for computing transcendental functions (sine and cosine), and sharing 4 multiplier floating-point cores and 2 addition/subtraction floating-point cores for computing the arithmetical operations. Synthesis results show that the proposed hardware architecture for direct kinematics of robots is feasible in modern FPGAs families, in which there are plenty of logic elements available.

The overall computations were successfully performed and were validated using the Matlab results as a statistical estimator. The computation time to implement the direct kinematics in hardware is over 1.24 μs, whereas the same formulation implemented in software using a PowerPC processor needs 1.61 ms, obtaining a considerable speed-up in the processing time.

As future works, a hardware architecture for computing the inverse kinematics of the proposed robot manipulator will be implemented using the floating-point cores currently developed, such as division, square root, and arctangent functions. In addition, we intend to perform a performance

comparison of the proposed hardware architecture with a multicore embedded processor approach.

References

[1] A. Y. Zomaya, "Parallel processing for robot dynamics computations," *Parallel Computing*, vol. 21, no. 4, pp. 649–668, 1995.

[2] P. McKerrow, *Introduction to Robotics*, Adison-Wesley, New York, NY, USA, 1991.

[3] S. Kilts, *Advanced FPGA Design*, John Wiley & Sons, Hoboken, NJ, USA, 2007.

[4] U. Meyer-Baese, *Digital Signal Processing with Field Programmable Gate Arrays*, Springer, Berlin, Germany, 2001.

[5] R. Andraka, "Survey of CORDIC algorithms for FPGA based computers," in *Proceedings of the ACM/SIGDA 6th International Symposium on Field Programmable Gate Arrays (FPGA '98)*, pp. 191–200, February 1998.

[6] X. Shao, S. Dong, and J. K. Mills, "A new motion control hardware architecture with FPGA-based IC design for robotic manipulators," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA '06)*, pp. 3520–3525, May 2006.

[7] X. Shao, D. Sun, and J. K. Mills, "Development of an FPGA-basedmotion control ASIC for robotic manipulators," in *Proceedings of the IEEE International Intelligent Control and Automation*, pp. 8221–8225, 2006.

[8] Y. S. Kung and G. S. Shu, "Development of a FPGA-based motion control IC for robot arm," in *Proceedings of the IEEE International Conference on Industrial Technology (ICIT '05)*, pp. 1397–1402, December 2005.

[9] Y. S. Kung and G. S. Shu, "Design and implementation of a control IC for vertical articulated robot arm using SOPC technology," in *Proceedings of the IEEE International Conference on Mechatronics (ICM '05)*, pp. 532–536, July 2005.

[10] M. Petko and G. Karpiel, "Hardware/software co-design of control algorithms," in *Proceedings of the IEEE International Conference on Mechatronics and Automation (ICMA '06)*, pp. 2156–2161, June 2006.

[11] R. Wei, M. Jin, J. J. Xia, Z. Xie, and H. Liu, "Reconfigurable parallel VLSI co-processor for spacerobots using FPGA," in *Proceedings of the International Conference on Robotics and Biomimetics (ROBIO '06)*, pp. 374–379, 2006.

[12] R. Wei, X. Gao, M. Jin et al., "An FPGA based hardware architecture for HIT/DLR hand," in *Proceedings of the International Conference on Intelligent Robots and Systems*, pp. 523–528, 2005.

[13] S. Galvan, D. Botturi, and P. Fiorini, "FPGA-based controller for haptic devices," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '06)*, pp. 971–976, October 2006.

[14] Y.-S. Kung, K.-H. Tseng, C.-S. Chen, H.-Z. Sze, and A-P. Wang, "FPGA-implementation of inverse kinematics and servo controller for robot manipulator," in *Proceedings of the IEEE International Conference on Robotics and Biomimetics (ROBIO '06)*, pp. 1163–1168, 2006.

[15] J. U. Cho, Q. N. Le, and J. W. Jeon, "An FPGA-based multiple-axis motion control chip," *IEEE Transactions on Industrial Electronics*, vol. 56, no. 3, pp. 856–870, 2009.

[16] Z. Yili, S. Hanxu, J. Qingxuan, and S. Guozhen, "Kinematics control for a 6-DOF space manipulator based on ARM Processor and FPGA Co-Processor," in *Proceedings of the International Conference on Industrial Informatics (INDIN '08)*, pp. 129–134, 2008.

[17] D. Hildenbrand, H. Lange, F. Stock, and A. Koch, "Efficient inverse kinematics algorithm based on conformal geometric algebra—using reconfigurable hardware," in *Proceedings of the 3rd International Conference on Computer Graphics Theory and Applications (GRAPP '08)*, pp. 1–8, Springer, 2008.

[18] R. Hartenstein, "Why we need reconfigurable computing education," in *Proceedings of the IEEE Workshop on Reconfigurable Computing Education*, pp. 1–4, 2006.

[19] R. P. Paul, *Robot Manipulators—Mathematics, Programming, and Control*, MIT Press, Boston, Mass, USA, 1981.

[20] "IEEE standard for binary floating-point arithmetic," ANSI/IEEE Std 754-1985, August 1985.

[21] X. Wang, *Variable precision floating-point divide and square root for efficient FPGA implementation of image and signal processing algorithms*, Ph.D. dissertation, Northeastern University, 2007.

[22] J. Zhou, Y. Dong, Y. Dou, and Y. Lei, "Dynamic configurable floating-point FFT pipelines and hybrid-mode CORDIC on FPGA," in *Proceedings of The International Conference on Embedded Software and Systems (ICESS '08)*, pp. 616–620, 2008.

[23] J. Valls, M. Kuhlmann, and K. K. Parhi, "Evaluation of CORDIC algorithms for FPGA design," *Journal of VLSI Signal Processing*, vol. 32, no. 3, pp. 207–222, 2002.

[24] F. E. Ortiz, J. R. Humphrey, J. P. Durbano, and D. W. Prather, "A study on the design of floating-point functions in FPGAs," *Field Programmable Logic and Applications*, vol. 2778, pp. 1131–1135, 2003.

[25] J. Detrey and F. De Dinechin, "Floating-point trigonometric functions for FPGAs," in *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL '07)*, pp. 29–34, August 2007.

[26] D. M. Muñoz, D. F. Sanchez, C. H. Llanos, and M. Ayala-Rincón, "Tradeoff of FPGA design of floating-point transcendental functions," in *Proceedings of the IEEE International Conference on Very Large Scale Integration*, pp. 1–4, 2009.

[27] D. M. Muñoz, D. F. Sanchez, C. H. Llanos, and M. Ayala-Rincón, "FPGA based floating-point library for CORDIC algorithms," in *Proceedings of the 6th Southern Programmable Logic Conference (SPL '10)*, pp. 55–60, March 2010.

[28] D. F. Sánchez, D. M. Muñoz, C. H. Llanos, and M. Ayala-Rincón, "Parameterizable floating-point library for arithmetic operations in FPGAs," in *Proceedings of the 22nd ACM Symposium on Integrated Circuits and Systems Design (SBCCI '09)*, pp. 253–254, 2009.

[29] C. Runge, "ber empirische funktionen und die interpolation zwischen quidistanten ordinaten," *Zeitschrift für Mathematik und Physik*, vol. 46, pp. 224–243, 1901.

[30] D. Gajski, *Principles of Digital Design*, Prentice Hall, New York, NY, USA, 1997.