Proceedings of the Pakistan Academy of Sciences:
A. Physical and Computational Sciences 55 (1): 59–69 (2018)

ISSN: 2518-4245 (print), 2518-4253 (online)

Pakistan Academy of Sciences

Research Article

Design and Implementation of FPGA-based Concurrent Controller

Muhammad Majid Gulzar[1*], Ali Faisal Murtaza[1], K M Hasan[2], Syed Tahir Hussain Rizvi[3], Muhammad Yaqoob Javed[4], and Sajid Iqbal[2]

[1]Faculty of Engineering, University of Central Punjab, Lahore, Pakistan
[2]University of Engineering and Technology, Lahore, Pakistan
[3]Politecnico di Torino, Turin, Italy
[4]COMSATS Institute of Information Technology, Lahore, Pakistan

Abstract: In this paper, a concurrent motion control system is designed and implemented to control and achieve the consensus of a multi-axis structure using inverse kinematic technique. Speed and precision were the main targets. Thus a synthesizable model to support floating point calculations is presented using a combinational divider. This model is used to implement trigonometric equation using Look up Tables (LUT) and hence can easily be implemented on FPGA devices. The gate level demonstration of the entire model containing Arithmetic Logic Unit (ALU), multiplier and divider are also presented. As FPGA has a concurrent structure for high-speed arithmetic calculations, which can be utilized for parallel control of several motors, so this algorithm has improved the efficiency and has reduced execution time from 5.2 μsec to 1.4 μsec with an accuracy of ±1 to manipulator position. For more precision, the trade-off is between accuracy and execution time. Synthesis model to support floating point division calculations up to n-bits is designed, where implementation results for floating digit 1 to 11 are given with their time lag, slices and LUT used. The test points were verified in simulation and on hardware platform which exhibits the high speed implementation of the proposed model.

Keywords: VHDL, FPGA, Spatan-3 kit, Xilinx, floating point, inverse kinematics

1. INTRODUCTION

There is a significant requirement in various engineering automation industries for accuracy and high speed processing. The improvement execution would have several gains in terms of manufacturing efficiency, preciseness and processing time [1-4]. Therefore high speed accurate motion control model is very challenging and demanding research focus in different areas to manage the motion of various types of multiple-axis machinery. Mostly the controller for motion control systems rely on a microprocessor or Digital Signal Processor (DSP), and the system also consists of some additional circuits like interface and memory cards or may have some software motion libraries [5-7]. So multiple functions are required by motion controllers for immediate and precise execution of such complicated and complex tasks [8-10].

Moreover, standard arithmetic computation techniques for such functions have various complexities. It may include iterative computations for complicated multiplication, division, non-linear and trigonometric functions. This condition leads to a complicated hardware structure, exclusive computation, enhanced structure size, high power consumption and expensive design [11, 12]. As FPGA technology is practical that could be reconfigured and implemented at the same time [13]. In addition, it has some salient features like low power consumption, short cycle design, high density and programmability, which makes it to suitable for digital systems as well [14, 15].

So a better consensus performance of multiple-

Received, March 2017; Accepted, March 2018
*Email: Muhammad Majid Gulzar: majidgulzar3@gmail.com

axis machine can be achieved using FPGA because it provides an interchange between an Application Specific Integrated Circuit (ASIC) and a general standard processor [16]. Furthermore, the execution time for Personal Computers (PCs) and DSPs are comparatively high as they use sequential approach in contrast to FPGA-based system which has concurrent design. Also in FPGA, large size values can be processed due to the support of arbitrary length of data arrays.

Floating point computations are not easy to manage using FPGA, so an optional solution was to utilize DSP kit as an auxiliary device for floating digit computation. But it was an expensive solution, so the low-cost idea was to round floating points to the nearby whole integer wherever needed, particularly after the implementation of arc cosine, arc tangent and square root function. Also integer division and floating point in FPGA designs is hard to synthesize and execute due to a high demand for the structural resources of the kit. In VHDL, divide operator is not synthesizable and returns only one value as an output. Thus a synthesizable model of a divider is discussed which returns more compact solution with remainder and quotient as outputs. To solve the floating point problem a simple technique is developed which can be easily implemented on FPGA. Although due to recent advancement in FPGA technology, manufacturers introduced commercial intellectual property (IP) cores based FPGA and these operations can be performed but there is a compromise between support of operations and cost, especially in concurrent operation. Also in IP-core, there is drawback of latency as the involvement of pipelined synchronous cores and clock induce time lag and higher power consumption which is undesirable especially in very high speed design. The main aim of our work is to enhance the performance of the angular manipulator by:

- Precise estimation of the angles through inverse kinematics algorithm.
- Synthesis model to support floating point division calculations up to n-bits.
- Arithmetic and non-algebraic functions are implemented without using FPGA IP-core.
- Simultaneous movement of the manipulator's arms by giving them parallel instructions using the concurrent architecture of FPGA.

To verify the performance, the inverse kinematic algorithm is implemented in Xilinx® software (Version Xilinx® ISE 9.1) which has a processor of XC3S200, to create the execution file. The execution file is transferred to the FPGA (Spatan-3) architecture which is connected to the two-axis manipulator while the desired position is given through a computer keyboard. The complete setup is tested by giving the various angles to the manipulator. In each case, the manipulator shows the satisfactory consensus performance by reaching the desired position with excellent precision and all the desired positions have been reached by the manipulator within time duration of 1.4μsec with an accuracy of ±1 to manipulator position. For more precision, depending on the floating point digit there will be a slight increase in execution time.

In section 2, proposed block diagram is presented while section 3 briefly introduces inverse kinematic algorithm. Section 4 describes the concurrent control system using divider and floating point digits. Simulation results are shown in section 5. Results and discussion can be seen in section 6, whereas for verification of the proposed system, practical implementation is demonstrated in section 7. Finally, concluding remarks are made in Section 8.

2. PROPOSED MODEL

The proposed model is meant to operate the several axis machines to ensure the required objectives using FPGA. The comprehensive block diagram of software and hardware co-design of manipulator's arm with computer interface and FPGA kit is given in Fig.1. The appropriate approach for manipulator's arm computation was designed and compiled in Xilinx® using inverse kinematics technique.

3. INVERSE KINEMATICS ALGORITHM

3.1 Inverse Kinematics Calculation

Inverse kinematics technique computes the joints angles (θ_1,θ_2, andψ) according to manipulator's arm tip location (x,y) given in the Cartesian plan. Arithmetic calculations for the proposed approach are as follow.

$$x = L1cos(\theta_1) + L2cos(\theta_1 + \theta_2)$$

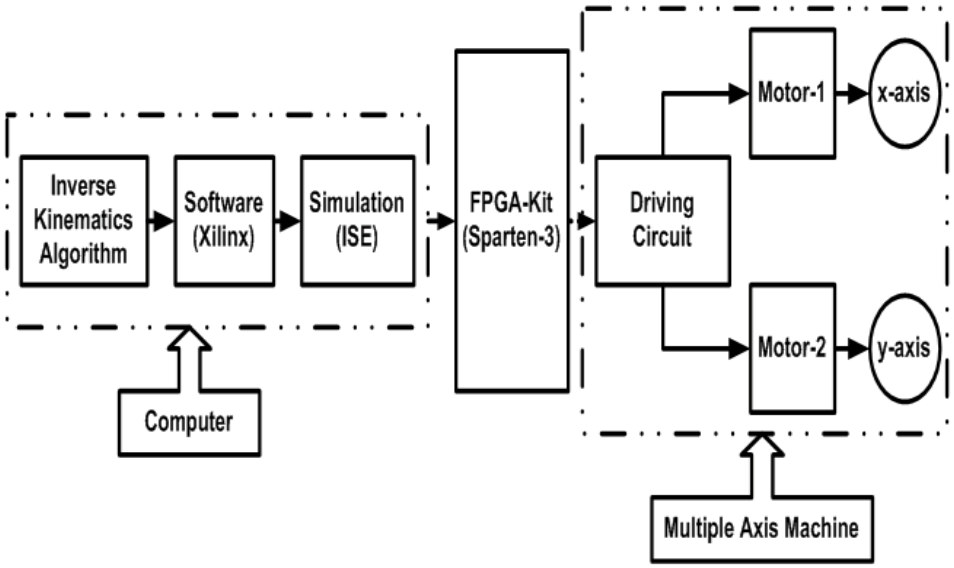


Fig. 1. Multiple axis machine with computer interface and FPGA kit.

$y = L1 \sin(\theta_1) + L2\sin(\theta_1 + \theta_2)$

$x^2 + y^2 = L1^2 + L2^2 + 2L1L2\cos(\theta^2)$

$$\Psi = \cos^{-1}\left(\frac{x^2+y^2 + L1^2 - L2^2}{2L1\left(\sqrt{x^2+y^2}\right)}\right) \quad (1)$$

Using "cosines law" for θ_1

$$L2^2 = L1^2 + \left(\sqrt{x^2+y^2}\right)^2 - 2L1\left(\sqrt{x^2+y^2}\right)\cos(\Psi)$$

Here we have two options

$\theta_1 = \beta + \Psi$, $\theta_1 = \beta - \Psi$

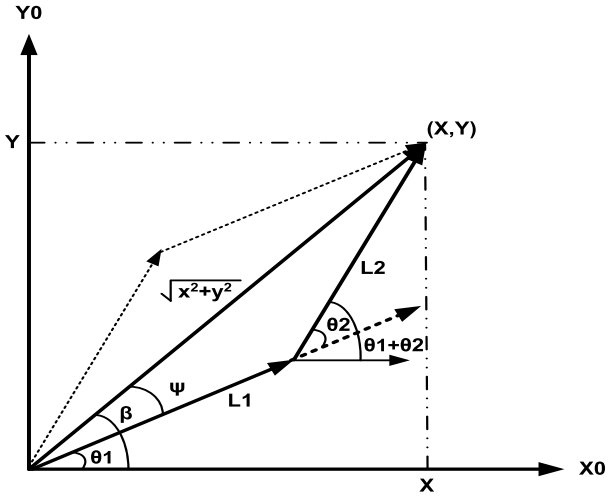


Fig. 2. Geometric solution of inverse kinematic.

$$\theta_1 = \tan^{-1}\left(\frac{y}{x}\right) - \cos^{-1}\left(\frac{x^2+y^2 + L1^2 - L2^2}{2L1\left(\sqrt{x^2+y^2}\right)}\right) \quad (2)$$

3.2 Inverse Kinematics Algorithm Implementation

The inverse kinematic algorithm can be implemented using sequential and concurrent approach. Brief introductions of both the techniques are given below.

3.2.1 Sequential Method

Due to the sequential implementation, DSP kit or any other pipeline device will take 26 steps to compute the manipulator's arm angles from the predefined Cartesian points. Fig.3 shows the complete sequential architecture with their calculations.

3.2.2 Concurrent Approach

As FPGA has a concurrent approach for arithmetic computations so it is selected for the effective compilation of inverse kinematic technique. It also ensures the fast arithmetic calculation resulting in improved multi-axis efficiency. Hence all motors of the manipulator's arm can be operated in concurrent way, ensuring the reduced execution time. This technique also

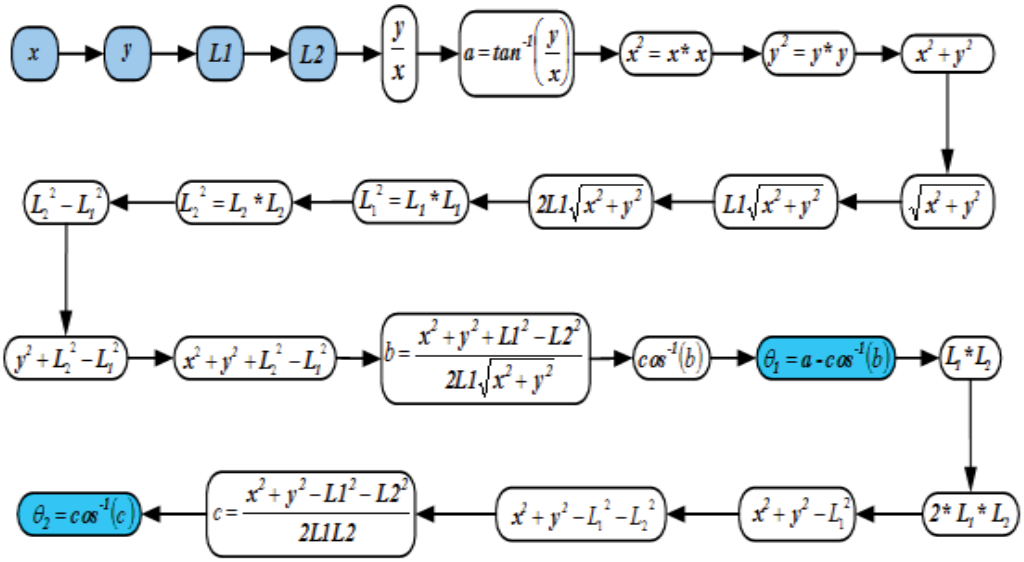


Fig. 3. Sequential solution of inverse kinematic algorithm.

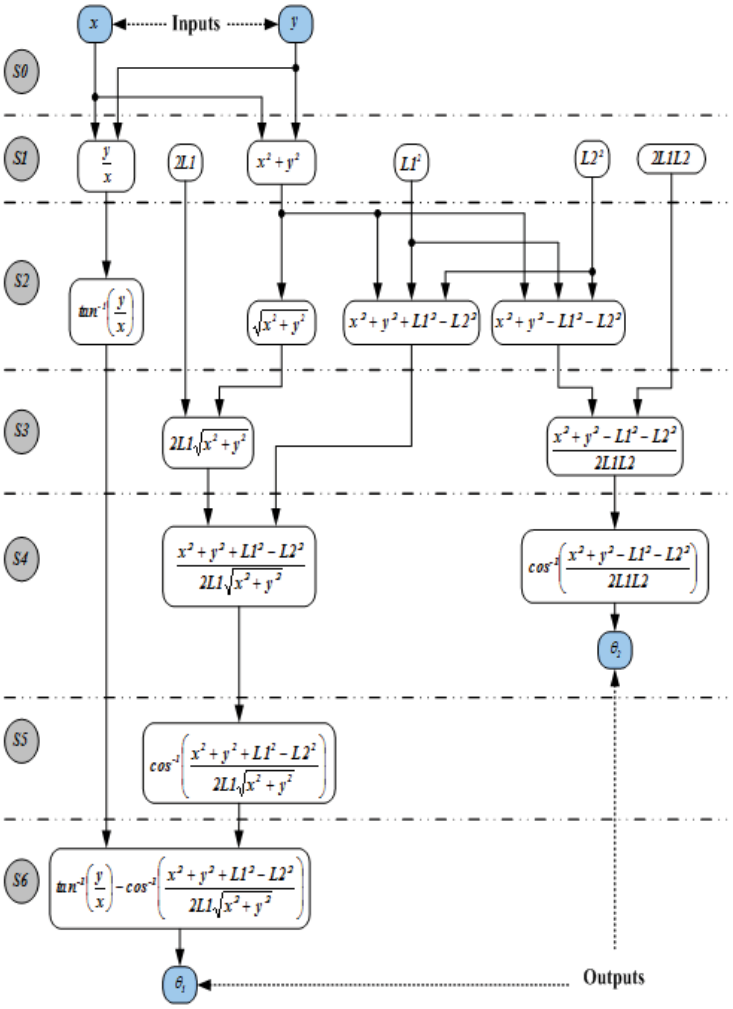


Fig. 4. Concurrent design of inverse kinematic algorithm.

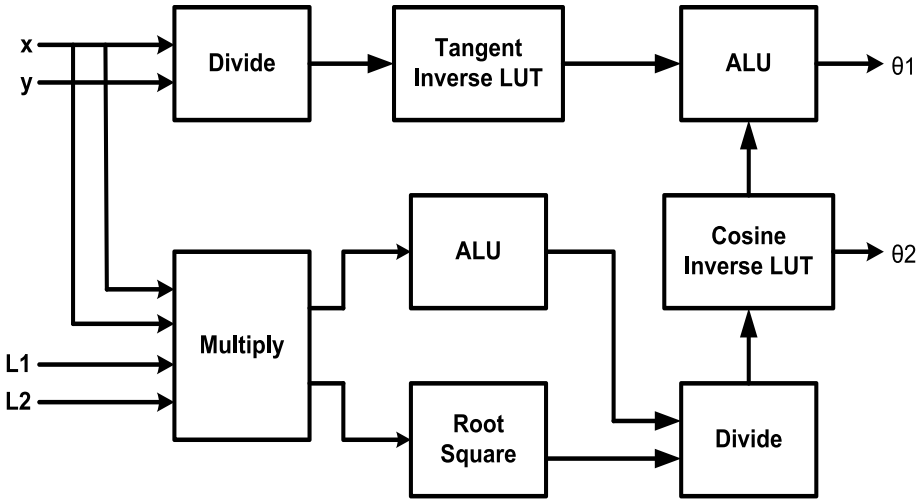


Fig. 5. Calculation module for inverse kinematic.

offers a low-power, highly sampled, flexible, and compact movement control design [17].

Due to parallel implementation, high computing power, low power consumption and fast sampling rate FPGA acquires only seven steps to compute the expected angles from the given Cartesian position. The complete concurrent design of inverse kinematic architecture can be seen in Fig.4.

The supporting language of FPGA is the VHDL which can execute multiple tasks concurrently as well as sequentially using process function. To perform multiple operations with high speed, VHDL's parallel compilation capability makes it superior selection to control multiple operations of complex arithmetical equations.

3.3 Calculation Module of Inverse Kinematics

Proposed design can be implemented on memory blocks of FPGA which are particularly known as LUT [18, 19]. As concurrent architecture can execute multiple blocks of code in parallel so the entire computational module with multiplier, divider, square root, ALU, arc tangent and arc cosine LUP is presented in Fig.5 for the given technique.

4. SYNTHESIS MODEL FOR CONCURRENT CONTROL

Spartan-3 FPGA kit has a processor of 50M Hz, therefore for execution of one instruction it will take 200nsec unless the internal delay should not be greater than this particular time.

Time required for executing one instruction on FPGA = 1/50M=200*nsec*

Total time required in pipeline architecture = 200 * 26 = 5200*nsec* = 5.2*μsec*

Total time required time in concurrent architecture = 200 * 7 = 1400*nsec* = 1.4*μsec*

So after implementing concurrent design using FPGA, the execution time has been cut down from 5.2μsec to 1.4μsec. This is just a ratio of 50M Hz processor while using external faster clock better high-speed result can be achieved. The algorithm comparison is presented in Table 1.

The purpose of using VHDL is to compute algorithm execution in parallel. Therefore, manipulator joints movement would be in parallel to achieve quicker response. But VHDL does not have square roots, divisions, tangent inverse and cosine inverse functions. In addition, FPGA does

Table 1. Architecture comparison.

	Number of Steps	Required Time (μsec)
Pipeline Technique	26	5.2
Concurrent Technique	7	1.4

Table 2. Implementation of division function.

a_input	Comparison	b_input	Y (Quotient)	Operation on 1st Column
1011	<	0011000	0	None
1011	<	0001100	0	None
1011	>	0000110	1	a_input- b_input
0101	>	0000011	1	a_input- b_input
0010 (rem)			0011(quot)	

not support floating point numbers. Different methods are used to compile these functions in VHDL. If only the given divider module is used then the speed will be increased and system will be quite simple, but if the precision is of more concern then along divider model floating point model should also be used.

4.1 The Divider Model

Synthesizable model of divider comes up from algorithm which is explained in [20, 21]. Here, simple comparison technique is used to perform division operation as shown in Table. 2.

Primarily, the number of bits of dividend (a_ input), divisor (b_input) and outputs (Quotient and Remainder) should be the same. If required, the condition can be fulfilled by padding zeros in front of the narrower operand. In this case, dividend, divisor and outputs (Quotient and Remainder) are of 4 bits (n+1). If decimal value of dividend is 11 and decimal value of divisor is 3, so for this particular case values of both quotient and remainder would be 2. Comparison is performed on shifted version of the divisor. The divisor is shifted left to reach the length of 2n+1 bits (7 bits =(2(3) + 1)).

Both dividend and divisor inputs are compared with each other. If the dividend is less than the divisor, the value of the quotient is 0. If the dividend is greater than the divisor, the value of the quotient would be 1 and the divisor would be subtracted from the dividend. This new outcome would substitute the value for the next comparison. This entire operation would be completed in n steps. After final iteration, updated value of the dividend would have a remainder and all bits received from comparison would make value of quotient. Here, the bits are added to the lowest positions and zero padded is done from left side. Also if the divisor is zero, then it will return all ones to the quotient and dividend as remainder.

Division operation model is implemented to calculate digits representing floating point. But the issue to apply inverse cosine and inverse tangent is that the inputs of these operations have floating point numbers while normally VHDL does not hold floating point digits. So there is one technique that can easily be implemented as to round the values to the nearest whole number for fast execution. After divider model, a LUT of inverse cosine and inverse tangent are saved and now this result would be used to choose particular value.

4.2 Floating Point Representation Model

Synthesizable model of the floating point representation comes from hand-written division. Above defined divider model is used to get values of quotient and remainder, these values can be used to extract value after decimal point. If 22 is divided by 7, remainder is 1 and quotient is 3. Now, this remainder can be used further to take digits after decimal point. Now the value of remainder is 1 that is smaller than divisor 7, it cannot be directly divided. Therefore, remainder is multiplied by 10, if the value is still smaller, than multiply again with 10. Now division of this value provides the first digit after decimal point and it will continue until n digits are made.

Only using the divider model, time will be reduced to 1.4μsec with an accuracy of ±1 degree. But for more precise calculation the floating point model can be used, which is an exchange between accuracy and time lag.

Table 3. Implementation results.

Floating Points	Time Lag (μsec)	LUT Used	Slices Used
1	1.4+0.06506	474/3840 = 12 %	264/1920 = 13 %
3	1.4+0.06730	988/3840 = 25 %	548/1920 = 28 %
5	1.4+0.06771	1502/3840 = 39 %	812/1920 = 42 %
8	1.4+0.06831	2273/3840 = 59 %	1260/1920 = 65 %
11	1.4+0.06891	3044/3840 = 79 %	1686/1920 = 87 %

Table 4. Test points with their respective angles.

(x,y)	θ_1	θ_2
(2,12)	57	77
(6,10)	34	84
(12,7)	15	47

Results of floating point implementation are given below in Table 3 where by using slices, critical path delay between input and output and number of LUT are compared for increasing number of digits after decimal point. These results are obtained from synthesis report generated from ISE 9.1i.

Accuracy of 11 decimal points can be achieved using 1686 slices out of 1920 slices, so nearly 87% of resources are utilizing. By using larger FPGA like XC3S1500, this model can be suitable for implementing floating representation. If the same code is executed in FPGA XC3S1500, it will utilize only 12% of resources because its total slices are 13312. So compromise is between accuracy and resources used, but this solution is implemented on cheaper hardware kit.

4.3 Angles Verification with test points

In the first half of the Cartesian plan three test points are selected for angle verification. These test points are used to compute θ_1 and θ_2 by using already calculated inverse kinematic equations as presented in equation (1) and (2). Table 4 demonstrates all the test points with their respective angles.

Geometrically these angles are also verified, which conclude that the actual measured angles and angles calculated from equation are same as shown in Fig. 6.

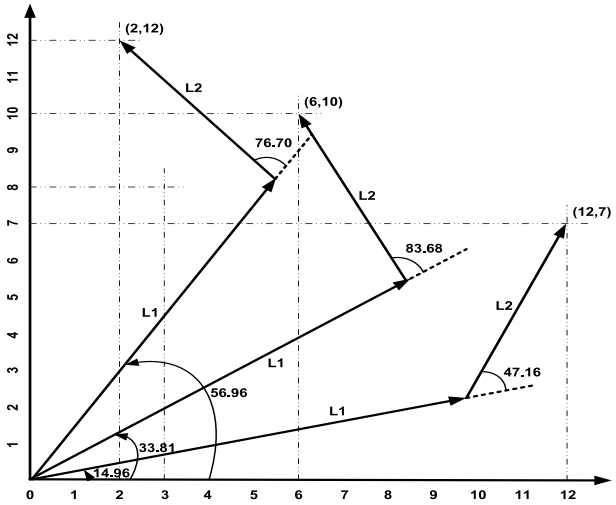


Fig. 6. Geometric solution of test points.

5. SIMULATION RESULTS

In FPGA the most considerable feature that must be considered while concurrent computation is implemented using floating point is the compromise between the need of acceptable precision and the expenditure of logic area [1]. So after applying the entire code in Xilinx®, ISE (Integrated Software Environment) simulator displayed the simulation test bench. In Fig. 7, it can be seen from test bench that for the 1st experimental point (x_1,y_1) = (2,12) the output angles are θ_1 =57 and θ_2 = 77, as previously calculated using equation (1), (2) and geometrically.

Test bench results illustrate that θ_2 and θ_1 are attained in 5th and 7th step respectively and total required time to compute both angles is 1.4μsec. In the same way remaining two test points can also be verified and their ISE simulation test benches are shown in Fig. 8 and Fig. 9 respectively. Test bench performance shows that the purposed model is preferable to the original one, considering the

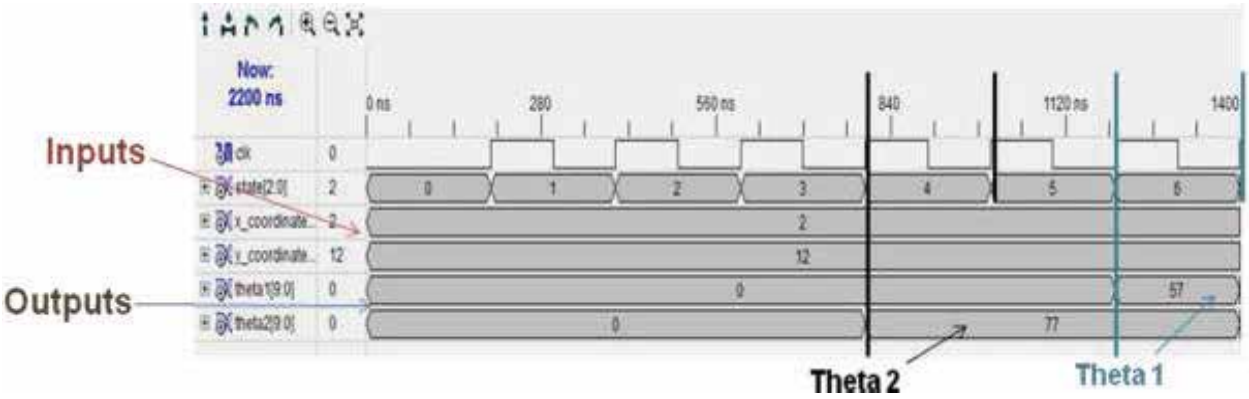


Fig. 7. ISE Simulator test bench for experimental point (2, 12).

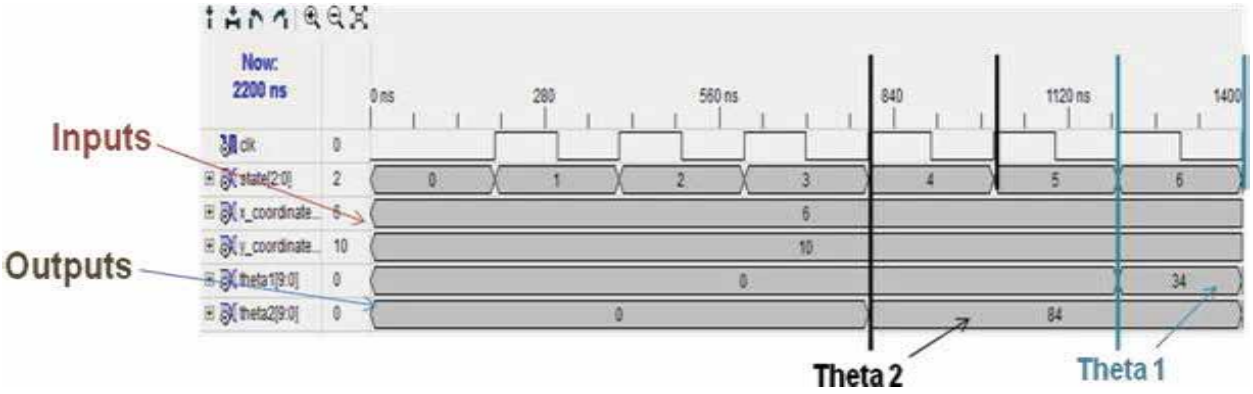


Fig. 8. ISE Simulator test bench for experimental point (6, 10).

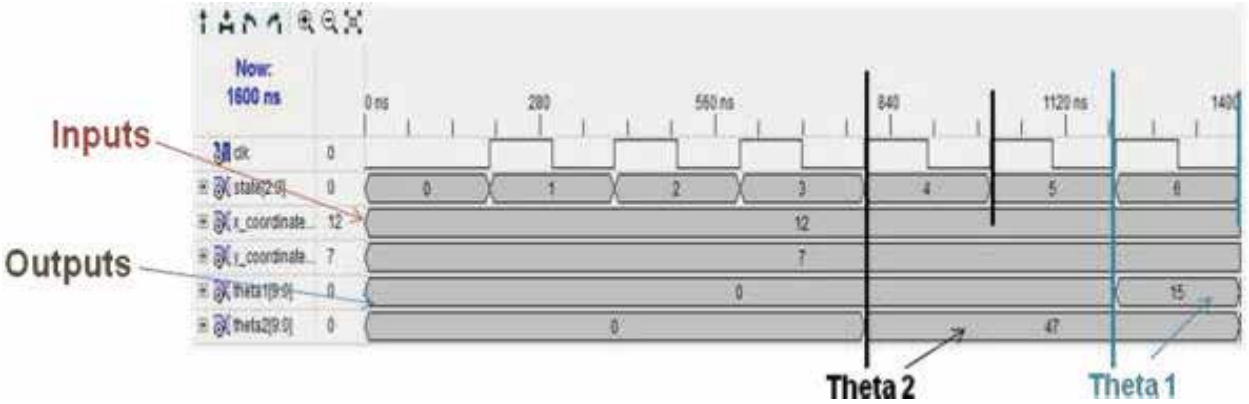
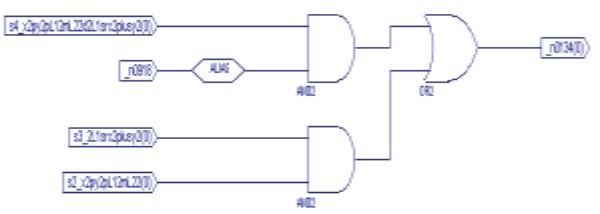


Fig. 9. ISE Simulator test bench for experimental point (12, 7).

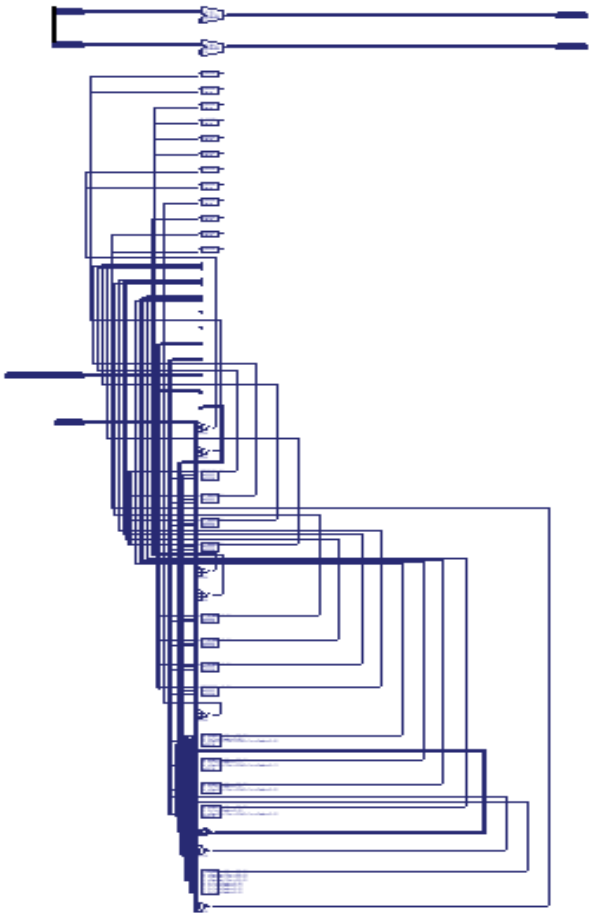
resources required and execution speed.

6. RESULTS AND DISCUSSION

Recently high speed computation algorithms required to compile multi real-time processing tasks within micro (μ) or mili (m) seconds in order to achieve fast processing. Therefore, pipeline computing machines and processors may not carry out the implementation requirements due to a massive number of transcendental functions and numeric operations. Also due to the involvement of vast calculations, such a task may be beyond the limitations of various sequential architectures [22]. So the preferable choice is FPGA due to its concurrent architecture processor. It offers a



(a)



(b)

Fig. 10. Gate Level Implementation: (a) One block; (b) Complete.

comparatively less processing time, regardless of the control algorithm complexity [17].

FPGA is a reprogrammable integrated circuits device where new codes can be burnt over repeatedly, after replacing old ones. Also large numbers of gates are required to implement the complete architecture which contains multiplier, divider and ALU. The above described model of floating point representation is implemented in Spartan 3 FPGA kit (XC3S200-FT256). The code embedded on processor kit mentions the capacity of gates in this chip. As XC3S200 have 220k gates, so this kit is appropriate for our system as gates required to execute our algorithm is less then 220k [23].

FPGA carries large number of logic blocks and according to VHDL code they are wired together. Logical operation and complex mathematical equations are carried out with the interconnection of these logic blocks. As the complete gate level illustration is quite congested, so only one block of gate level implementation is shown in Fig. 10 (a), whereas gate level Xilinx® implement of the whole algorithm with its interconnection is shown in Fig. 10 (b).

The proposed model is very efficient because, as the structure density of the latest devices is growing, the element cost is decreasing. Meanwhile, combinational design radiation is weaker when compared to the general IP-cores that can be used for sequential dividers as they do not require a clock that is needless for a combinational solution.

7. PRACTICAL IMPLEMENTATION

To verify the proposed algorithm experimentally, the code was burnt in FPGA Spartan-3 kit and exhibited on 7-segment display, where each pair of 7-segment ensured the corresponding resultant angles. The outcome of all three test points is shown in Fig. 11. The experimental set up of the manipulator is shown in Fig. 12 along with the angled sheet of test points.

Here, two servo motors are used which receives a PWM signal as input. So FPGA must generate a PWM signal of 50*Hz* (20*msec*) continuously for maintaining the particular position of shaft, otherwise it will come back to 0°. A pulse of 0.5 msec positions the shaft at 0° and pulse of 2.5 msec positions the shaft at180°. As rotation is limited from 0° to 180° and resolution is (2.5*msec*-0.5 *msec*=2*msec*). So the amount of rotation required for 1^{o} is 11.11μsec.

Fig. 11. Screenshot for different test points.

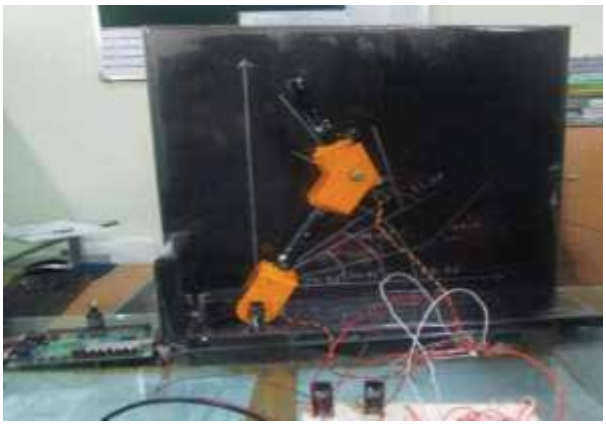


Fig. 12. Experimental setup of the robotic arm with angle sheet.

8. CONCLUSIONS

This paper performs the concurrent performance which reduced the execution time. Using synthesizable divider, inverse kinematics algorithm was implemented on VHDL. Along the system for its functional verification, code was burnt in Xilinx® software using ISE test bench as a simulation tool. Results of execution in Xilinx® Spartan-3, maximum time-lag and amount of FPGA resources used are also mentioned. The paper also introduces a synthesizable model to support floating point calculations. This is a VHDL component which can be extended to increase accuracy of design. Implementation results up to 11 decimal places are performed. Instead of using IP-core based FPGA, algorithm is designed and implemented on cheaper FPGA kit. Accuracy of results can be increased by just minor modifications, but the compromise is between accuracy and resources of hardware. Simulation results have confirmed the divider functionality.

Simulation and hardware implementation of an FPGA based controller have proven the effectiveness of the proposed method. Hence it has reduced the execution time from 5.2μsec to 1.4μsec due to its parallelism, supporting the high speed computation. In all cases, the two-axis manipulator shows the reasonable consensus performance by reaching the desired position with acceptable precision and performing the task within time limitations.

9. REFERENCES

1. Sánchez, D.F., D.M. Muñoz, C.H. Llanos, & J. M. Motta. FPGA implementation for direct kinematics of a spherical robot manipulator. In: *Proceedings International Conferenceon ReConFigurable Computing & FPGAs,* 9-11 Dec, Quintana Roo, Mexico, p. 416–421 (2009).
2. Saifee, M.A. Design and implementation of 2-axis circular interpolation controller in field programmable gate array (FPGA) for computer numerical control (CNC) machines and robotics. *International Journal of Computer Applications* 106 (13): p.1–7 (2014).
3. Aghdam, F.A., & S. S. Haghi. Implementation of high performance microstepping driver using FPGA with the aim of realizing accurate control on a linear motion system. *Chinese Journal of Engineering:* p. 1–8 (2013), DOI 10.1155/2013/425093.
4. Quang, N.A., Y.S. Kung, and Q.P.Ha. FPGA-based control architecture integration for multiple-axis tracking motion systems.In: *Proceedings of IEEE/ SICE International Symposim on System Integration,* 20-22 Dec, 2011, Kyoto, Japan, p. 591–596 (2011).
5. Fei, J., R. Deng, Z. Zhang, & M. Zhou. Research on embedded CNC device based on ARM and FPGA. In: *International Workshop on Automobile, Power & Energy Engineering.* 16: p. 818–824 (2011).
6. Lin, F.J., & P.H. Shen. Robust fuzzy neural network sliding-mode control for two-axis motion control system. *IEEE Transaction on Industrial Electronics* 53 (4): 1209–1225 (2006).
7. Chang, T.N., B.C.B. Cheng, & P. Sriwilaijaroen. Motion control firmware for high speed robotic systems. *IEEE Transactions on Industrial Electronics* 53 (5): 1713–1722 (2006).
8. Jose, D., P.N. Kumar, J.A. Shirley, & S. Ghayathrrie. Implementation of genetic algorithm framework for fault tolerant system on chip. *Information Japan* 17 (8): 3921-3945 (2014).
9. Chan, Y.F., M. Moallem, & W. Wang. Design and implementation of modular FPGA-based PID controllers. *IEEE Transactions on Industrial Electronics* 54(4): 1898–1906 (2007).
10. Simoni, L., M. Beschi, G. Legnani, & A. Visioli. Friction modeling with temperature effects for

industrial robot manipulators. In: *Proceedings of IEEE/RSJ International Conference onIntelligent Robots & Systems,* 28 Sept, Hamburg, Germany, p. 3524-3529 (2015).

11. Park, S.W., & J.H. Oh. Hardware realization of inverse kinematics for robot manipulators. *IEEE Transactions on Industrial Electronics,* 41 (1): p.45–50 (1994).
12. Gulzar, M.J., & Zain-ul-Abdeen. Optimal pitch control design of an airplane with analysis and verification using Matlab / Simulink. *Journal of the Institution of Electrical & Electronics Engineers Pakistan* (76): 26-30 (2012).
13. Rais, M.H, & M.H.Al Mijalli. FPGA based fixed width 4×4, 6×6, 8×8 and 12×12-bit multipliers using Spartan-3AN. *International Journal of Computer Science & Network Security* 11(2): p. 61–68 (2011).
14. Pimentel, J.C.G, & H. Le-Huy. A VHDL-based methodology to develop high performance servo drivers. In: In: *Proceedings of IEEE Industry Applications Conference,* 8-12 Oct, Rome, Itlay, p. 1502-1512 (2000).
15. Kung, Y.S., K.H. Tseng, & T.Y.Tai. FPGA-based servo control IC for X-Y table. In: Proceedings of *IEEE International Conference on Industrial Technology,* 15-17 Dec, 2006, Mumbai, India, p. 2913-2918 (2006).
16. Ying-Yu, T., & Tien-Sung Kuo. Design and implementation of all FPGA-based motor control IC for permanent magnet AC servo motors. In: *Proceedings of International Conference on Industrial Electronics, Control & Instrumentation,* 14 Nov 1997, New Orleans, USA, p. 943–947 (1997).
17. Cho, J.U., Q.N. Le, & J.W. Jeon. An FPGA-based multiple-axis motion control chip. 56 (3): 856–870 (2009).
18. Tao, Y, H. Lin, Y. Hu, X. Zhang, & Z. Wang. Efficient implementation of CNC position controller using FPGA. *IEEE International Conference on Industrial Informatics,*13-16 July, Daejeon, Korea, p. 1177–1182 (2008).
19. Iqbal, S., S.A. Qureshi & M.M. Gulzar. Concept building through block diagram using Matlab/ Simulink. *Journal of the Institution of Electrical & Electronics Engineers Pakistan* (66-67): 30-34 (2010).
20. Pedroni, V.A. *Circuit Design with VHDL*. MIT Press, London, England (2004).
21. Fedra, Z. & J. Kolouch. VHDL procedure for combinational divider. In: Proceedings of *International Conference on Telecommunication & Signal Processing,* 18-20 Aug 2011, Budapst, Hungary, p. 469–471 (2011).
22. Yang, Y., Y. Wu, & J. Pan. Parallel dynamics computation using prefix sum operations. *IEEE Robotics and Automation Letters* 2 (3): 1296-1303 (2017).
23. Fritz, D. *Spartan 3 FPGA Tutorial Design.* Oklahoma State University, USA (2005).