2014 IEEE International Conference on Robotics & Automation (ICRA)
Hong Kong Convention and Exhibition Center
May 31 - June 7, 2014. Hong Kong, China

A Highly Parallelized Control System Platform Architecture using Multicore CPU and FPGA for Multi-DoF Robots*

Sangok Seok[1], Dong Jin Hyun[1], SangIn Park[1], David Otten[2], and Sangbae Kim[1]

***Abstract*— This paper presents a control system platform architecture developed for multi-degrees of freedom (DoFs) robots capable of highly dynamic movements. In robotic applications that require rapid physical interactions with the environment, it is critical for the robot to achieve a high frequency synchronization of data processing from a large number of high-bandwidth actuators and sensors. To address this important problem in robotics, we developed a control system architecture that effectively utilizes the advantages of modern parallel real-time computing technologies: multicore CPU, the Field Programmable Gate Array (FPGA), and distributed local processors. This approach was implemented in the fast running experiments of the MIT Cheetah. In such a highly dynamic robot, the required control bandwidth is particularly high since the MIT Cheetah's leg actuation system is designed to generate high force (output torque up to 100Nm) with high bandwidth (400Hz electrical, 120Hz mechanical) with minimal mechanical impedance for fast locomotive capability. On the integrated control system, a multi-layered architecture is programmed. Inspired by the MapReduce model and the pipelining method, more than 50 processes are operated in parallel, and major processes among them are optimized to achieve the maximum throughput. The proposed architecture enables the control update frequency 4 kHz. With this control system platform, we achieved a high-force proprioceptive impedance control [1], and a trot-running up to 6 m/s with a locomotion efficiency rivaling animals [2]. This control system architecture is well suited for the future trend towards real-time computing system and, thus can be a candidate for a future standard robot control platform.**

I. INTRODUCTION

Robotic embedded systems have evolved along with the development of microprocessor and microcontroller system technologies. Since a 4-bit Instruction Set Architecture (ISA) appeared with sub-MHz CPU clock rate in 1971, the CPU clock rate had increased to over 3 GHz with a 64-bit ISA in the early 2000s. In 2005, a multi-core processor was developed to increase a system's overall thread-level parallelism, and a GPU was merged inside of the structure of a microprocessor [3]. The most recent processor is based on the System-on-Chip (SoC) solution which integrates a multicore CPU and a Field Programmable Gate Array (FPGA) [4], [5].

This emerging computing technology is especially suitable for robotic applications. As Bill Gates mentioned in [6], one of the most difficult challenges in robotics is the concurrency problem: how to process all the data from multiple sensors simultaneously, and send command instructions to a large number of actuators at the same time. The number of sensors and actuators in robots increases significantly to achieve more complex tasks. For example, Boston Dynamics BigDog has a total of 69 sensors [7], and 16 actuators [8]. In particular, if a robot with multi-DoFs is required to perform highly dynamic motion tasks, its controller needs to process dataflow in parallel and fast. Integration of multicore processors and FPGAs provides excellent embedded solutions for robotic applications because multicore processors enhance the parallel multi-thread performance, and FPGAs provide a softcore-implementation interface coupled to custom hardware more easily for parallel and fast processing. Note that because an FPGA has arrays of custom logic blocks [9], it can significantly reduce the execution time with a parallel

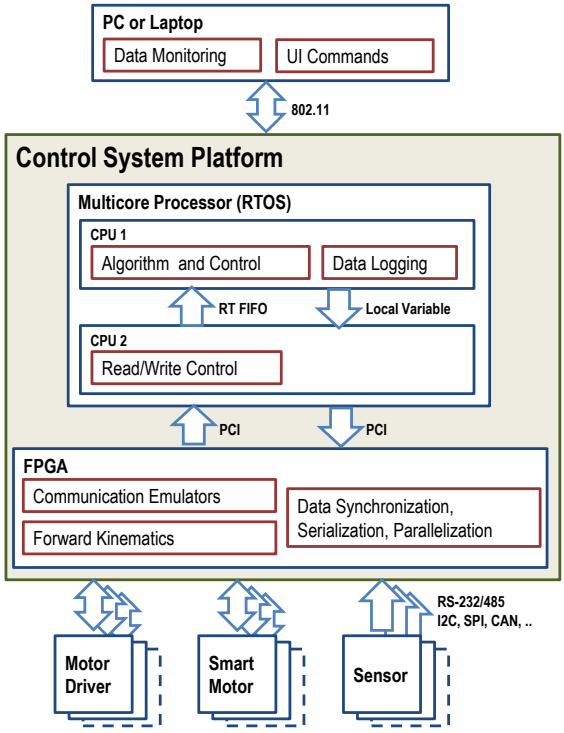


Fig. 1: Suggested control system platform architecture. The overall system has four layers which are a PC, a multicore processor, FPGA, and distributed actuators and sensors. The control system platform is a subset of the robot system which has a multicore processor and FPGA.

*This work was supported by the Defense Advanced Research Program Agency M3 program.

[1]Authors are with the Department of Mechanical Engineering, Massachusetts Institute of Technology, Cambridge, MA, 02139, USA, corresponding email: sangok at mit.edu

[2]Authors are with the Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA, 02139, USA

architecture [10]. Therefore, FPGAs are widely applied to industrial control systems and robot controllers [11], [12], [13], [14], [15].

In this paper, we propose a novel control system architecture which maximizes the performance, scalability, and compatibility by exploiting the advantages of parallel computing capability of these modern technologies (a state-of-the-art SoC processor or a single embedded controller that has multicore CPU and FPGA). Three major approaches implemented in this architecture are the MapReduce programming model, Single Instruction Multiple Data (SIMD) operation, and pipelining. This control system was implemented in a highly dynamic high bandwidth platform: the MIT Cheetah. In order to robustly stabilize this system, a fast control sample rate of the robots controller is indispensable. Moreover, the multi-tasking capability of the controller is significant for concurrent closed-loop control on all twelve DoFs. The proposed architecture enables the control update frequency of 4 kHz; more than 50 processes are running in parallel, and major processes among them are optimized to achieve the maximum throughput. With this control system platform, we achieved a high-force proprioceptive impedance control [1], and a trot-running up to 6 m/s with a locomotion efficiency rivaling animals [2].

The rest of the paper is organized as follows. Section II describes a control platform architecture, and Section III introduces the control system requirements for highly dynamic robots. Section IV explains system design methods such as MapReduce model, SIMD, and Pipelining. Section V shows implementation processes. Finally, conclusions and future work are discussed in Section VI.

II. Control System Platform Architecture

The design goal is to develop a high performance control system platform for highly complex robots that is scalable and compatible. For the scalability and compatibility, a distributed control systems architecture is used. All the distributed actuators and sensors are connected to the control system platform via industrial standard communication buses such as RS-232/485, I2C, SPI, CAN, etc. Since an FPGA usually has more than a hundred 3.3 V high-speed digital IOs, these can be programmed as communication ports with proper communication interface ICs. Also because an FPGA's processing speed does not change with the number of parallel processes, a star topology is used instead of a line topology. As shown in Fig. 1, each motor driver, smart motor, and sensor is connected via the dedicated communication port. However, each communication port can also be used for line topology such as RS-485 or CAN, depending on the application.

The FPGA layer has three main functions. For communication to each distributed device, communication emulators are programmed, as many as the number of connected devices. To achieve accurate control performance, all the incoming data are synchronized, and after the necessary calculations such as forward kinematics are done, the data are serialized and sent to the CPU 2 via the DMA channel in the PCI bus. After the multicore processor's processing is done, the FPGA layers get the data via the PCI bus. Finally, the data is parallelized and sent to each device at the same time. Depending on each device's update speed or supporting communication speed, communication emulators may run at different speeds. The synchronization loop speed is synced with the motor driver's update speed. Regardless of the reading speed of other devices, the loop reads the most recent data from each device.

The multicore processor has two CPUs, but three functions. CPU 2 only controls the receiving and transmitting flows in the FPGA layer, and CPU 1 runs the control algorithm as well as logging all the data. Because deterministic receiving and transmitting control through the system is the most important and critical process, CPU 2 is dedicated to only this job. Algorithm and control functions run under the highest priority, and data logging runs under the lowest priority in CPU 1. Even though the logging function is running under low priority, it does not lose data because data is saved in the Real-Time FIFO memory before it is saved to the drive.

A PC, a laptop, a smart phone, or a tablet can be used for the data monitoring system. Also, through the user interface, intermittent messages or commands can be sent to the control system platform with low latency through 802.11 g/n wireless communication.

The control system platform architecture is designed for a commercial single SoC or a single controller hardware such as Xilinx Zynq, Altera Cyclone and Arria, and National Instruments CompactRIO-9082. In this paper, the CompactRIO-9082 is used because it has the best hardware specification commercialized so far.

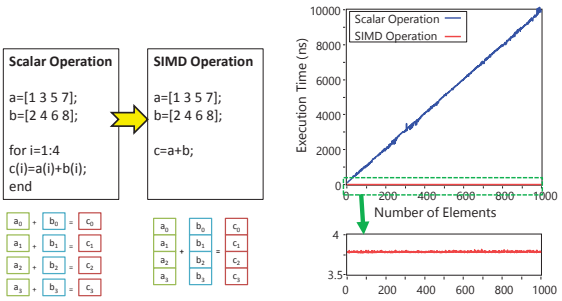
III. Control System Requirements

The control system is designed for highly dynamic robots, so it is critical for the system to achieve a high frequency synchronization of data processing from a large number of high-bandwidth actuators and sensors. However, for a dynamic robot that physically interacts with the environment, it needs high bandwidth actuators. In order to realize a wide range of virtual impedances and stabilize the high bandwidth limbs, high frequency control is a critical requirement. According to Khosla [16], a stably achievable maximum value of the proportional gain increases as the sampling rate increases while a maximum value of the implementable derivative gain is limited by measurement noise and quantization error. In the MIT Cheetah, the high allowable gain enables a larger range of programmable leg impedance. The proposed control system platform is designed to maximize the sampling rate.

IV. System Design Methods

A. MapReduce Programming Model in FPGA

MapReduce is a programming framework that can process huge amount of raw data in parallel over distributed stand-alone computers. It is developed and has been executed on Google's clusters, processing big data everyday [17].



(a) (b)

Fig. 2: (a) Conventional scalar operation and SIMD operation code (b) Benchmark test results of add instruction. Two one-dimensional arrays are added with scalar operation and SIMD operation. X axis is the number of elements in each array, and Y axis is execution time.

MapReduce has two steps: map and reduce. The map step takes a raw data set and processes it to another meaningful set of data which consists of pairs of key and value. The reduce step takes data from a map and combines it into a smaller set of data. For example, let's assume that there is a telephone book and you want to count the number of people whose first name is David. First, the book is separated into four pieces and distributed to four workers, who are helping you. Second, each worker counts the number on each pages. This is the map step. And then, you gather the results from four workers and calculate the total number. This is the reduce step. While you are calculating, other workers are counting the next page. By this way all workers can work efficiently in parallel.

This architecture framework can be applied to the FPGA layer to deal with processing the distributed data from actuators and sensors from the robot in parallel.

B. Single Instruction Multiple Data (SIMD) Operation

SIMD operation is designed for processing multiple data in parallel with a single instruction. This technology is developed for digital image processing [18] and widely adopted to most microprocessors.

Fig. 2(a) shows the comparison between a conventional scalar operation and a SIMD operation. For a scalar operation, the add instruction is executed four times to get the result. In the meantime, the add instruction is executed only once to get the result with a SIMD operation. Therefore, a SIMD operation is more efficient than a scalar operation.

Fig. 2(b) shows the benchmark test result of a scalar and a SIMD operation. Two one-dimensional arrays with double-precision floating-point elements are added by a scalar operation and a SIMD operation. The numbers of elements are increased from 2 to 1000, and execution times are measured for both operations. The programming language is LabVIEW, and the test code is run in NI CompactRIO-9082. Execution time of the scalar operation is increased linearly with respect to the number of elements, whereas the SIMD operation takes around 3.6ns for all calculations. This results show that the control system performance can be sustained regardless of the number of parallel processes if a SIMD operation is properly utilized.

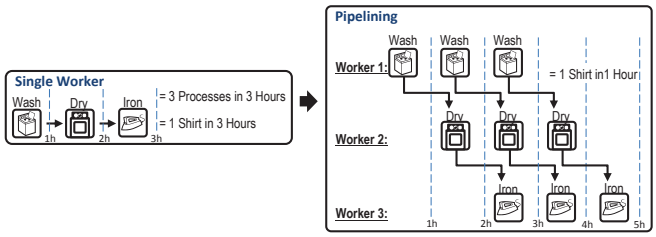


Fig. 3: Example of pipelining method. If there is one worker, it takes three hours per one shirt to finish the job. If there are three workers, it takes one hour per one shirt.

C. Pipelining on Multicore Processor

Pipelining is one of the parallel programming techniques widely used for improving serial tasks. The key idea is breaking down a serial task into separated tasks that can be carried out in an assembly-line fashion. [19]. Fig. 3 shows a simple example of the pipelining method. There is a laundry setup which has one washing machine, one dryer, and one iron. Assuming that each task takes one hour, the job takes three hours per one shirt. If there are three workers doing the laundry, it takes one hour per one shirt as in Fig. 3. By this way, overall throughput is improved three times.

Pipelining can be easily applied to a multicore processor. In general, a system control task consists of three distinct tasks: receiving data, processing data, and transmitting data. By running theses tasks on the dedicated CPU cores, system throughput can be improved.

V. Implementation

A. Bus Protocol Design for BLDC Motor Driver

The bus protocol between BLDC Drivers and the system controller is specially designed for minimizing data transfer time. The protocol basically used UART data frame which has 1 start bit, 8 data bits, and 2 stop bits. The maximum baud rate that the motor drivers can generate is 1 Mbps with RS-485/422 signals. It takes 11 μs per byte to transmit or receive 1 byte of data. When current commands are transmitted to the motor driver as binary two bytes; measured current, encoder position, and motor status are returned as binary four bytes. It takes 22 μs to transmit data, and 44 μs to receive data.

The motor driver's latency is measured by the scope. The scope is set to the envelope acquire mode to show all the possible responses from the motor driver. This allows us to measure the shortest latency the scope ever saw and the longest. The motor driver's control program takes about 17 μs to execute, but runs every 50 μs. The motor driver's response time is 3.8 μs, however when the motor driver is busy with running the control algorithm, it does not respond to the command until it is finished. The measured latency from the scope is a minimum of 3.8 μs and a maximum

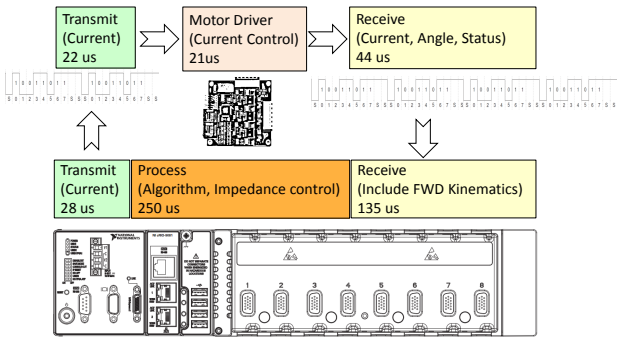


Fig. 4: Data transfer time in the motor driver. Transmitting 2 bytes current commands takes 22 μs and receiving 4 bytes data from the motor driver takes 44 μs. The motor driver's maximum system latency is 21 μs.

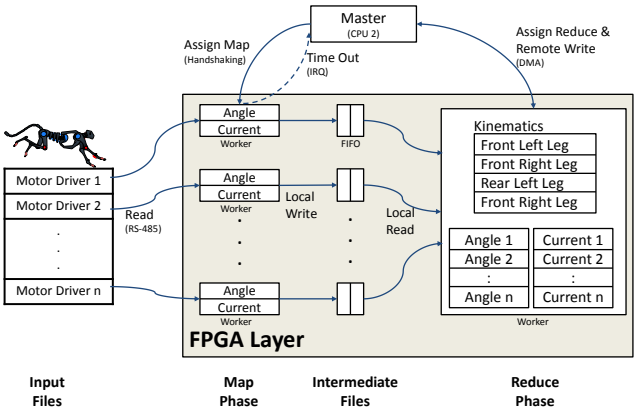


Fig. 5: Execution overview of the MapReduce model in the FPGA layer. The robot's sensing data from each motor driver is mapped and reduced in the FPGA layers and output data is sent to the CPU 2.

of 20.8 μs. As a consequence, a maximum of 87 μs is expected to transmit, process, and receive data. Overall data flow diagram through the system is shown in Fig. 4.

B. MapReduce Architecture on FPGA

MapReduce model is applied to the FPGA layers as shown in Fig. 5. MapReduce model is originally for processing big data with multiple computers, but the model is used to maximize the use of the FPGA resource and system throughput in this case. The sensing data from the robot is measured by the motor drivers. Each motor drivers has encoder angle and current data. Data from each motor driver is transmitted in parallel via RS-485 bus to the FPGA layer. In the FPGA layer, each worker is a single independent processing loop corresponding to a PC in actual data processing clusters. In the Map phase, received data bits are converted to angle and current data and written to the dedicated FIFO memory. In the Reduce phase, collected data from the FIFO memories are calculated and reduced as a Kinematics data array, angle array, and current array. Then, the array data set is sent to CPU 1 via DMA channel and processed with SIMD operation. Overall MapReduce tasks are assigned and controlled by CPU 1 in the dual-core processor.

C. Pipelining Method on Multicore Processor

1) Task Separation: For a single CPU control platform, the minimum total computing time ΔT for separate tasks such as data receiving (R), processing (P) and transmitting (T) is:

$$\Delta T = \Delta t_r + \Delta t_p + \Delta t_t = 500\mu s \quad (1)$$

where Δt_r, Δt_p, Δt_t are time durations taken for receiving, processing and transmitting tasks. ΔT is physically determined and can be considered to be fixed. Along with the time axis, each task is arranged on the single CPU without an idle CPU state as shown in Fig.7 where k represent the iteration of a closed-loop control. For this case, the system throughput, or rate of data transmitting, is $\frac{1}{\Delta T}$.

For a dual core CPU platform, two tasks can be concurrently executed. Therefore, system throughput is increased

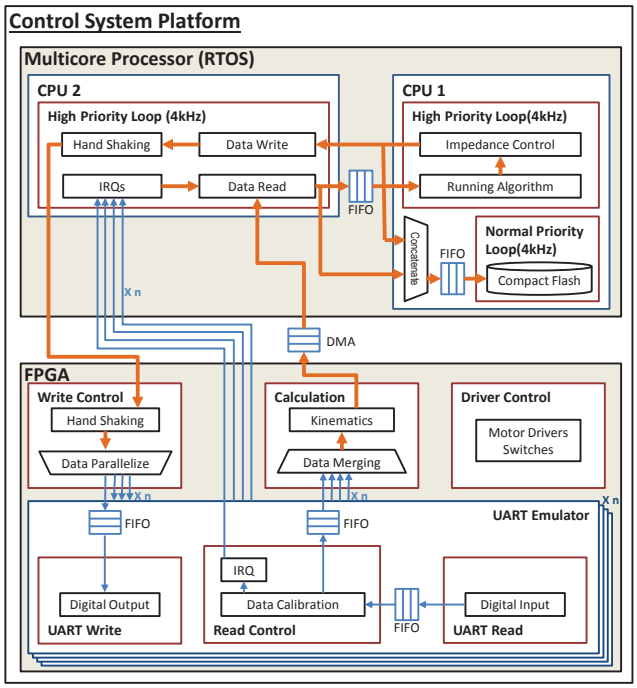


Fig. 6: Detailed architecture design and data flow diagram for FPGA and Multicore CPU

through task separation and its physically feasible maximum value of system throughput is $\frac{2}{\Delta T}$, twice as fast as the single CPU case. Therefore, if this maximum value is achieved through task separation and sequence optimization, the dual-core CPU executes tasks optimally. Generally, Δt_p is longer than Δt_r, Δt_t; the data processing task is assigned on CPU 1, and the the data receiving/transmitting are assigned on CPU 2 as shown in Fig.8. Note that in the k-th iteration block, the sequence of R_k-P_k-T_k along with the time axis should be conserved because of the data flow. In this task separation, the system throughput is still $\frac{1}{\Delta T}$ with idle states in each CPU. Therefore, sequence optimization is necessary to increase the system throughput, maximally up to $\frac{2}{\Delta T}$.

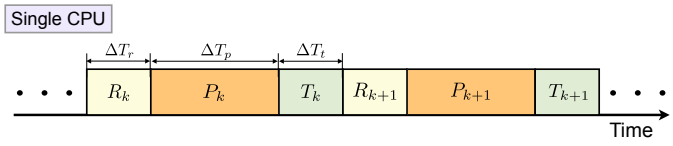


Fig. 7: Single CPU task sequence

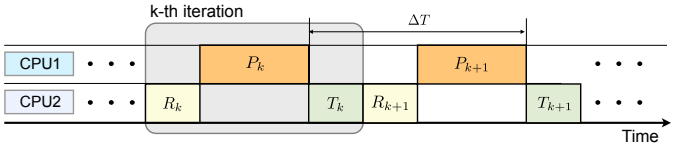


Fig. 8: Dual core CPU task separation

2) Sequence Optimization: The objective of the sequence optimization is to increase the system throughput by removing the CPU idle states as much as possible. If Δt_p is equal to $\Delta t_r + \Delta t_t$, the iteration blocks can be arranged by overlapping stacking of iteration blocks as shown in Fig.9. For this overlapping stack case, the system throughput rate

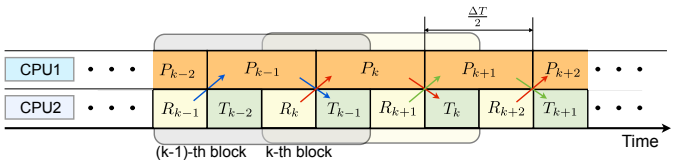


Fig. 9: Dual core CPU task sequence optimization

is $\frac{2}{\Delta T}$, the achievable maximum rate of data transmitting. Therefore, to satisfy the condition that $\Delta t_p = \Delta t_r + \Delta t_t$, sub-tasks under tasks R, P and T are properly assigned through task separation. Then, the sequence optimization in Fig.9 enables the maximized throughput, $\frac{2}{\Delta T} = 250\mu s$.

Fig. 6 shows overall system architecture design and data flow diagram applied to multicore CPU and FPGA. MapReduce model in the Fig. 5 and optimized pipelining method are programmed with LabVIEW.

D. Data logging

In general, data logging on the real-time (RT) target is avoided because 1) File IO is not deterministic, 2) introduces the priority inversion risk, and 3) becomes the speed bottleneck in RT application [20].

The system is required to log data with about 1 MB/s. Even though the embedded Compact flash Card (Swissbit CFast Card F-100 32GB)) has a writing speed of 120 MB/s, which exceeds the required data logging speed, it causes milliseconds level of jitters not only in the logging process itself but also in the other calculation processes because of the listed reason above.

However, utilization of the RAM in the CompactRIO-9082 can enable data logging to the system controller, because the system has abundant size of RAM (2GB). As shown in Fig. 6, by allocating 500kB of the memory as the RT FIFO and run the data logging loop under the normal priority level (technically lowest priority as possible) with a maximum writing speed, stable data logging is empowered. When logging to the Compact Flash is not smooth, FIFO memory acts as a buffer and stores unsaved data until the logging condition gets back.

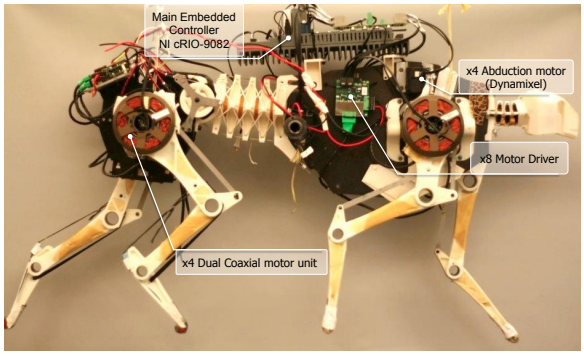


Fig. 10: MIT Cheetah. The robot has 8 BLDC motors, 4 smart motors (Dynamixel EX-106+). The size of the robot is 66 cm from front shoulder to back shoulder, and leg length when it is fully stretched is 60 cm for front legs, and 62 cm for rear legs, and the weight is 33 kg.

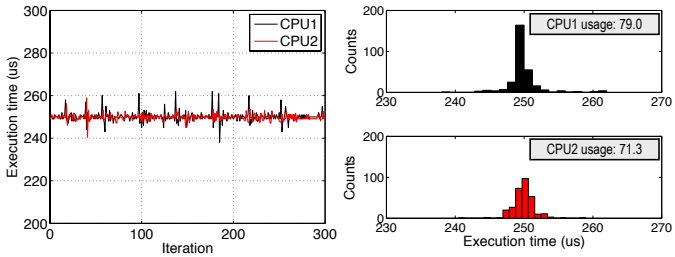
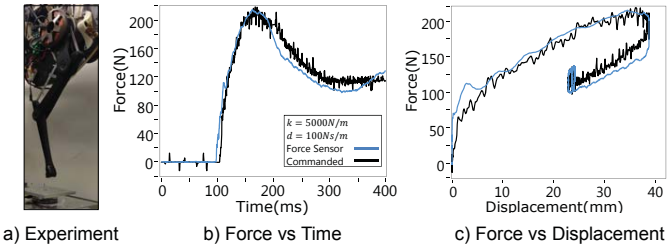


Fig. 11: Jitter in execution time at CPU1 and CPU2

E. Performance Test with the MIT Cheetah (Fig. 10)

1) Control system performance: The consistency of the execution times on both CPU1 and CPU2, one of system performance indices, is evaluated in Fig.11. Through the task separation and the sequence optimization, the proposed control platform with the dual core CPU and the FPGA targets 250 μs sampling rate. In 300 iterations of measurement data, the CPU1 with 79.0% CPU usage has an execution time with average $m_1 = 250.013$ μs and variance $\sigma_1 = 6.381$ $\sqrt{\mu s}$, and the CPU2 with 71.3% CPU usage shows an execution time with average $m_2 = 249.842$ μs and variance $\sigma_2 = 2.532$ $\sqrt{\mu s}$. Note that CPU usage lower than 80% is recommended [20]. Two histograms in Fig.11 reflects consistency of the sampling rate on both CPUs. As shown by the small values of each variance, the target sampling rate, $250\mu s$ is measured to be consistently achieved.

2) Leg impact test with the MIT Cheetah's leg: To test robust performance of the control system platform for the quadruped robot application, the leg-drop impact test was executed as shown in Fig.12. The multiple-legged robots inherently have external disturbance from a rough terrain and their controller are required to be robust to physical interaction with environments. When the impedance-controlled leg



a) Experiment b) Force vs Time c) Force vs Displacement

Fig. 12: The MIT Cheetah's Leg Impact Test with the control system platform; a) a photo of the experimental setup, b) commanded force and measured GRFs in the vertical direction with respect of time, c) commanded force and measured ground reaction force with respect to the leg's vertical displacement

is dropped on the force sensor, the impact is transferred to the actuator through the leg's structure with low mechanical impedance. This can lead to instability of the closed-loop control on the robot's leg, but the control platform with the 4 kHz sample rate was robust to the impact. Along with the system stability, high force proprioceptive impedance control [1] is successfully achieved.

VI. Conclusion and Future Work

The control system platform based on the single embedded controller that has multicore CPU and FPGA is successfully developed and tested. Because the FPGA layer's architecture follows the proven MapReduce technology, scalability of the system is guaranteed as far as the FPGA's space is allowed. Moreover, since the communication protocol between the control system and distributed devices can be designed as industrial standard communication buses, the system is certainly compatible.

The target robot platform, MIT Cheetah is operated by this control system platform. With this, a high-force proprioceptive control actuation system is successfully implemented [1] and high speed and highly efficient trot-running [2] is reliably executed.

Since the control architecture is applied only to the CompactRIO-9082, which has the best hardware specification among the commercial embedded systems, the architecture should be tested on the other single SoC solutions such as Xilinx Zynq.

ACKNOWLEDGMENT

This program is sponsored by the DARPA M3 program. We appreciate the hardware/software support of National Instruments.

References

[1] S. Seok, A. Wang, D. Otten, and S. Kim, "Actuator design for high force proprioceptive control in fast legged locomotion," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, October 2012.

[2] S. Seok, A. Wang, Chuah, M. Y. (Michael), D. Otten, J. Lang, and S. Kim, "Design principles for highly efficient quadrupeds and implementation on the mit cheetah robot," in *2013 IEEE International Conference on Robotics and Automation*, 2013.

[3] Intel, "The evolution of a revolution," 2007.

[4] Xilinx, "A generation ahead for smarter systems," 2013.

[5] Altera, "Altera's user-customizable arm-based soc," 2013.

[6] B. Gates, "A robot in every home," *Scientific American*, vol. 296, no. 1, pp. 58–65, 2007.

[7] Boston Dynamics, "Bigdog overview."

[8] M. Raibert, K. Blankespoor, G. Nelson, R. Playter *et al.*, "Bigdog, the rough-terrain quadruped robot," in *Proceedings of the 17th World Congress*, 2008, pp. 10 823–10 825.

[9] J. J. Rodriguez-Andina, M. J. Moure, and M. D. Valdes, "Features, design tools, and application domains of fpgas," *Industrial Electronics, IEEE Transactions on*, vol. 54, no. 4, pp. 1810–1823, 2007.

[10] E. Monmasson and M. N. Cirstea, "Fpga design methodology for industrial control systems - a review," *Industrial Electronics, IEEE Transactions on*, vol. 54, no. 4, pp. 1824–1842, 2007.

[11] E. Monmasson, L. Idkhajine, and M. W. Naouar, "Fpga-based controllers," *Industrial Electronics Magazine, IEEE*, vol. 5, no. 1, pp. 14–26, 2011.

[12] M. W. Naouar, E. Monmasson, A. A. Naassani, I. Slama-Belkhodja, and N. Patin, "Fpga-based current controllers for ac machine drivesa review," *Industrial Electronics, IEEE Transactions on*, vol. 54, no. 4, pp. 1907–1925, 2007.

[13] J. Huang, Z. Xie, H. Liu, K. Sun, Y. Liu, and Z. Jiang, "Dsp/fpga-based controller architecture for flexible joint robot with enhanced impedance performance," *Journal of Intelligent and Robotic Systems*, vol. 53, no. 3, pp. 247–261, 2008.

[14] H.-C. Lin, C. H. Chen, G.-S. Huang, Y.-C. Liu, and W.-C. Hsu, "Design of communication interface and control system for intelligent humanoid robot," *Computer Applications in Engineering Education*, vol. 20, no. 3, pp. 454–467, 2012.

[15] R. Wei, X. Gao, M.-H. Jin, Y. Liu, H. Liu, N. Seitz, R. Gruber, and G. Hirzinger, "Fpga based hardware architecture for hit/dlr hand," in *Intelligent Robots and Systems, 2005.(IROS 2005). 2005 IEEE/RSJ International Conference on*. IEEE, 2005, pp. 523–528.

[16] P. Khosla, "Choosing sampling rates for robot control," in *Robotics and Automation. Proceedings. 1987 IEEE International Conference on*, vol. 4. IEEE, 1987, pp. 169–174.

[17] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[18] R. J. Gove, K. Balmer, N. K. Ing-Simmons, and K. M. Guttag, "Multiprocessor reconfigurable in single instruction multiple data (simd) and multiple instruction multiple data (mimd) modes and method of operation," May 18 1993, uS Patent 5,212,777.

[19] National Instruments, "Multicore programming with labview technical resource guide."

[20] ——, "Best practices for target file io with labview real-time."