2020 IEEE/ASME International Conference on
Advanced Intelligent Mechatronics (AIM), Boston, USA
(Virtual Conference), July 6-9, 2020

Exploiting the SoC FPGA Capabilities in the Control Architecture of a Quadruped Robot

Chrysostomos Karakasis[1], *Student Member, IEEE*, Konstantinos Machairas[3,4], *Student Member, IEEE*,
Charalampos Marantos[2],*Student Member, IEEE*, Iosif S. Paraskevas[3],
Evangelos Papadopoulos[3], *Fellow, IEEE* and Dimitrios Soudris[2], *Member, IEEE*

***Abstract*— The use of FPGAs in the field of robotics has the potential to create interesting results considering the former's assets and the latter's necessities. Motion control is highly demanding both in resources and data, while the number of sensors employed in robotic devices can be large. On the other hand, FPGAs have gained a renowned place in the field of digital circuit implementation media. Simultaneously, the vast number of peripherals they are equipped with, renders them as ideal for the management of multiple external signals. In this paper, a highly affordable control architecture has been implemented for the quadruped robot Laelaps II based on a SoC FPGA, in order to examine the use of FPGAs in demanding robotic applications. The proposed system has real-time characteristics and is capable of controlling more than eight actuators at very high frequencies, outperforming most motion control platforms existing in the market, while keeping the total cost low. To achieve this, signals from all the robots peripherals are handled in parallel by the FPGA at a very high frequency, leaving the embedded ARM processor free to allocate all its resources to motion control and user interfacing tasks. The whole hardware/ software architecture is analyzed and evaluated compared with state-of-the-art approaches, while a trotting experiment with the Laelaps II motion system is presented for validation purposes.**

I. INTRODUCTION

Although quadruped robots have evolved drastically over the years, concurrently their requirements in motion control have become highly demanding, both in resources and data, whilst the number of sensors employed has been considerably increased. As a result, researchers seek new solutions, which can alleviate this computational burden, while offering programming flexibility and sufficient computing power.

[1]Chrysostomos Karakasis was with the School of Electrical and Computer Engineering, National Technical University of Athens (NTUA), Athens Greece. He is now with the Department of Mechanical Engineering, University of Delaware, Newark, DE 19716, USA chryskar@udel.edu

[2]Charalampos Marantos and Dimitrios Soudris are with the School of Electrical and Computer Engineering, National Technical University of Athens (NTUA), Iroon Polytechneiou 9, 15780 Athens, Greece hmarantos@microlab.ntua.gr, dsoudris@microlab.ntua.gr

[3]Konstantinos Machairas, Iosif S. Paraskevas and Evangelos Papadopoulos are with the School of Mechanical Engineering, National Technical University of Athens (NTUA), Iroon Polytechneiou 9, 15780 Athens, Greece. kmach@central.ntua.gr, isparas@mail.ntua.gr, egpapado@central.ntua.gr

[4]Konstantantinos Machairas' research has been funded by the "IKY Fellowships of Excellence for Postgraduate Studies in Greece - Siemens Programme in the framework of the Hellenic Republic - Siemens Settlement Agreement.

Considering the requirements of quadruped robots and the solutions up until now, there is a number of challenges and open issues in designing them. First of all, the physical dimensions and weight of the computing platforms are crucial. Furthermore, there is an increasing need for low cost, easily programmable and expandable solutions. Low power consumption of the computational system is also a key requirement that can significantly improve the power autonomy of the robot.

In the past decade, an ascending tendency of implementing Field-programmable Gate Array (FPGA) devices in robotic applications has been observed [1], as they offer parallel processing at a low cost, reconfigurability, while they have provision for a vast number of peripherals. Specifically, fields of interest include: parallel processing of multiple peripherals for multi-node control [2], acceleration of Simultaneous Localization and Mapping (SLAM) and visual odometry algorithms [3], [4], as well as motion control [5], [6].

Furthermore, similar approaches have been recorded on miniature quadruped robots. Chakravarthy and Xiao [7] developed a novel FPGA-based control system for miniature robots, where the FPGA is responsible for the generation of Pulse-width Modulation (PWM) output and the counting of quadrature encoder pulses, while an on-chip processor executes a Proportional-Integral-Derivative (PID) control on software. As a result, this is considered as a simplistic approach that uses only a few of the capabilities that FPGAs can offer. Similarly, Pan *et al.* [8] proposed a novel technical solution for underwater robots design, where a Xilinx Zynq-7000 All Programmable System on a Chip (SoC) was used to integrate motor control, sensor management, data acquisition and other functional units or circuits into a single chip for the whole robot, utilizing instantiations of IP cores. This solution targets again only miniature robots, while no detailed information about the control algorithm designed are given. On the other hand, notable implementations in conventional quadrupeds exist, which mainly were developed on FPGA-based devices by National Instruments, programmed in Labview. Chernyak *et al.* [9] describe a high mobility biomimetic quadrupedal robot, which utilizes an onboard SBRio FPGA for the management of all sensors (Inertial Measurement Unit (IMU), camera, LIDAR) and the motor positioning and velocity control. It should be mentioned here that an external netbook computer is utilized for user interfacing. Seok *et al.* [10] present a highly parallelized

control system platform, based on a multicore Central Processing Unit (CPU) and FPGA SoC device by NI, which maximizes compatibility, scalability and performance. In particular, the FPGA's parallel nature is exploited in the communication between the processor and all sensors and actuators, while the execution of the control algorithm and data logging are handled by the processor. While this work seems to be the most complete and the closest to the problem addressed herein, it has a number of limitations. First of all, the cost of the utilized device is very high. Furthermore, this approach implements the low-level functions (encoder reading, velocity estimation, and PWM signal generation) via the motor drivers. Additionally the FPGA is underutilized, while the ARM processor is used extensively by the control.

In this paper, a centralized control scheme for quadruped robots is introduced, using a low-cost SoC device that comprises an ARM-Cortex A9 processor and an FPGA of the Zynq-7000 Xilinx family, as part of the Zybo Development Board. Specifically, a Hardware/ Software Co-design architecture is presented, where the management of the motor control and encoder signals will be handled on the FPGA, while the higher level motion planning and control will be executed on the ARM processor. The paper will evaluate the capabilities of low-cost SoC devices in quadruped robots control, aiming to give new perspectives in the prototyping stage of robots, by offering powerful, configurable, easy to use, and energy efficient solutions of low mass and small size. The proposed scheme is experimentally validated using the quadruped robot Laelaps II (Fig.1)

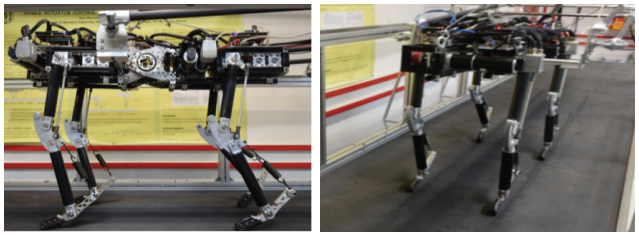

Fig. 1: The quadruped robot Laelaps II, constructed by the Legged Team at the Control Systems Lab of NTUA.

II. HARDWARE DESCRIPTION AND CONTROL ARCHITECTURE

A. Laelaps II

In order to examine the potential of using FPGA in robotic systems, the Laelaps II system has been used. Laelaps II is a quadruped robot with three-segmented legs constructed by the Legged Team at the Control Systems Lab of the Mechanical Engineering School of NTUA (Fig.1). Although the hip and knee joints are driven via actuators, a spring of high stiffness is located at each ankle joint, which practically reduces the number of segments to two, with lengths l_1 and l_2, respectively (Fig.2). Currently, the motion control scheme is based on elliptical toe trajectories. Specific parameters of the robot can be found in [11]. With the current architecture, a large number of boards are needed to control the motors, distribute signals, etc., while a large number of cabling runs throughout the robotic system to connect all necessary elements, thus increasing weight, power consumption and delays during communications.

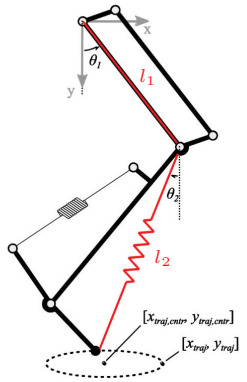

Fig. 2: Leg model [12].

B. Proposed Control System Architecture

The design goal is to develop a high performance and low-cost control system for Laelaps II that is scalable and reconfigurable. All sensors and actuators should communicate with the control platform directly via circuit boards that serve as mediators. The control platform will be equipped with several Inputs/Outputs (I/Os), as part of the Peripheral Module (Pmod) Interface defined by Digilent Inc. [13]. The general proposed architecture is depicted in Fig.3.

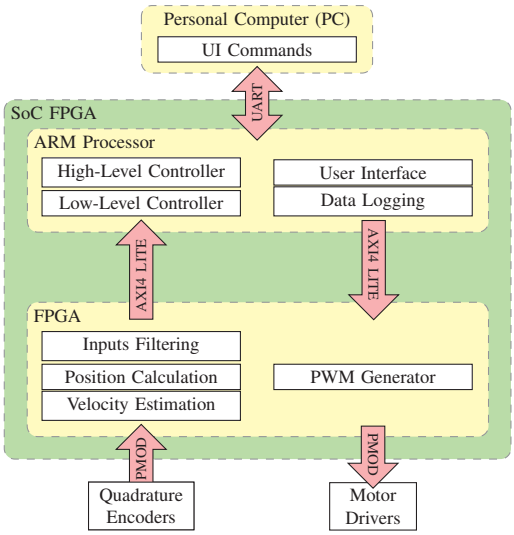

Fig. 3: Overview of the proposed control architecture.

C. FPGA component

In order to implement the proposed architecture, the Zynq 7000 AP SoC will be used. The main distinctive trait of Zynq is its amalgamation of traditional FPGA logic fabric, based on Xilinx 7-series FPGA architecture, with a dual-core ARM Cortex-A9 processor, capable of supporting operating systems such as Linux and Real-Time Operating

Systems (RTOSs), or Stand-alone programs that run directly on hardware (Bare Metal). It encapsulates two parts, (a) the Processing System (PS) revolved around the processor, and (b) the Programmable Logic (PL), which encloses the FPGA. In addition, the communication between the two units is achieved via the AXI4 Standard protocol, which is widely considered as the optimal interconnection technology for FPGA designs, since it ensures low latency and high bandwidth connections. As a result, hardware-software cooperation is attainable, where both the FPGA logic and processor can reach their full potential. Namely, the negative overhead of interfacing between two physically separate devices is avoided, while the overall cost and physical size drop substantially.

Apart from the PS and the PL, a variety of peripherals, an embedded memory and the previously mentioned AXI4 communication interface are featured in the Zynq architecture. Standard FPGA design approaches are: Hardware Description Language (HDL)-based and schemetic-based, while other appealing methods use High-Level Synthesis (HLS) tools, such as the Vivado HLS by Xilinx and National Instruments' Labview environment. For computationally demanding and intricate algorithms the HDL-based (Very High Speed Integrated Circuit Hardware Description Language (VHDL) or Verilog) approach was considered to be the most prevalent method [14].

III. IMPLEMENTATION

A. Approach

The FPGA (Hardware) layer consists of two functions. Initially, each joint's position is calculated via Quadrature Encoder Interface (QEI) blocks, while for precision and synchronization purposes, the incoming signals from the sensors are sampled, using a fast clock, in order to determine whether they are stable or not. Next, the filtered signals are supplied to QEI blocks realized in VHDL, which enable the calculation of each joint's position in encoder counts. The joint velocities are then estimated utilizing a custom period count based method, as the encountered rotational velocities are relatively low [15]. Once both position and velocity have been computed for each joint, they are transmitted to the ARM processor via the AXI4-Lite bus protocol. After the execution of the overall motion controller, control torques are generated for each leg's motors and forwarded back to the FPGA via the AXI4-Lite bus. Finally, appropriate PWM signals are produced to drive the leg motors via the motor drivers. The above procedure is completed in parallel for all joints, by dedicating a set of filtering, QEI, velocity estimation and PWM blocks for each of them. Moreover, the data are not serialized when transmitted through the AXI4-Lite Interface.

Although the ARM processor is equipped with two cores, currently only one is used for two distinct operations. Throughout an experiment, the CPU runs the User Interface (UI), while an Interrupt Service Routine (ISR) is responsible for the execution of the Active Compliance Controller (ACC) [11]. The motion planning and control algorithm, as well as the communication with the FPGA, are implemented in the low-level segment, while the high-level segment takes over the tuning of the planning and control parameters.

The user can insert commands and monitor the experiment's data from an external computer connected to the SoC FPGA via the UI. Specifically, a wired Universal Asynchronous Receiver-Transmitter (UART) serial communication is used with a high baud rate of 115200 bps.

Although this centralized control scheme was designed for the Zybo Zynq-7000 All Programmable SoC Trainer Board, it can be easily transferred unaltered to similar devices equipped with a Zynq-7000 SoC by Xilinx, while its principles could be applied in platforms by Altera and National Instruments (NI) as well.

The processing flow of the proposed architecture is illustrated in Fig.4. Namely, a Hardware/ Software Codesign was implemented that consists of two partitions: hardware and software parts, which are in charge of interfacing with the robot's sensors and actuators, and running the motion control algorithm respectively.

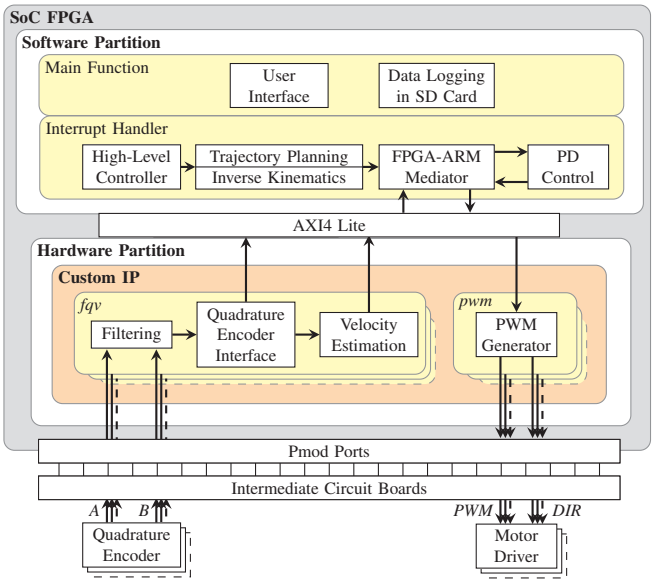

Fig. 4: Operating diagram.

B. Hardware Partition

Quadrature Encoder Interface Component: In order to track the position of each joint and thus the position of each toe, quadrature encoders were deployed, which are commonly used in a wide variety of applications from robotics to opto-mechanical mice. As in most encoders, two square waves are generated in quadrature, A and B, which indicate the moving direction of the respective encoder. According to those signals, an n-bit counter is either increased or decreased, where n is the minimum integer that satisfies $2^n \geq sr \cdot trr$, where sr is the encoder's standard resolution (counts per revolution) and trr is the total reduction ratio of the gearhead used with the motor. Currently, the counter's resolution is equal to 18 bits, however it is configurable, since it depends on the generic variables sr and trr. Generic

variables in VHDL provide static information to logic blocks similarly to constants in software, with the exception that their values can be defined outside their environment. For more clarification, one could think of them as passing arguments of a software function that are assigned with values when it is called.

Velocity Estimation Component: Due to the fact that the encountered rotational velocities are relatively low [15], a custom period count based function was created to calculate the velocity of both joints in each leg. Specifically, the system keeps track of the change in encoder counts and every time its value has changed by four counts, an event occurs and the velocity is calculated. In case of a direction change between those events velocity is set to zero. The approximation is shown on Eq.1, where X is the change in counts, while Δt is the number of clock cycles between two consecutive events.

$$v(k) = \frac{X}{t(k) - t(k-1)} = \frac{X}{\Delta t} = \frac{4\ counts}{\Delta t\ \ clock\ \ cycles} \quad (1)$$

Taking into consideration that $counts$ depend on the inverses of the total reduction ratio of the actuator and the standard resolution of the encoder, while $clock\ \ cycles$ are inversely proportional to the FPGA's operating frequency, the velocity can be expressed in rad/s via Eq.2, where $f_{FPGA-Hz}$ is the operating frequency of the FPGA in Hz.

$$u(k) = \frac{2\pi \cdot f_{FPGA-Hz}}{\Delta t \cdot sr \cdot trr} \ \ \frac{rad}{s} \quad (2)$$

Due to the fact that the VHDL code utilizes integer division, it was opted to multiply the dividend by $c_2 = 10^4$, instead of using a complex and demanding in resources float division, in order to maintain accuracy of four decimal places. Afterwards, the result is divided by c_2 in the ARM Processor, before the execution of the control algorithm. Furthermore, the velocity equation was expressed using generic variables that allow the designer to have parametric values during component instantiation. In conclusion, the velocity is estimated as the 32-bit integer quotient of the final division in Eq.2 that exploits the four generics c_2, trr, sr and $f_{FPGA-Hz}$.

PWM Component: A key factor in the motion control scheme is the creation of PWM signals of a specific frequency and duty cycle. By varying the duty cycle of the PWM signal, the torque of the motor is controlled, since a current control architecture is implemented in Laelaps II motor drivers. The PWM component was programmed in VHDL and like the aforementioned components, it utilizes generic variables for adaptability. Namely, the user can configure the FPGA's clock frequency (*sys_clk*), the desired PWM frequency (*pwm_freq*) and the resolution of the Duty Cycle in bits (*bits_res*). Currently, the PWM frequency is set to 20kHz, as specified in the employed motor drivers' datasheet (range 10-25 kHz), while 14 bits are dedicated for the Duty Cycle's resolution.

Custom AXI4 IP: The pith of the Hardware/ Software Codesign is the custom AXI4 IP that integrates the designed hardware along with the AXI4-Lite protocol and hence establishes the communication between the hardware and the software. A pair of *fqv* and *pwm* blocks is dedicated for each motor, where the *fqv* block conflates the filtering of the inputs with the QEI and Velocity Estimation components. Naturally, the IP also contains the same generic variables as its components and in fact it is responsible for defining their values. Once all position and velocity values have been computed, they are promptly transferred to the processor via the AXI4-Lite bus. Once the processor has concluded its procedure, the computed duty cycle values, along with the desirable Direction signals for each motor, are transferred to the corresponding *pwm* blocks, again via the AXI4-Lite bus. Among the three separate bus protocols of AXI4, the simplified memory mapped AXI4-Lite was selected as it was proved to be sufficient, whilst maintaining the benefit of a small logic footprint.

C. Software Partition

This section presents the software running on the ARM Cortex-A9 processor, responsible for the application of the control scheme, the interfacing with the user, as well as the retention of the necessary data for post processing. The corresponding source code was written in C language (Bare Metal) through the Xilinx Software Development Kit (XSDK) design environment, based on Eclipse 4.5.0 Integrated Development Environment (IDE).

Controller Implementation: The CPU is constantly occupied with the management of the UI, while the control framework is executed at a specific frequency as part of a timer ISR. An active compliance control framework for dynamic trotting was utilized, which includes a low and a high-level part [16]. The low-level segment is responsible for driving the legs along elliptical trajectories, while the high-level part is for the tuning of the low-level controller's parameters. Initially, the trajectory planning part produces elliptical primitives in each leg's workspace, according to which the inverse kinematics part calculates the desired joint angles. Consequently, the current joint angles and velocities are accessed from the AXI4-Lite interface, where the angles are converted to degrees and the velocities are restored from integer to floating-point format through division by $c_2 = 10^4$. Both angle and velocity values are then forwarded together with the desired angles to the joint-level active compliance part, which computes the appropriate duty cycles to drive each leg. Ultimately, the absolute values of the torque commands and their signs are segregated and sent to the FPGA, where corresponding PWM and Direction signals will be created.

IV. SYSTEM EVALUATION

A. Finalized Architecture

Intermediate Circuit Boards, as illustrated in Fig.4, were developed, aiming to fabricate a structured system without a tangled mass of wires. The first board's purpose is to firmly connect the tower with the SoC FPGA's board, whilst regulating the input signals voltage via a logic-level shifter.

On the other hand, secondary boards manage all connections between the SoC FPGA, the motor drivers and the encoders. In total, a centralized control tower was assembled (Fig.5), which was mounted on the torso of Laelaps II (Fig.6).

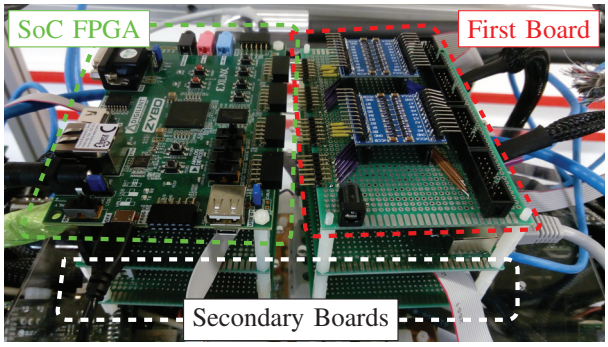

Fig. 5: Centralized Control Tower.

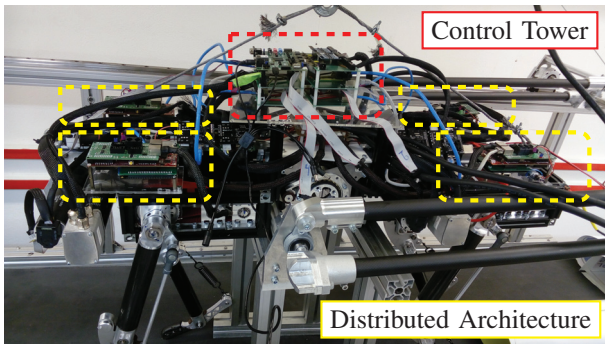

Fig. 6: Laelaps II unified with the Centralized Control Tower compared with the previous Distributed Architecture.

B. Hardware Resources Utilization

In this paragraph the FPGA resource utilization is analyzed. The main resources that were deployed, are the Lookup Tables (LUT) and I/Os. A small amount of the available Flip-Flops (FF) & Digital Signal Processors (DSPs) were also utilized. These resources can be used for new tasks, useful for the overall functionality of Laelaps.

The resources utilization results are illustrated in Figure 7, where the hardware designs for one, two and four legs are compared. As observed, the resources utilization is proportional to the number of legs, proving the scalability of the developed system; for each new leg, identical hardware blocks are added in parallel, showing the ease-of-use of the proposed framework. On the contrary, in CPU architectures, like in the previous Laelaps versions, increasing the number of legs would add more instructions and hardware interrupts in the serial processing loop. As a result, the CPU's performance would be heavily degraded, considering that a CPU, opposed to the parallel nature of the FPGA, is limited to the number of instructions it can process at a time, depending on the data dependencies, multithreading and pipelining.

The utilized platform is one of the cheapest SoC FPGA devices in the market that offers rather limited resources and it is utilized mainly for educational purposes. However, it was

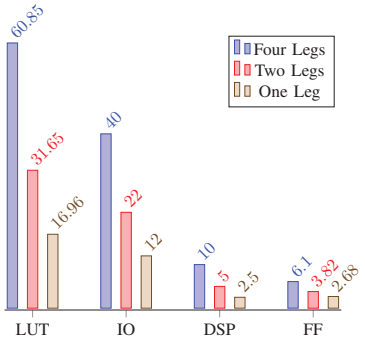

Fig. 7: Comparison of Resources Utilization (%).

possible to incorporate the whole control scheme without the system being stretched (60% of LUTs and 1 of the total 2 CPUs are used) in this platform; more advanced devices are recommended for the exploitation of the full potentials of FPGAs in even more complex control schemes.

C. Performance Evaluation on Laelaps II

The validity of the proposed system was examined and verified through several experiments. In fact, all eight joints of the robot were successfully controlled and performed a trotting pattern in sync. A supplementary video file of one of the conducted experiments can be found in [17]. In these experiments, the processor's running frequency was 650MHz, while the FPGA was operating at 83MHz.

Figure 8 presents the trajectory of the front left toe during the steady state. As it can be seen, the desired trajectories are closely followed with small errors. By increasing the control gains, the errors get smaller (trajectory is closely tracked) but also a less compliant behaviour is expected for the robot [16]. Tuning the system compliance via the control gains is critical in legged locomotion and not addressed in this work. The presented experiment is considered successful, since the results are identical to the outputs of the existing control system of Laelaps, which allows for its nominal operation as presented in [18].

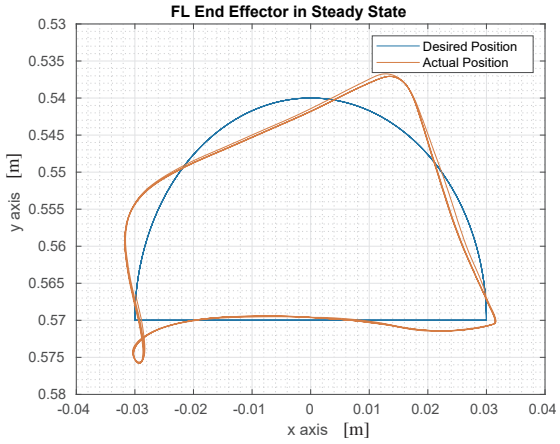

Fig. 8: Response of the Front Left Toe.

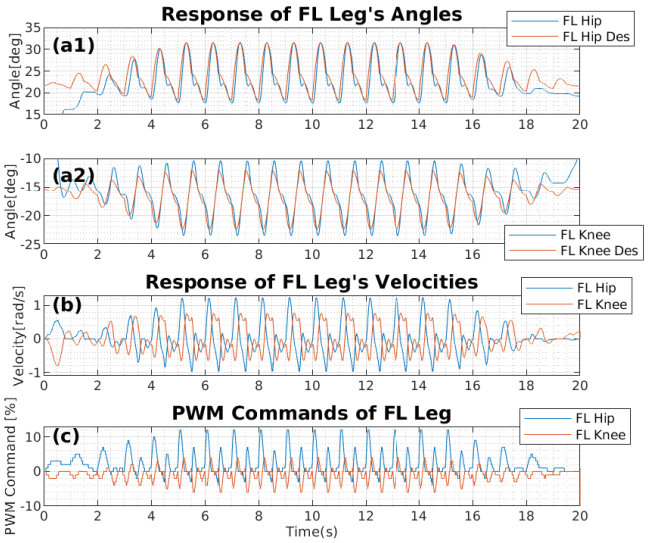

Fig. 9: (a1) The FL Hip's angle, (a2) the FL Knee's position, (b) The FL Leg's rotational velocities, (c) the PWM Commands sent to the motors of the FL Leg.

Figure 9 illustrates additional joint level experimental data for the front left leg. Similarly to the trajectory results, the desired values are closely tracked, while small errors exist. Additionally, as observed during the first 5 seconds of the experiment the ellipse semi-major and semi-minor axes are gradually increased from 0 to the desired values and vice versa for the last 5 seconds to achieve smooth start/stop operation.

V. Comparative Analysis

A. Comparison with Laelaps II Control Architectures

Laelaps II with Centralized Control: In the early version of the robot, a centralized architecture was used, featuring a high-cost, hard-to-extend, bulky control tower (PCIe/104 CMA34CRD1700HR-4096) with four control cards to handle all communications [18]. Off-the-shelf architectures of this form factor would not allow for solutions of lower cost, size and mass, capable of handling this high number of Laelaps peripherals (e.g. reading more than 10 encoders). Lastly, a non-real-time Linux OS was used that caused several synchronization issues. We note that this fact led to the real-time Bare Metal approach followed in the proposed SoC FPGA scheme. Experimenting with this early version, quickly showed that such robotic systems can largely benefit from less expensive, more expandable and easier to use and to develop solutions. The proposed SoC FPGA system provides solutions to all the problematic issues of this first PCIe 104-based architecture, while reducing the cost to 2%.

Laelaps II with Decentralized Control: In contrast with the above version, the most recent version of Laelaps' control system consists of one EtherCAT Master node and four EtherCAT slave nodes (one node per leg) [12]. One dual-core 32-bit 200 MHz microcontroller TMS320F28379D is used at each slave node to take over the communication tasks (EtherCAT), handle all the signals (incremental encoders, PWMs etc.), and execute the control algorithm for each leg. Microcontroller Units (MCUs) of this type typically support less than three QEIs, imposing an important limitation in an architecture of this complexity; four MCUs had to be used to read all the encoders of the robot. Apart from several advantages, this distributed architecture has several disadvantages when compared to the SoC FPGA approach. First, the deployment and maintenance of the EtherCAT network comes with significant development time. Second, the size of the whole system can be much larger than the FPGA-based system. The total cost and the number of cables can be also significantly higher. Last, to exploit the real-time characteristics of the EtherCAT network, significant time is required to program a Linux real-time OS as the master computer of the robot [19]. In contrast, the Proposed Model offers a powerful and highly affordable (20% of the distributed's cost) signal handling device capable of supporting more than 8 encoders and actuators using a single chip with an embedded ARM Cortex-A9 processor, similar to which hardly any devices exist in the robotics market.

B. Comparison with State-of-the-Art Relevant Techniques

From the related approaches described in the Introduction section, the most interesting is [10]. A comparison was considered very useful in order to highlight the advantages of the proposed design.

Table I presents some typical comparison points which are critical for almost any robot design, having as reference the results of [10] ("Base Design"). First of all, the proposed design is based on a ZYBO Zynq 7000 SoC which *costs* 30 to 50 times less than the cRIO-9082 of the Base Design. Furthermore, the position and velocity estimation, as well as the PWM Signals Calculation parts, are performed by motor drivers in the Base Design, while in the proposed approach all these critical parts are implemented in the FPGA, offering flexibility and the capability of adding more motors or making changes according to the needs without being restricted to the current infrastructure. Moreover, in the Base Design the FPGA collects information from Drivers, serializes and unifies them into one single array and vice versa for the current array. The proposed framework uses the FPGA to calculate and forward in parallel the position and velocity signals for each motor to the ARM Processor and vice versa for the PWM signals, reducing the communication overhead and offering a more compact solution, thus better exploiting the capabilities that FPGAs offer.

The next difference concerns the communication between FPGA and CPU. In the Base Design the array with all necessary signals is transferred to the first CPU via a Direct Memory Access (DMA) Channel, while in the proposed approach all signals are transferred in parallel to the ARM Processor via the AXI4-Lite bus protocol. Due to the fact that a relatively small amount of data is transferred at each step, AXI4-Lite is considered preferable as it offers faster communication. Regarding the control frequency, the Base Design architecture achieves a sampling rate of 250μs and hence enables a control update frequency of 4kHz. In the Proposed Model, an indicative control frequency of

TABLE I: Comparison between proposed approach and Base Design [10].

	Base Design [10]	**Proposed**
Cost	$6.000-11.500	$200
Position, Velocity, PWM Signals Calculation	Motor Drivers	FPGA
FPGA Utilization	FPGA collects information from Drivers	FPGA calculates position, velocity and PWM
Communication between FPGA and CPU	DMA Channel	AXI4-Lite bus protocol
ARM Processor Utilization	Utilizes both CPUs	Utilizes only one CPU
Control Loop Frequency	4kHz	Supports frequencies higher than 4kHz (currently 1kHz)

1kHz was selected for the experiments, despite the fact that frequencies higher than 4kHz can be assigned by the user. Namely, the maximum possible frequency is limited only by the update frequency of the sensors and actuators.

Finally, the Base Design utilizes both CPUs of the chip. The first one serves as mediator between the FPGA, while the second one runs the control algorithm, which implements a PID loop and generates new torque commands for the motors. In the proposed implementation only one CPU is utilized, which is responsible both for the communication with the FPGA and the implementation of a PD controller, generating new torque commands for the motors. This approach keeps the second CPU free in order to be used for other high level processes and operations that can be used by the robot (e.g. camera drivers, and image processing).

Summarizing, with the proposed approach, the FPGA is being used more substantially and is being exploited to a larger extent, proving that its use in robotic systems can be very beneficial and for this reason it is highly recommended for robotics applications.

VI. CONCLUSION

In this work an optimal use of FPGAs in robotic applications was presented. The extensive capabilities of the former seem to allow the latter of having extensive capabilities with the minimum footprint in terms of mass, volume and cost, increasing at the same time the communication speed. In order to examine this, a control architecture using a SoC FPGA has been implemented for a quadruped robot. Various building blocks have been implemented and presented, while the performance of the system has been tested with successful results. Consequently, a cooperation between the two fields seems reasonable, and hence the exploitation of FPGAs in robotic applications is highly recommended. Future work in order to further exploit this combination is underway.

ACKNOWLEDGMENT

The authors wish to thank Konstantinos Koutsoukis, Athanasios Mastrogeorgiou, John Valvis and George Bolanakis for the design and development of the quadruped robot Laelaps. This research work was partially supported by the EU funded project H2020 SDK4ED (http://www.sdk4ed.eu).

REFERENCES

[1] X. Shi, L. Cao, D. Wang, L. Liu, G. You, S. Liu, and C. Wang, "Hero: Accelerating autonomous robotic tasks with fpga," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 7766–7772.

[2] M. Li, X. Duan, H. Li, T. Cui, L. Gao, Y. Zhan, and Y. Xu, "Control and experimental validation of robot-assisted automatic measurement system for multi-stud tensioning machine (mstm)," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2016, pp. 2816–2821.

[3] J. Nikolic, J. Rehder, M. Burri, P. Gohl, S. Leutenegger, P. T. Furgale, and R. Siegwart, "A synchronized visual-inertial sensor system with fpga pre-processing for accurate real-time slam," in *2014 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2014, pp. 431–437.

[4] G. Lentaris, I. Stamoulias, D. Soudris, and M. Lourakis, "Hw/sw codesign and fpga acceleration of visual odometry algorithms for rover navigation on mars," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 26, no. 8, pp. 1563–1577, 2015.

[5] J. C. Linares, A. Barrientos, and E. M. Márquez, "Hybrid bio-inspired architecture for walking robots through central pattern generators using open source fpgas," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 7071–7076.

[6] A. Majeed, S. B. Ozturk, and D. K. Tureli, "An encoder fault tolerant fpga based robot control using bluetooth of a smart phone," in *2017 10th International Conference on Electrical and Electronics Engineering (ELECO)*. IEEE, 2017, pp. 1336–1341.

[7] N. Chakravarthy and J. Xiao, "Fpga-based control system for miniature robots," in *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2006, pp. 3399–3404.

[8] S. Pan, S. Guo, L. Shi, Y. He, Z. Wang, and Q. Huang, "A spherical robot based on all programmable soc and 3-d printing," in *2014 IEEE International Conference on Mechatronics and Automation*. IEEE, 2014, pp. 150–155.

[9] V. Chernyak, T. Flynn, J. O'Rourke, J. Morgan, A. Zalutsky, S. Chernova, S. S. Nestinger, and T. Padir, "The design and realization of a high mobility biomimetic quadrupedal robot," in *Proceedings of 2012 IEEE/ASME 8th IEEE/ASME International Conference on Mechatronic and Embedded Systems and Applications*. IEEE, 2012, pp. 93–98.

[10] S. Seok, D. J. Hyun, S. Park, D. Otten, and S. Kim, "A highly parallelized control system platform architecture using multicore cpu and fpga for multi-dof robots," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2014, pp. 5414–5419.

[11] K. Machairas and E. Papadopoulos, "An analytical study on trotting at constant velocity and height," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 3279–3284.

[12] S. Athiniotis, "Firmware design for microcontrollers on EtherCAT network for quadruped robot motion control," http://dx.doi.org/10.26240/heal.ntua.15750, 2018.

[13] Digilent Inc., "ZYBO^{TM}FPGA Board Reference Manual," 2014, revised February 27, 2017.

[14] S. M. Qasim, A. A. Telba, and A. Y. AlMazroo, "Fpga design and implementation of matrix multiplier architectures for image and signal processing applications," *International Journal of Computer Science and Network Security*, vol. 10, no. 2, pp. 168–176, 2010.

[15] W.-H. Zhu, "Fpga-based adaptive friction compensation for precision control of harmonic drivers," in *2010 IEEE International Conference on Robotics and Automation*. IEEE, 2010, pp. 4657–4662.

[16] K. Machairas and E. Papadopoulos, "An active compliance controller for quadruped trotting," in *2016 24th Mediterranean Conference on Control and Automation (MED)*. IEEE, 2016, pp. 743–748.

[17] https://www.youtube.com/watch?v=7Io6hpUmc00.

[18] https://nereus.mech.ntua.gr/laelaps/.

[19] M. Karamousadakis, "Real-time programming of EtherCAT master in ROS for a quadruped robot," http://artemis.cslab.ece.ntua.gr:8080/jspui/handle/123456789/17313, 2019.