2021 IEEE International Conference on Electronics, Computing and Communication Technologies (CONECCT) | 978-1-6654-2849-1/21/$31.00 ©2021 IEEE | DOI: 10.1109/CONECCT52877.2021.9622538

A Fully Integrated, High Speed, Motion Controller Hardware Accelerator for Autonomous Systems

Ankit Gupta
Intel Labs
Intel Technology
Bangalore, India
ankit.gupta@intel.com

Saksham Soni
Intel Labs
Intel Technology
Bangalore, India
saksham.soni@intel.com

Dibyendu Ghosh
Intel Labs
Intel Technology
Bangalore, India
dibyendu.ghosh@intel.com

Sriram K. Muthukumar
Intel Labs
Intel Technology
Bangalore, India
sriram.k.muthukumar@intel.com

Karthik Narayanan
Intel Labs
Intel Technology
Bangalore, India
karthik1.narayanan@intel.com

***Abstract*—Efficient and accurate motion control for any autonomous system (such as, aerial vehicles, ground robots) is a major challenge for robotics researchers and enthusiasts. Fast moving systems demand accurate control of motion for robust operation. To achieve this, high loop rate with fixed minimum jitter is required. As the size of these robots shrinks, they become more agile and are capable of performing very aggressive maneuvers. For such operations, the overall control loop timing requirements become critical. In state-of-the-art approaches, the problem of motion control is solved using a dedicated microcontroller with a specialized software stack. However, in this approach, the control loop suffers from jitter. Also, this approach fails for a multi-axis system (with increased number of axis in the system). Further, this approach increases the hardware component on the platform which adds to the total cost and weight. These challenging requirements demand a dedicated hardware accelerator for efficient operation. To this end, we devise an architecture and data flow for the design of motion controller hardware accelerator for the autonomous system use cases. This accelerator implements the complete control loop in hardware and allows multiple controllers (hence multiple actuators or motors) to run in parallel. Motion controller (MC) hardware accelerator was implemented and integrated on a test System-on-Chip (SoC) and the performance was fully characterized. Further, we have demonstrated the hardware accelerator operation on a physical mobile autonomous robotic system (ground and aerial vehicle). The experimental results demonstrate efficient control operation and our solution achieves ~44x faster response time and ~1000x improvement in overall power consumption compared to the software based solution.**

***Index Terms*—Motion Controller, Hardware Accelerator, Mobile Robot**

I. Introduction

Motion control is an integral aspect of any robotics system which insures overall stability and robustness of the system. Every autonomous system has a specific kinematic configuration, for example surface systems can be holonomic or non-holonomic type. Similarly, for aerial vehicle, it could be fixed wings or multi rotor type. Feedback control subsystem plays a key role in manipulating these mechanism for accurate navigation of autonomous system in any environment. Autonomous systems have multiple tasks which requires real time processing viz. Perception, Localization, Cognition and Motion Control etc. The response time (in Hz) requirement for any task is governed by system specification. In general, the Motion Control subsystem needs to respond approximately ~10x faster [1] than other tasks to navigate reliably without damaging the system and the surroundings. To develop a robust autonomous system, its very important to have hard real time response for the controller sub-system and isolation from non-motion system services. Traditionally, feedback controller sub-system design requirements are solved by using a dedicated microcontroller [2], [3]. These solutions suffers from jitter. Also, this approach fails for a multi-axis system (with increased number of axis in the system). Jitter increases with increase in number of axis. Processing capabilities and communication bandwidth requirements between the controller and application processor limits the complexity of control algorithms and the number of actuators that can be controlled efficiently. Stringent turnaround time which is required for optimal system response is limited by processing capability. Having multiple processors in the system complicates the overall firmware design as well. Some of the major disadvantage of above solutions are

- Data communication interface between microcontroller and main SoC limits the performance
- Sequential execution limits the number of actuators that can be controlled simultaneously
- Extra hardware in form of microcontroller and larger control loop time due to multiple actuators

More recent alternative to software based solution is to implement the controller in Field Programmable Gate Array (FPGA). A number of authors have considered the imple-

mentation of Proportional-Integral-Derivative (PID) controller using FPGA [4]–[8]. Parallel PID controllers with shared hardware resources have been proposed [7], [9]. Design methodology to select the correct computation data unit width is proposed in [5]. Most of the works talk about acceleration of only the PID control equation in FPGA. This is only one of the building blocks of Motion Controllers(MC). In these implementations rest of the preprocessing and post processing involved are taken care using software routines. Such type of Hardware/Software partitioning is disadvantageous for smaller, more agile systems due to higher overhead required to handle data movement. To build a robust autonomous system, an efficient accelerator requires tighter integration of many components (data acquisition peripherals, preprocessing and post processing elements, controller etc.).

To this end, this paper proposes a scalable Motion Controller Hardware flow and reference sub-system architecture. This addresses the above-mentioned limitations of the current solutions and enables control of complex robotic configurations at higher loop rates, using small lightweight and highly integrated platform. This design can be ported on to FPGA or integrated in a System-on-Chip (SoC). Our main contributions are

1) Highly configurable hardware architecture catering to hard real time requirements of different locomotion mechanisms.
2) Built in fail safe feature to avoid any accidental damage due to CPU/system latencies.
3) Hardware based sensor fusion scheme to improve controller response.

The remainder of the paper is organized as below. General pipeline for any autonomous system is presented in Section II-A and feedback control system overview is presented in Section II-B. The control data flow for the hardware is provided in Section II-C. Detailed architecture description and configuration options are presented in Section III and FGPA testing on ground bot and SoC results on a drone are presented in Section IV.

II. Preliminaries

A. Autonomous System Pipeline

Data flow for any typical robotic system consists of various steps which demands real time processing. The whole process starts with environment perception followed by specific computation to generate a target location [1]. Motion trajectory is computed based on this information which is consumed by motion controller block to navigate the system. Feedback from actuators helps motion control block to have a closed loop control over bot movement. Localization block detects the current pose of the bot and gives that information to path planner to generate new set points for the bot to move.

B. Feedback Control System

Typical block diagram of a Single Input Single Output feedback control system is shown in Fig 1. Here plant/actuator is the process that we want to control. Objective of these control systems is to maintain the desired set point value for the plant.

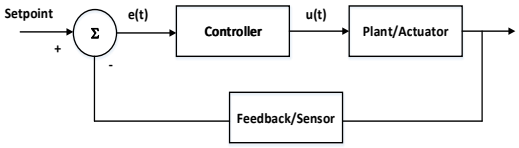


Fig. 1: Block Diagram of a Feedback Control system

This value is given to the control system and based on the feedback (sensor data) received from the plant/actuator controller generates an output, which is in turn fed to the plant. The controller may have different structures and design methodology to achieve desired performance. It can implement various new control algorithms like Fuzzy control, Adaptive control, Sliding window control and Model Predictive Control etc [9]. But Proportional-Integral-Derivative (PID) type controller is the most popular due to its simplicity and robustness. It has three correction terms the Proportional mode, Integral mode, Derivative mode. These three modes act on the error value at any point in the system. Overall corrective action of a PID controller can be written as

$$u(t) = K_p * e(t) + K_i \int_0^t e(t)dt + K_d \frac{de(t)}{dt} \quad (1)$$

Where K_p, K_i and K_d are the proportional , integral and derivative gains for PID controller respectively. $u(t)$ is the controller output and $e(t)$ is error term. Above equation is in continuous time which can be written in discrete time difference equation as

$$u_n = K_p * e_n + K_i * T_s * e_{sum} + \frac{K_d}{T_s} * (e_n - e_{n-1}) \quad (2)$$

$$e_{sum} = e_{sum} + e_n \quad (3)$$

Where T_s is the time step. It is desired to have fixed time step value for the stable operation.

C. Hardware Control Flow

Hardware control flow for the motion accelerator is shown in Figure 2. This can be scaled to fit in different use cases and configuration of ground/aerial robots. In order to support all the required flexibility and programmability of the parameters in the design, this block has a configuration and status register file (CSR). The CSR is accessible from SoC's CPU to enable run time configuration. These CSR will be referenced in the following flow description.

In the above flow, events in time proceeds as we go from top to bottom. Flow description is as follows

- All the blocks have dedicated CSRs which can be configured to update the parameters. These blocks can be enabled/disabled individually or can be configured to work in close loop flow. Once the initialization is done, all the blocks wait for global enable.

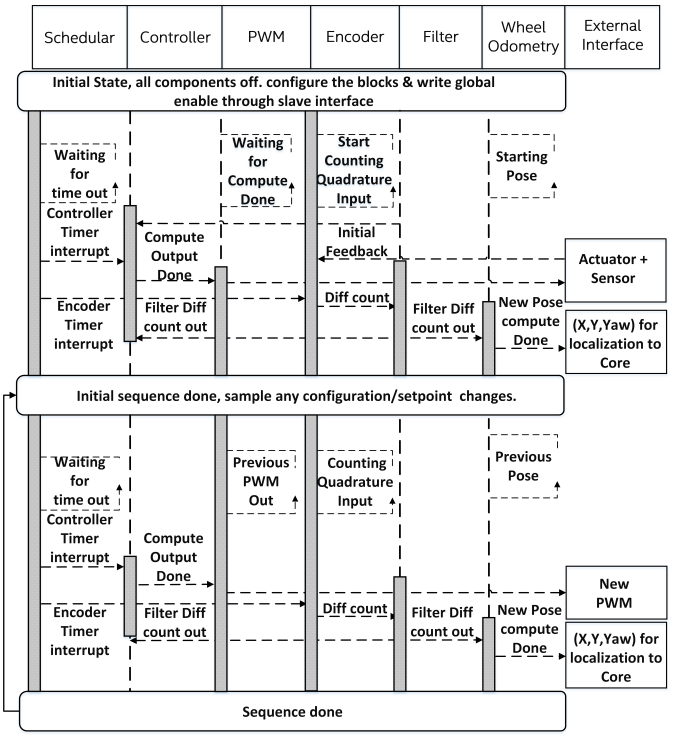


Fig. 2: Hardware flow of motion controller accelerator

- Scheduler waits for the time out to occur. All other blocks wait for the interrupt from the scheduler.
- As soon as the timer interrupt comes for the first time, controller computes the PWM pulse width based on the setpoint and the initial feedback and PWM block consumes the same.
- Sensor Feedback module (Encoder) starts counting the quadrature input and keeps track of the number of edges received. Once the encoder scheduler interrupt comes, difference of count between the previous count and the new count is generated and is consumed by the filter and the output of which acts as an input for the controller and the 2D-Pose block.
- Pose block then generates the new position of the bot using input from filter and the previous location (i.e. x, y, and yaw). This localization data is then consumed by the core for other tasks. This pose data is useful for ground bots. Aerial vehicles/Drones require a more complex 6D-Pose estimation block which requires data from other sensors as well.
- Once this initial sequence is done, controller computes the new command every time its scheduler timer expires. PWM changes the pulse width according to the latest command output from controller.
- Sensor Feedback module keeps on counting continuously and provides the difference count every time it receives the new interrupt and filter captures the same and provide that to controller and Pose block.
- SoC CPU can change the setpoint and other configuration value any time during the operation, but controller reacts to these changes only after the new scheduler timer interrupt.
- System keeps on running until it receives an interrupt from Watch Dog Timer (WDT) or is being stopped by the core. Data bypassing from above mentioned flow is done in the architecture to generate cascaded controllers.

III. Design Architecture

Block diagram for the Motion Controller accelerator based on the above mentioned flow is shown in the Fig 3. This architecture can be used for both mobile ground and aerial robot. This block can be integrated with system bus on SoC or on a FPGA along with any tiny CPU. All the configurations can be done using the slave interface. Once configured it can run autonomously with a provision of interrupt in case of any specific exception conditions.

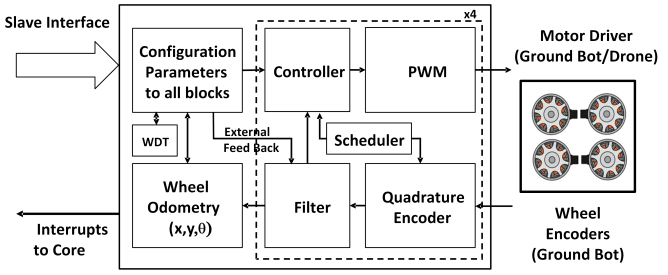


Fig. 3: Block diagram for Motion Controller accelerator

Controller Block : This block Fig 4, is the architecture of PID difference equation 2 and 3. Block has configurable tuning parameters and flexible scheduler timing. This controller can be configured to run in auto or manual mode. In auto mode, controller takes in the feedback from the sensor feedback block (encoder)/filter and adjust the controller output according to the desired set point. In manual mode, controller takes in the feedback but does not change the output. Rather it tracks the feedback coming from encoder and adjust the control output only when configured into auto mode. This will avoid sudden spike in the output.

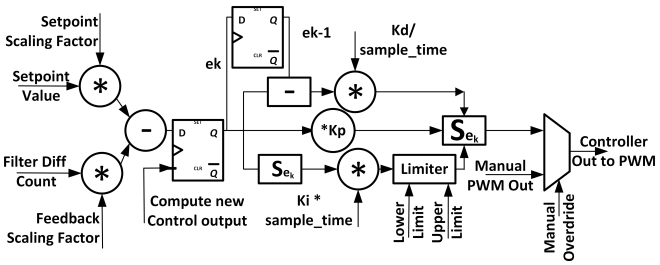


Fig. 4: Architecture diagram for PID block

PWM Block : PWM is integral to speed/position control in any DC, BLDC or Servo motors. This module is responsible for generating the pulses of specific width computed by the controller. Period and width of the pulses can be updated anytime but changes are reflected only after current PWM pulse is

complete. This module can be configured in either continuous mode or one-shot mode (Only one pulse is generated by the block).

Scheduler Block : There are individual scheduler timers in the system for Controller and Encoder. Using scheduler, we can configure overall control loop frame per sec. Encoder, Filter (when enabled) and 2D Pose (when enable) runs on the same scheduler timer. Only one-time configuration is needed for scheduler to run continuously.

Sensor Feedback/Encoder Block : Architecture of the block is shown in Fig 5. Rotational feedback from any motor is generally received in form of square pulses, the frequency of which is proportional to speed. Counting the number of pulses in a given duration will indicate the speed of motor/actuators. These pulses can be generated from variety of sensors such as rotary encoder, Hall Effect sensors etc. There could be a single pulse, or a pair 90 phase shifted with respect to each other. Since pulses are generated from mechanical structure, they are very noisy. These pulses when given to encoder block are first passed through a fully configurable debounce logic (glitch filter) to detect rising and falling edge separately. The system can determine direction of rotation by identifying the leading phase pulse. Encoder block can operate in three modes i.e. Quadrature (all four rising and falling edges), single rising, single rising and falling edge count. Triggering in quadrature mode will increase the resolution up to four times as compared to single rising edge. Upon receiving the scheduler interrupt, encoder will send incremental count difference.

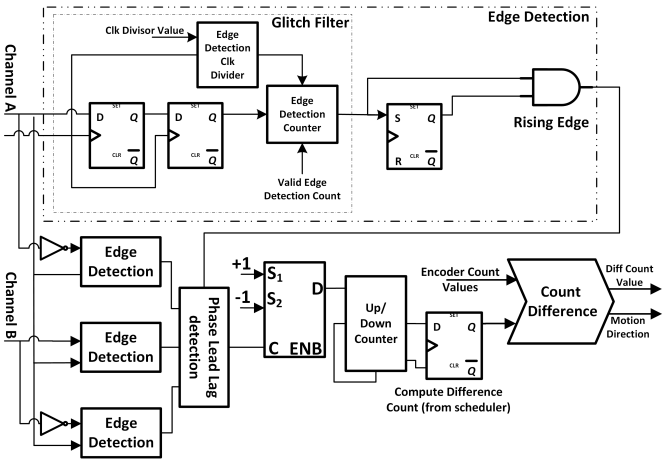


Fig. 5: Architecture diagram for Encoder block

Filter Block : This block Fig 6, is a moving average filter with a configurable window size to provide best estimate of the feedback incremental encoder count. There is an option of taking input from encoder or through configuration register which can be written by the core. Scheduler configures the sample duration and if the input is taken from the encoder, then both encoder and filter run at same frequency.

2D Pose Block : This is a dead reckoning based localization block to determine the current position (i.e. x, y, yaw) of the ground mobile robot. It takes bot kinematic structure and

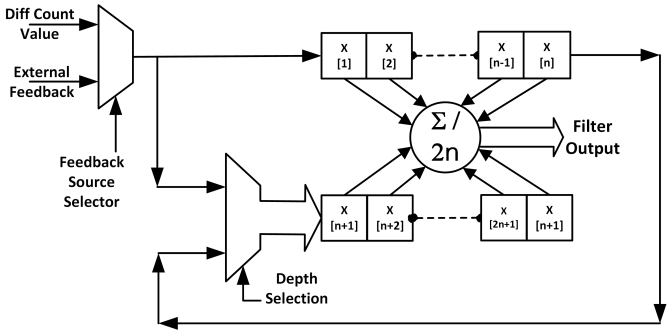


Fig. 6: Architecture diagram for Filter block

different physical dimension as info and calculates current position using filter/bypass filter data with an option to fuse with inertial yaw. System CPU can access this data and fuse with the information received from other sensors (Visual Odometry, LiDAR etc) to get better estimate of current pose. Architecture for the block is shown in Fig 7.

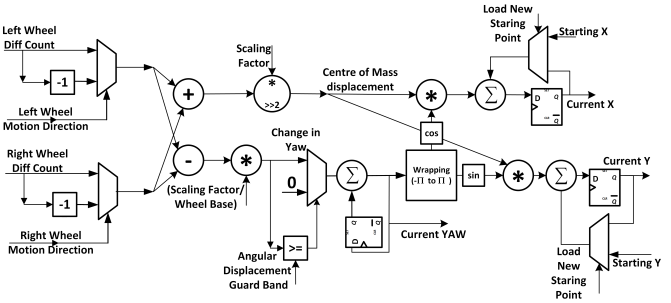


Fig. 7: Architecture diagram for 2D Pose block

Watch Dog Timer(WDT) Block : This is the built in fail safe mechanism included in the design. WDT has same specs as Scheduler timer. It is used to make sure that bot doesnt overrun with the given set point value. Once a new setpoint is configured, WDT is enabled. Different alert level can be configured to inform core about the end of this duration. Expectation is core will come and update setpoint or re-enable the WDT.

IV. Hardware Implementation & Results

The proposed architecture was implemented in SystemC and was taken through High Level Synthesis flow to generate the Verilog code. To minimize the resources in hardware design we followed the guidelines mentioned in [10]. We preferred direct computation instead of look-up tables at places to save on chip memory. Various pipelined architectures are explored to get the sweet spot for performance and area Fig 8. Fixed point implementations are done over Floating Point to save the logic area and still retain the desired accuracy. 44 Bit fixed point representation (24 bit after radix point) was selected to implement the internal stages of the controller. Extensive pre-silicon validation was done using Universal Verification Methodology (UVM).

To test the design on physical system, it was integrated with a small generic CPU core, memory subsystem and some basic peripherals for communication and debugging. This helped to program the accelerator and run it autonomously. Our design is validated in two different platforms.

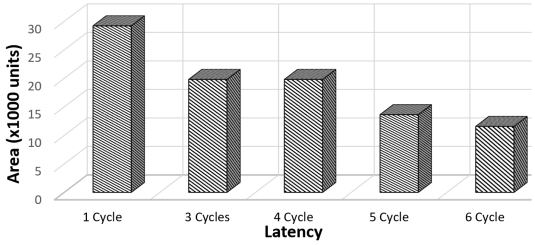


Fig. 8: Controller area variation vs Latency (number of pipeline stages) plot. Area gain is not very significant once we go beyond 3 pipeline stages.

FPGA based validation In this we ported the Verilog code of the complete integrated system on Intel® Stratix®-IV board. The design was synthesized to run the controller block input clock at 2.048MHz clock. This frequency is selected as this number provides easy division for 10 Bit PWM. Board I/Os were connected a 12V High precision DC motor with quad encoder, 512 Counts per Revolution, 9200RPM through a DC motor driver. We programmed the control loop to run at 500Hz. Setup is shown in Fig 9.

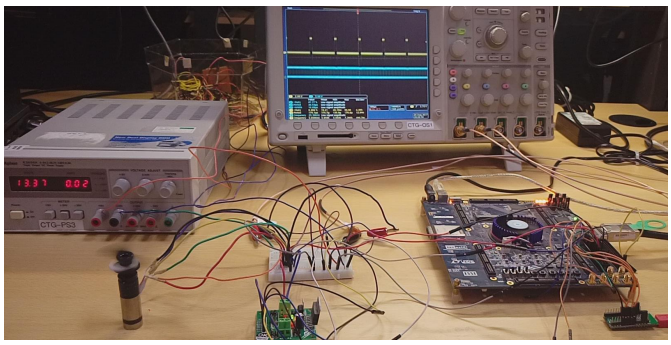
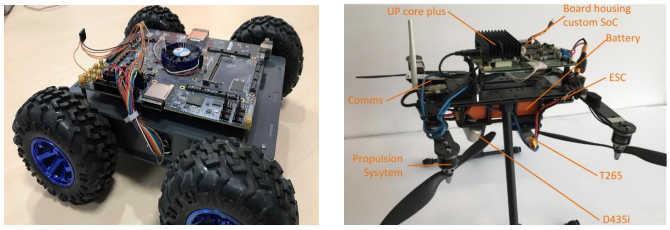


Fig. 9: FPGA testing setup for the system with control loop maintaining the motor speed at higher voltage 13.37v, CRO displays the injected PWM

Motor Controller was programmed in Speed control mode with DC motor running at fixed speed. Control loop was tested by loading the motor and varying the applied voltage. After tuning the parameter we found the control loop running very smoothly and observed less than 3% jitter in loop time. After the testing, FPGA setup was mounted on an off the shelf four wheeled robot chassis. Weight of the complete system is ~3Kg. It was fitted with high precision DC motors Fig 10a.

Comparison results while running the bot on ground are shown in the Fig 11. To test robustness, robot was made to climb 4m on a slope of 45 degree with a fixed small speed of 20cm/sec with once control loop disabled and enabled the



(a) (b)

Fig. 10: (a) Ground bot with FPGA (b) Drone with the custom SoC and other components

second time. Robot was able to climb the slope with constant speed in case of controller while it failed to climb without motion controller being enabled.

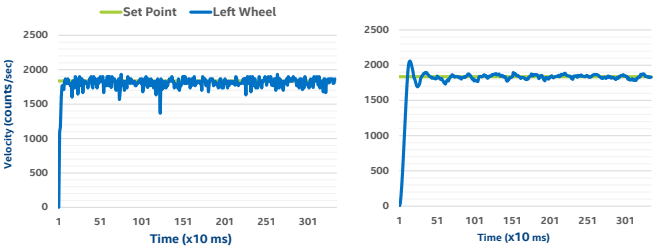


Fig. 11: Open loop and closed loop behavior of system with hardware implementation

SoC test results This accelerator is designed to integrate as part of a small test SoC specifically designed for cm-scale mini bot platform and drones. This SoC has light weight CPU with very small amount of onboard memory and other accelerators. Chip was fabricated in 22nm CMOS process. Once the test chip was back to enable detailed testing it was mounted on a quadrocopter (drone). Hardware platform consisted of a 550mmx250mm (without propeller) drone hosting a customized compute platform. The drone was designed to handle 2m/s and 1m/s forward and climb speeds respectively with an endurance of appoximately 12 mins Fig 10b. The drone was also fitted with mutiple sensors ranging form Intel Realsense T265 for localization, Intel Realsensee D435i for imaging and mapping, integrated IMU etc.

The compute platform consists of an Up Core plus (credit card sized high performance, low power board with quad core CPU) along with a custom silicon built to host a microcontroller class core along with the highlighted motion controller accelerator IP. The compute was designed to allow navigation, planning and motion control of the UAV to be seamlessly divided among the different types of compute available on board.

Details of the control loop architecture is provided in Fig 12. The outer loop PIDs operating on localization were designed to consume positioning data from the Realsense D435i at 50 fps and provides set points to the inner loop running on the custom SoC. Buit in bypass mechanisms in the accelerator

allowed efficient data flow between inner and outer loops. The motion controller was programmed to run at a 2.5ms feedback loop controlling Roll, Pitch and change in Yaw based on the set points from the outer control loop. The control action generated by the inner loop controller is converted to PWM commands in 1 shot mode to control the Electronic Speed Controller (ESC).

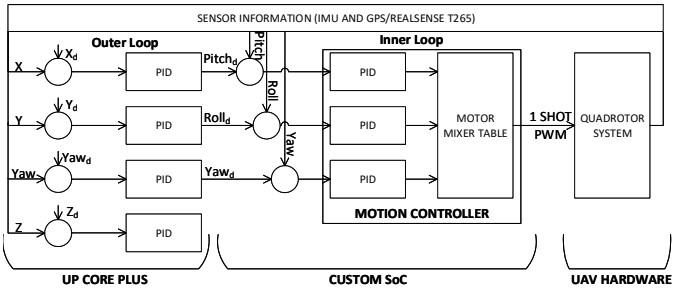


Fig. 12: Software architecture and partitioning for drone testing

Fig 13 shows performance of the controller while following a helical trajectory. Inner loop of the controller was configured to run at 400Hz which was the limitation of the ESC. Drone was able to follow the commanded trajectory accurately. It was able to achieve a loop rate of 4000Hz during standalone testing.

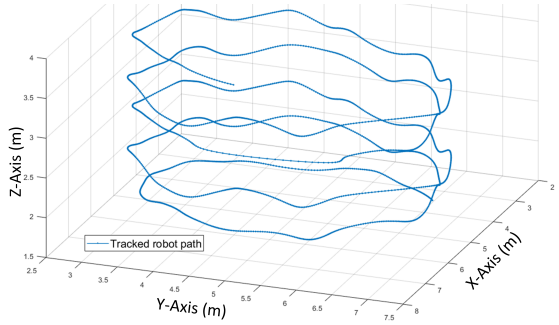


Fig. 13: Observed drone trajectory

Motion controller accelerator characterization was done and power, performance numbers are shown in Fig 14. Control loop timing comparison is done with a Software based controller implementation on an IoT edge platform, based on Intel Edison which consists of a dual core Atom CPU running at 500MHz and 4GB RAM. Table I shows the results of power, performance and memory comparison of the two. Results demonstrate accelerator operating at 0.5-4MHz (0.65V-0.8V) consuming 1-6mW. Size of the designed Motion Controller Accelerator is ~255K, 2-input NAND equivalent gate count.

V. Conclusion

In this paper we have presented a light weight accelerator design to enable completely autonomous Motion Controller for multi actuator systems. This helps meet the real time operation requirement and provides the capability to provide precise timing control with high precision. This offers significant power ~1000x and perform gains ~44x faster response over software based solutions while showing significant reduction in jitter.

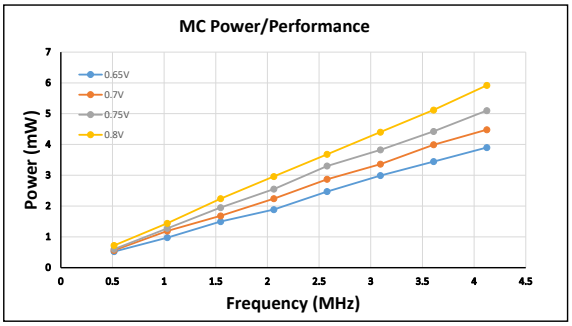


Fig. 14: Power vs Frequency plot of different voltage values for the Motion Controller accelerator on test SoC

TABLE I: Performance Comparison of MC Accelerator and Software Controller on Intel Edison

System	Memory KB	Execution Time ms	Energy μj
Accelerator	18	0.011	0.044
Edison	25	0.49	70.168

References

[1] R. Siegwart, I. R. Nourbakhsh, and D. Scaramuzza, *Introduction to Autonomous Mobile Robots*. The MIT Press, 2011.

[2] Jianxin Tang, "PID controller using the TMS320C31 DSK with on-line parameter adjustment for real-time DC motor speed and position control," in *IEEE International Symposium on Industrial Electronics Proceedings (Cat. No.01TH8570)*, vol. 2, pp. 786–791 vol.2, June 2001.

[3] H. D. Maheshappa, R. D. S. Samuel, and A. Prakashan, "Digital PID controller for speed control of DC motors," *IETE Technical Review*, vol. 6, no. 3, pp. 171–176, 1989.

[4] A. Trimeche, A. Sakly, M. Abdellatif, and M. Benrejeb, *Advances in PID Control*, ch. PID Controller Using FPGA Technology. Sept 2011.

[5] J. Lima, R. Menotti, J. M. P. Cardoso, and E. Marques, "A methodology to design FPGA-based PID controllers," in *IEEE International Conference on Systems, Man and Cybernetics*, vol. 3, pp. 2577–2583, Oct 2006.

[6] M. Kocur, S. Kozak, and B. Dvorscak, "Design and implementation of FPGA - digital based PID controller," in *Proceedings of the 2014 15th International Carpathian Control Conference (ICCC)*, pp. 233–236, May 2014.

[7] Wei Zhao, Byung Hwa Kim, A. C. Larson, and R. M. Voyles, "FPGA implementation of closed-loop control system for small-scale robot," in *ICAR '05. Proceedings., 12th International Conference on Advanced Robotics, 2005*, pp. 70–77, July 2005.

[8] D. A. Gwaltney, K. D. King, K. J. Smith, and J. Ormsby, "Implementation of adaptive digital controllers on programmable logic devices," in *5th Annual Military and Aerospace Programmable Logic Devices (MAPLD) International Conference*, Jan 2002.

[9] L. Samet, N. Masmoudi, M. W. Kharrat, and L. Kamoun, "A digital PID controller for real time and multi loop control: a comparative study," in *IEEE International Conference on Electronics, Circuits and Systems. Surfing the Waves of Science and Technology (Cat. No.98EX196)*, vol. 1, pp. 291–296 vol.1, Sep. 1998.

[10] Z. Zhang, A. Suleiman, L. Carlone, V. Sze, and S. Karaman, "Visual-inertial odometry on chip: An algorithm-and-hardware co-design approach," in *Robotics: Science and Systems XIII, Massachusetts Institute of Technology, Cambridge, Massachusetts, USA*, July 2017.