UNIVERSITY OF CALIFORNIA,
IRVINE

Very Low Power High-Frequency Floating Point FPGA PID Controller

THESIS

submitted in partial satisfaction of the requirements
for the degree of

MASTER OF SCIENCE

in Computer Science

by

Radhit Dedania

2022

DEDICATION

To my mother Alkaben Dedania who has been a constant source of inspiration and motivation for me along with other family members who unconditionally supported me during my journey from kindergarten to graduate school.

TABLE OF CONTENTS

Page

LIST OF FIGURES **v**

LIST OF TABLES **vi**

ACKNOWLEDGMENTS **vii**

ABSTRACT OF THE THESIS **viii**

1 Introduction **1**

2 Motivation **4**

3 Background and Related Works **7**
- 3.1 PID Controller 8
- 3.2 PID Controller Acceleration 9

4 PID Controller **11**
- 4.1 Controller Tuning 11
- 4.2 Controller Convergence 12

5 Accelerator System Design **15**
- 5.1 Accelerator Microarchitecture 16
  - 5.1.1 Single Controller 18
  - 5.1.2 Interleaved Double Controller 20
  - 5.1.3 Cascaded Double Controller 20
- 5.2 Simplified Floating Point Core 22

6 Evaluation **24**
- 6.1 Implementation Details 24
- 6.2 Performance Evaluation 25
- 6.3 Power Efficiency Evaluation 28
- 6.4 Analysis on Operator Re-Use and Pipe-lining 29

7 Discussion **30**
- 7.1 Benefits 30
- 7.2 Drawbacks 31

8 Use Cases **32**

9 Conclusion and Future Work **34**

Bibliography **35**

LIST OF FIGURES

Page

2.1 Conventional Control of a Micro-Aerial Vehicle 5

3.1 PID Controller. 8

4.1 PID Controller Tuning Process 13
4.2 PID Controller Convergence 14

5.1 Overall architecture of a PID accelerator. 15
5.2 Design of our PID Accelerator (re-used operators share fill patterns and are marked with a dashed-line boundary). 16
5.3 Detailed Control Flow of a PID accelerator. 17
5.4 Architectural Design of a single PID accelerator. 19
5.5 Architectural Design of an interleaved double PID accelerator. 21
5.6 Architectural Design of a cascaded double PID accelerator. 22

6.1 Comparison of Average Throughput across platforms. 27
6.2 Comparison of Power Efficiency across platforms. 28

8.1 Micro-Aerial Vehicle 33
8.2 Insect-Sized Robot 33
8.3 Swarm of Micro-Aerial Vehicles 33

LIST OF TABLES

Page

4.1 Final Configuration of Controller Variables 13

6.1 Chip resource utilization on the UP5K FPGA. 25
6.2 Chip resource utilization across different controller versions 25
6.3 Comparison of the evaluated computing platforms. 26

ACKNOWLEDGMENTS

I would like to thank Professor Sang-Woo Jun for guiding me through this thesis. He illuminated my way whenever I got engulfed in darkness. His patience and dedication towards this project were phenomenal. His time and effort were invaluable for the successful completion of the project. A special thanks to Professor Marco Levorato and Professor Eli Bozorgzadeh for their interest in the project and their contribution as committee members. I really appreciate every effort put in by all the committee members. In the end, I would thank all researchers, whose work in the form of publications, directly or indirectly motivated as well as inspired creative ideas and ultimately, paved way for this project.

ABSTRACT OF THE THESIS

Very Low Power High-Frequency Floating Point FPGA PID Controller

By

Radhit Dedania

Master of Science in Computer Science

University of California, Irvine, 2022

Assistant Professor Sang-Woo Jun, Chair

In this work, we propose the design of a floating-point Proportional-Integral-Derivative(PID) controller accelerator and present its implementation on a Lattice UP5K FPGA which attains a high throughput rate of 645-K samples per second for a single controller and a net throughput of 1032-K samples per second for an interleaved double controller at the expense of 20-mW of power consumption. Our single controller and interleaved double controller systems respectively achieve over 70× and 120× the performance of a similar sized microprocessor with comparable power constraints, and 5× the power efficiency compared to a large and more potent ARM Cortex-M4F capable of hardware floating-point operations. We obtain such a high performance with a systolic array design that uses simplified hardware floating-point operations that get synthesized on embedded DSP blocks of a low-power FPGA. Additionally, we support a simple treatment of complex reference signals such as sinusoidal inputs by storing the reference in an on-chip block RAM in the form of a time series. The level of power efficiency and high performance that we achieve on a small sized board is imperative for our target applications of micro, or insect-scale robotics.

Chapter 1

Introduction

Complex real-world systems like Cyber-Physical Systems (CPS), Robotic Systems, Self-driving cars, Autonomous Underwater Vehicles (AUV) and Unmanned Aerial Vehicles (UAV) have high-performance and accurate control systems as one of their core components. Low resource requirements complemented by simple algorithmic structure contribute to the popularity of the Proportional-Integral-Derivative (PID) controller in comparison with a variety of other control algorithms and systems under research and deployment. PID controllers take into account measured state of the system being controlled through a feedback mechanism and use this to generate control signals and thereby, fall into the class of closed-loop control systems. PID controllers are useful for a wide range of situations [29, 10, 4] due to their versatile adaptability attributed to their flexible parameters that can be tuned manually or algorithmically using one of the available standard techniques [36, 22, 19].

Synthesizing PID controllers on resource-constrained robotics systems or even CPS is a quite challenging task despite its simplicity. Insect-scale robotics has gained prominence in recent years and proves to be an exciting field of modern research [15, 21, 20, 16]. Lack of essential computing power results in low-power micro-controllers' struggle to keep up with the required signal rates, with each sensor input sampled at 10-KHz or higher [15], though they may meet the stringent power budgets. Absence of a dedicated floating-point processing unit in many low-power micro-controllers makes this matter worse. Fixed-point PID controllers are not amenable for many modern-day applications due to their rigidness and software emulation of floating-point arithmetic degrades performance by an order of magnitude [3, 35]. On the other hand, computationally powerful micro-controllers possessing floating point units breach the upper threshold of size and power budget requirements of these micro-scale robots [21, 20, 16]. It is highly desirable to use dedicated Application-Specific Integrated Circuits (ASIC) equipped with low-power cores and floating point operators for this task but their fabrication is very expensive.

In this work, we present a solution approach based on off-the-shelf low-power Field Programmable Gate Array (FPGA) chips, which meet system specifications in every critical aspect including performance, power efficiency, cost and size. We use low-cost lattice UP5K FPGA for synthesizing our solution and build simplified, resource-efficient floating-point operations on its Digital Signal Processor (DSP) cores. We implement the PID controller using low-power FPGA resources and optimize its design to fit it within these available on-chip resources. Our controller allows storing reference signals as a time series in the FPGA BRAM in order to support handling of complex reference signals like sinusoidal signals for control of flapping wing motion and wing speed of a micro-scale robot. Our work has been accepted to appear at the 12^{th} International Symposium on Highly Efficient Accelerators and Re-configurable Technologies(HEART 2022) and will be published by ACM through the International Conference Proceedings Series [11].

Our single controller and interleaved double controller solutions respectively attain a 26-MHz & 23-MHz clock frequency on the Lattice UP5K FPGA and result in a 20-mW power budget as they adequately fit within the board while consuming 75% & 90% of the chip resources. As far as throughput is concerned, our single controller design emits 637-K control outputs per second which is roughly equivalent to a computational throughput of 7-MFLOPS. On the other hand, our interleaved double controller emits a total of 1032-K control outputs per second which is roughly 11.4-MFLOPS. For the above two cases, we get 70$\times$ & 120$\times$ the computational throughput of an ATMega328 micro-controller which has similar size and clock speed as our UP5K FPGA and also consumes comparable power. Lack of a floating point unit on ATMega328 also partly degrades its performance. We also evaluate our system against a more potent and power-consuming ARM Cortex-M4F which is equipped with floating point operators and clocks at 180-MHz but far exceeds the size and power budget constraints of our target application. In comparison to this highly capable system, our accelerator displays a comparable computational performance and more saliently, we outperform it drastically in terms of power efficiency. We maintain the upper hand in power efficiency even when compared to much more powerful and sophisticated desktop-class systems such as Raspberry Pi machines.

The rest of this work is structured as follows. In Chapter 3, we cover the background information and prior works related to PID controllers and their accelerators. In Chapter 5, we present the design details of various components constituting our accelerator architecture. We evaluate the performance of our solution with existing systems in terms of throughput and power efficiency in Chapter 6. At last, in Chapter 9, we conclude our discussion with suggestions for future research in this direction.

Chapter 2

Motivation

The motivation for this work comes from the power budget, cost, size and throughput constraints needed for our target applications of insect-scale robotics and micro-aerial vehicles. These use cases require controllers implementations which have low power consumption of the order of 10 mW-100 mW [21, 20, 16], reasonably high throughput over 10 KHz [15] and small sized boards, which measure a few inches in both dimensions of length and width, that can be embedded on the target devices. Another inspiring reason for this work has been the limitation of fixed-point arithmetic which results in significant accuracy loss even though it works on low-power devices.

Much needed support for floating-point operations that conform to IEEE-754 Standard [12] requires use of high-power devices and its implementation alone takes up a sizeable chunk of available resources on the device. Thus, the need of the hour is to get good from both these worlds by using simplified floating-point operations that work on low-power devices. As micro-processors and micro-controllers consume a lot of power, are relatively large and too generic due to which they are not optimized for a specific task, it is better to avoid them for our target applications. Application-Specific Integrated Circuits (ASICs) are the best choice for high-performance and low-power applications but their fabrication is very expensive and additionally, are not re-configurable like Field-Programmable Gate Arrays (FPGAs). FPGAs are the best choice as they are relatively cheaper and these integrated circuits can be reprogrammed multiple times at no additional costs. Moreover, they provide dedicated computational units in the form of Digital Signal Processing (DSP) blocks and have adequate on-chip Block RAM (BRAM) memory for storage purposes.

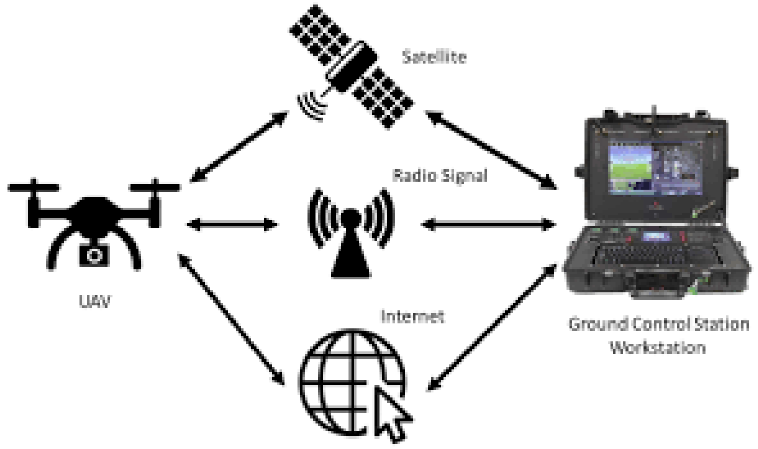

Figure 2.1: Conventional Control of a Micro-Aerial Vehicle

Source: https://doi.org/10.1007/s42452-020-2749-5

To understand the application requirements, consider a motivating example of a micro-serial vehicle that is controlled from a command center like desktop, data center, etc. in a traditional way as shown in Figure 2.1. This mode of control requires communication of data over wireless network. The sensors on the vehicle send observed inputs to the command center which then computes control outputs based on these inputs and transmits them back to the vehicle for next course of action. There are many limitations of this control scheme. There is a round-trip latency involved between the sensor inputs and control outputs which could become a bottleneck for throughput. As the network is unreliable, both the sensor inputs and control outputs can get lost or arrive out-of-order. There are also security concerns about possible snooping over the communication channel and malicious modification of data. Hence, there is a need arises for an embedded control solution that comfortably fits on the vehicle, is low-powered so that its battery lasts for a longer duration, is comparably cheaper and meets computational requirements of the target device.

Chapter 3

Background and Related Works

The prime objective of a controller in any dynamic system is to modulate the output value of the system in order to synchronize it with a given reference input value. This is generally expected to be carried out in an automated fashion. There are two types of control loop systems, namely, open-loop and closed-loop available to accomplish this task. The control inputs of an open-loop system are independent of the measured output variable. Despite their design simplicity and economical nature, they are quite inaccurate and highly vulnerable to fluctuations in the environment which renders them unreliable. These drawbacks of open-loop systems can be overcome with the introduction of an output to input feedback mechanism that closes this open loop in a closed-loop control. Thus, a closed loop system also considers value of the output variable, which is mainly sensor measurements, and contrasts it with a provided input reference to produce an error. If the final objective is to equate these input and output values, then the complete control problem gets transformed into one whose end goal is to minimize this error. The addition of feedback raises accuracy score of a closed-loop system even in non-linear settings [8] and makes it more resilient against extrinsic perturbations as opposed to open-loop control systems [13]. Some works have also explored adaptive hybrid control that transitions between open-loop and closed-loop for carrying out repetitive tracking [31].

3.1 PID Controller

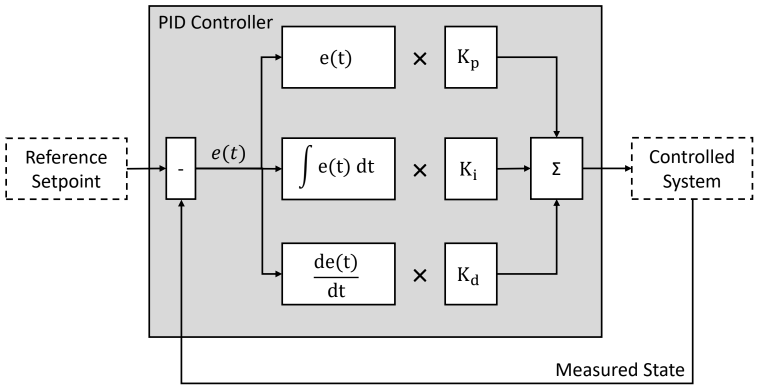

Figure 3.1: PID Controller.

Proportional-integral-Derivative (PID) controllers [6] are the most popular and widely used controllers among various other sub-categories of the class of closed-loop control. Despite having a simple algorithmic structure [32], PID controllers are less prone to environmental disturbances [28]. Moreover, they are quite robust to tuning errors and mismatches [23], and reasonably economical due to dependence on fewer resources [26]. These controllers are composed of proportional (P), integral (I) and derivative (D) parts as presented in Figure 3.1 and this class of controllers derive their name from these three components. An output proportional to the error signal $e(t)$ and scaled by a gain factor of K_p is generated by the P component. The error signal $e(t)$ is integrated over time $\int e(t)dt$ by the I component and multiplied by the constant K_i before it gets added to the control input. The D component calculates the time differentiation of the error signal $\frac{de(t)}{dt}$ and weighs it by a multiplicative coefficient K_d. Each of the above components adds some value to the final control signal by taking care of different output features like overshoot, oscillations, etc. and hence, play an integral role in the correct functionality of the controller.

The parameters K_p, K_i, and K_d must be correctly tuned for the PID controller to accurately adapt to the external environment. This can be achieved by using any of the three prominent tuning techniques which are heuristic-based, rule-based and model-based methods. An important category of heuristic-based methods is trial and error tuning [1] which is a manual recursive process of setting controller parameters that takes into account observable characteristics like overshoot, oscillations, steady-state error, etc. for faster and better tuning. Rules-based methods rely on the response model of the given system and transform it into a mathematical expression that guides the parameter tuning process. There are many distinct rule sets available for this purpose each of which requires differing amount of system information with varying accuracies [36, 22, 19]. Ultimately, model-based methods [17] provide a framework for setting parameters in a way that meets the given control specifications but they are quite time-intensive and are mainly used for critical applications.

3.2 PID Controller Acceleration

PID controllers must swiftly record the behaviour of the controlled system and rapidly act on it in order to be helpful. This implies operating on a sampling rate of 10-KHz or higher [15]. A floating-point implementation of the PID control algorithm puts tremendous overhead on low-power embedded processors due to inherent accuracy requirements. Thus, enhancing power efficiency of the PID controllers is an active area of research.

To address this issue, one method is to utilize parameter quantization and then rely on cheaper fixed-point arithmetic for computation of control inputs. However, it results in a significant loss of accuracy [35] and also complicates the control design to a greater extent [25]. Another approach is to employ application-specific hardware accelerators like FPGAs. Some prior works avail of techniques based on distributed arithmetic to minimize resource utilization and chip area for synthesising the design [9]. Other designs trade-off on-chip resources for parallel control flow architecture [34] to accelerate computation of control input which also pays reasonable attention to its accuracy. Some projects have implemented multiple controllers on the same chip but they worked with a large and powerful Xilinx Spartan-6 board and furthermore, did not utilize floating-point arithmetic which is essential for higher accuracy and better control [2].

Many optimized standard FPGA implementations are also available from multiple vendors. One such implementation on a Spartan-3 board focuses more on resource efficiency through a parallel as well as a compact control design and thus, achieves it at the cost of increased latency[33]. Another version utilizes Vivado HLS Design Suite for optimizing a RTL implementation constructed from a C++ based instance of the control design. There are multiple applications of PID controllers based on FPGA architecture including but not limited to stabilization of DC-DC converter [37] and non-linear control of manipulator robot arm [24].

There are variety of PID controller designs on FPGA. As far as we know, there are no published implementations that cater to target applications of insect-scale robotics which requires a lower threshold of 10-KHz sample rate realised in tens of mW power.

Analog versions of PID controllers also exist. They are cheaper but are much slower. Additionally, relative to digital implementations, they are characterized by increased tuning difficulty and decreased stability [3].

Chapter 4

PID Controller

4.1 Controller Tuning

The tuning process of a PID controller involves adjusting the gains of various paths (K_p, K_i, K_d) to obtain a steady-state error of zero. There are multiple standard methods available for setting the correct values of these controller parameters as discussed in Chapter 3. The tuning of a controller is specific to a domain/task of interest. Once it changes, this process has to be repeated again from scratch. For setting parameter values in this work, we used heuristic-based tuning[18] which is a manual process of iterating through controller parameter values based on some observable characteristics like overshoot, oscillations, steady-state error, rate of convergence, etc. The plots of reference signal, control input signal and observed signal at various phases during the tuning process used in our work are shown in Figure 4.1. These plots highlight the importance of observable characteristics in the manual tuning process. Some of them are magnified at the initial time steps to emphasise more on the legibility and distinction aspects.

In this method, we start off by setting the value of every controller parameter to zero.

Then, we decide on the need of a positive or a negative feedback by setting K_p to a small positive and negative value respectively and depending on the value of steady-state error, we select one which provides a minimum value. Now, we change the value of K_p in orders of magnitude and continue till there is no significant change in the observed steady-state error which should decrease over time. Subsequently, we start changing the value of K_d and stop when most of the oscillation overshoots die out due to the damping effect of the parameter. Finally, we start adjusting the value of K_i to ensure further reduction in steady-state error but it comes at the cost of instability in the form of some overshoot. To stabilize it, values of other parameters can be modified. Additionally, two or more parameters can be scaled simultaneously by a factor to reduce steady-state error while keeping transient response the same. This iterative process is repeated till the final steady state error becomes close to zero and there are no visible oscillations that aim to destabilize the system by inhibiting steady-state convergence. The tuning process can also be stopped once desired characteristic values provided in specifications(if any) are reached. It should be noted that very high gain values often lead to chattering although they might meet characteristic requirements and hence, should be avoided at all costs.

4.2 Controller Convergence

The convergence of the controller to steady-state error depends on a variety of factors like controller parameters (K_p, K_i, K_d), reference signal (ref), value of initial observation (obs), sampling frequency (dt), etc. The final configuration of chosen variables including controller parameters is shown in Table 4.1 below. The final convergence plot is illustrated in Figure 4.2 along with a magnified version of the initial part of the plot for the sake of legibility. It can be observed that for the above choice of parameters, the convergence to steady-state error is very fast as it occurs within tens of controller cycles and is devoid of any overshoots which

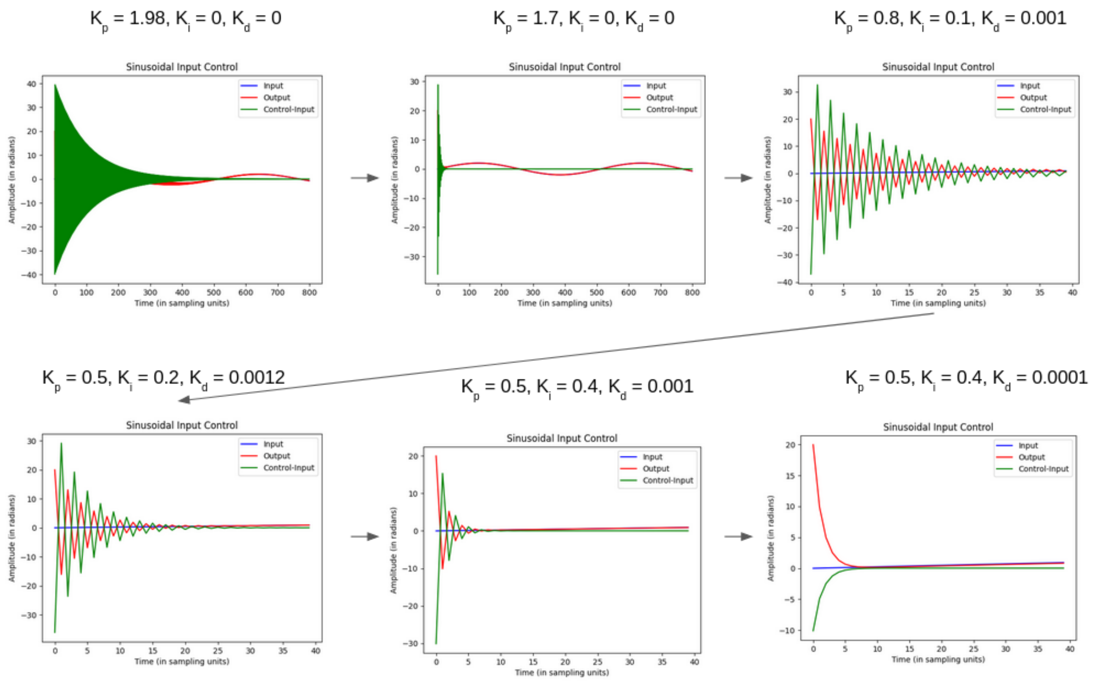

Figure 4.1: PID Controller Tuning Process

generally give rise to oscillations. Moreover, it is stable as once the measured output comes within $\pm 2\%$ boundary of the reference signal, it does not cross it again and remains inside it forever.

Table 4.1: Final Configuration of Controller Variables

Controller Variable	**Final Value**
Reference Signal (ref)	$2sin(2\pi t)$
Initial Observation (obs)	20
K_p	0.5
K_i	0.4
K_d	0.0001
Sampling Frequency (dt)	512
Sampling Time ($\frac{1}{dt}$)	$\frac{1}{512} = 0.00195$

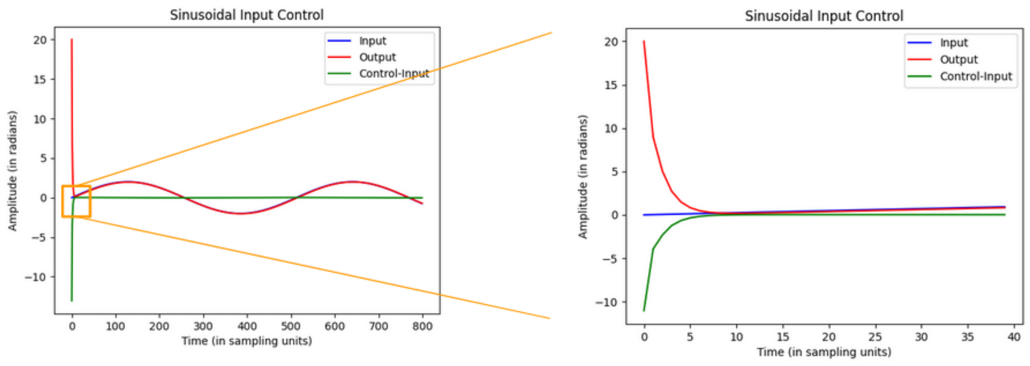

Figure 4.2: PID Controller Convergence

Chapter 5

Accelerator System Design

The overall high-level architecture of the system containing our PID controller accelerator is shown in Figure 5.1. Time series representation of the reference setpoints sampled at rate $\frac{1}{dt}$ and parameters like tuning constants (K_p, K_i, K_d) along with period-dependent dt values are stored in the configuration array of the PID accelerator. This array can be set up at compile time and modified in a dynamic fashion over either the UART or the SPI channel by the host controller. Similarly, the sensor inputs and controller generated outputs are fed and emitted respectively from the accelerator over the SPI channel.

Although the I/O to and from the system can be carried out using 8-bit or 16-bit fixed-point integer values, every internal arithmetic operation is executed using 32-bit floating point operators that are realised using a few on-chip Digital Signal Processing (DSP) blocks.

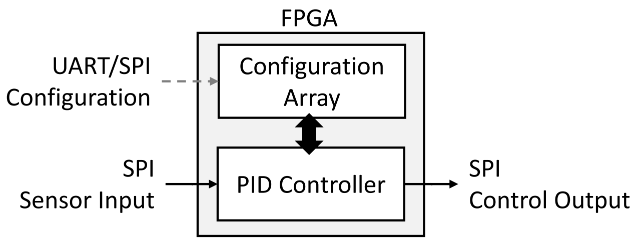

Figure 5.1: Overall architecture of a PID accelerator.

5.1 Accelerator Microarchitecture

The architectural design of the interior of our PID accelerator is illustrated in Figure 5.2. We highlight two important aspects of our accelerator micro-architecture: Use of a Block RAM (BRAM) array for storing reference setpoints as time-series and time-sharing of floating point adders as well as multipliers. A more detailed flowchart for a PID controller depicting the operations executed in each cycle and the flow of control between them is shown in Figure 5.3.

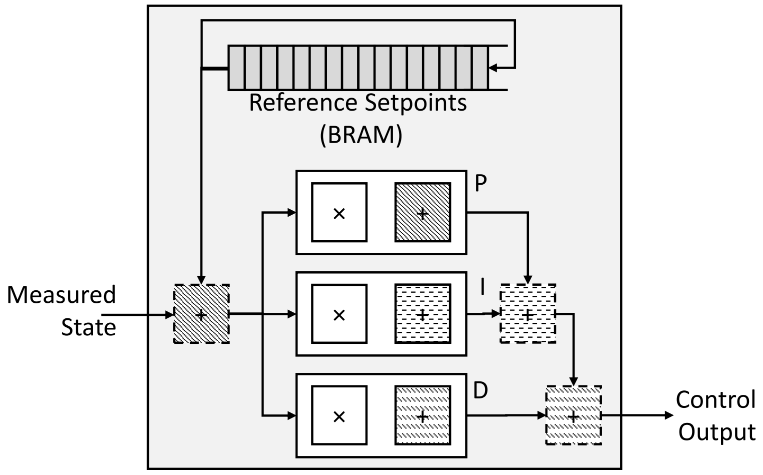

Figure 5.2: Design of our PID Accelerator (re-used operators share fill patterns and are marked with a dashed-line boundary).

As the sensor input of the next time-step depends on the control output of the previous time-step, the design of our accelerator takes advantage of this information and hence, there is no overlap between control output computations across different time-steps. To benefit from this, our accelerator generates and time-shares a small collection of floating point operators. Specifically, we create three instances of floating point adders and multipliers, so that a pair of one adder and one multiplier is available for each of the three P, I and D paths. All the operations performed exterior to the P, I and D paths share the floating-point operators, that were created within these paths, in a deterministic scheme.

The floating-point operations carried out in the three paths and outside these paths do not overlap which makes our chosen design trade-off productive. Additionally, there are similar number of addition and multiplication operations performed inside and outside these paths which complement our design choice. The synthesis of a floating-point operator occupies a sizeable portion of the chip's re-configurable fabric which will be covered in Section 6. Thus, without operator re-use, it would not be feasible to fit the floating-point PID controller design on our target Lattice iCE40 UP5K FPGA.

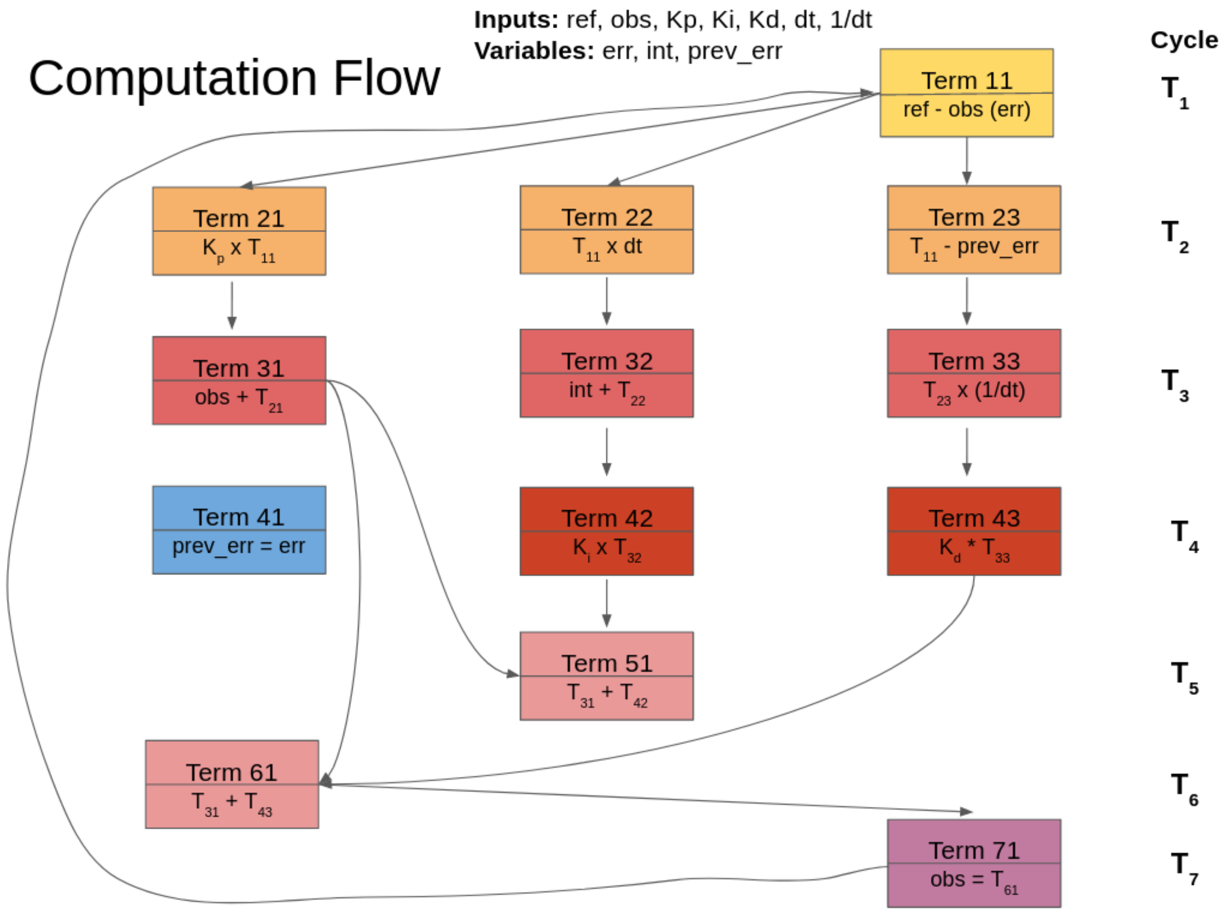

Figure 5.3: Detailed Control Flow of a PID accelerator.

Apart from that, the inner implementation of each of these paths is a quite elementary implementation of the PID control algorithm. Our assumption of non-overlap of control signal computations across time-steps reduces the parallelism available for exploitation. Due to this, we are excused from speeding too much effort and resources in path pipeline design. Alternatively, we take a straightforward, blocking approach of feeding the arithmetic computation requests into the floating-point operator units, waiting for these results to become available, and then inserting the obtained control outputs into the same units for next round of computation. The parallelism across the P, I and D paths is only exploited for single controller case. Considering the inherent dependency constraint between current control output and next sensor input, we observe that our design is very effective in terms of resource-efficiency as well as raw cycle count latency when compared against the reference implementation available from Xilinx [7] .

The second re-utilization of the I-path adder to modify the integrated state register by adding the current control output value is not shown in Figure 5.2. This occurs after the I-path adder is used to first add the results of P and I path together and when this result is being added to the output of the D-path by the D-path adder, there is a reuse of the I-path adder to compute the updated integrated state value.

Due to on-chip resource efficiency and floating-point operator reuse as discussed in Chapter 6, we were able to implement multiple PID controllers (multi-controllers) on the same chip without incurring much re-configurable logic as well as block storage overhead and at no additional requirement of DSP computational units. The applications of multi-controllers are further elaborated in Chapter 8. The design details of the single, the interleaved multi-controller and the cascaded multi-controller are discussed in the following subsections.

5.1.1 Single Controller

The computational flow of a single controller is shown in Figure 5.4. The periodic reference setpoints along with other controller variables are passed over the Serial Peripheral Interface (SPI) Input channel. The reference setpoints are then saved as a time-series in a Block RAM (BRAM) FIFO which uses on-chip storage. A constant buffer, which is inherently a random-access vector, is used to store the received controller variables. Apart from these, it also stores some state variables (represented using light orange blocks in Figure 5.4) like integral, previous error, etc. which are necessary for maintaining continuity of computational flow between two distinct time steps (states). The controller computation starts once all the inputs are stored appropriately. The floating-point operator units are coloured according to a legend to lay emphasis on their time-sharing nature. In every computation round, error signal and control output signal are emitted over SPI Output channel. The superscripts are used to denote time-step corresponding to an emitted output. The control algorithm execution is blocking in that computation of the next control output starts after previous control output is emitted as the observed signal changes according to generated control output and serves as an input for next round of computation.

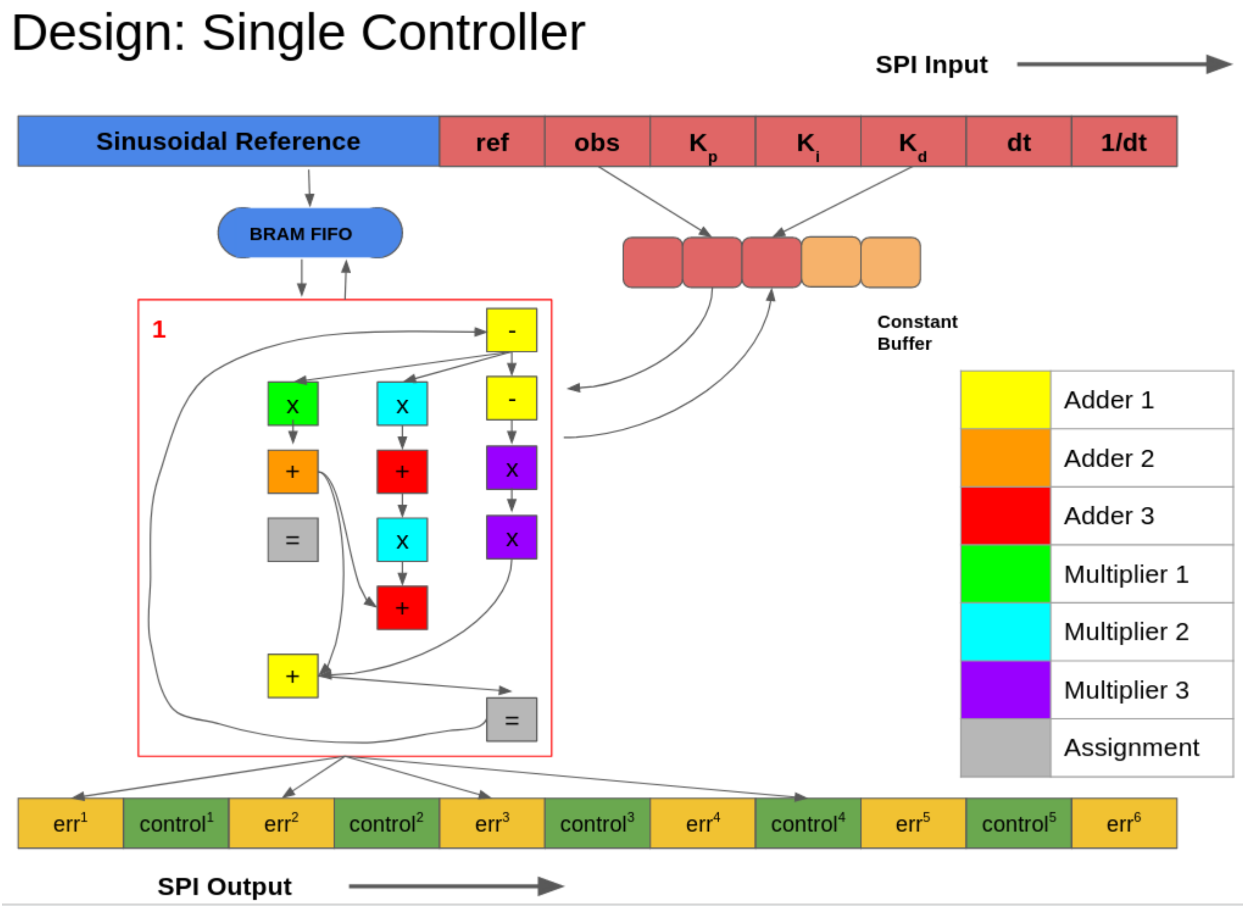

Figure 5.4: Architectural Design of a single PID accelerator.

5.1.2 Interleaved Double Controller

The computational flow of an interleaved multi-controller shown in Figure 5.5 is similar to that of a single controller as far as SPI inputs and SPI outputs are concerned. However, two sets of reference setpoints and controller variables are provided for the two distinct controllers. This allows both controllers to simultaneously execute on completely different inputs and does not complicate the implementation. The execution flow of both the controllers is interleaved so that floating-point operations, which are scheduled to fire in a specific cycle, are computed simultaneously irrespective of the controller they belong to. This design further enhances the time-sharing behavior of floating-point units and reduces their idle time by appropriate pipe-lining of computations which is vividly depicted using colored units in Figure 5.5. The computational throughput almost gets doubled. The emitted error signals and control output signals also get interleaved over SPI Output channel. The superscripts denote time-step and the subscripts mark the identity of the controller to which an output belongs.

5.1.3 Cascaded Double Controller

The cascaded double controller shown in Figure 5.6 is very similar to a single controller except that it alternatively computes control outputs of two different controllers. Although it doesn't enhance throughput or increase resource efficiency, it allows two controllers to time-share the same available resources as is lucidly shown in Figure 5.6 using colored floating units. Just like the interleaved case, there are two sets of reference points and controller variables provided over SPI Input channel which permit individual controllers to act on their own unique input. The execution flow retains the same blocking nature of a single controller as now instead of waiting across time-steps of the same controller, there is a holdup across different controllers as one controller has to wait for the other to finish before starting its own

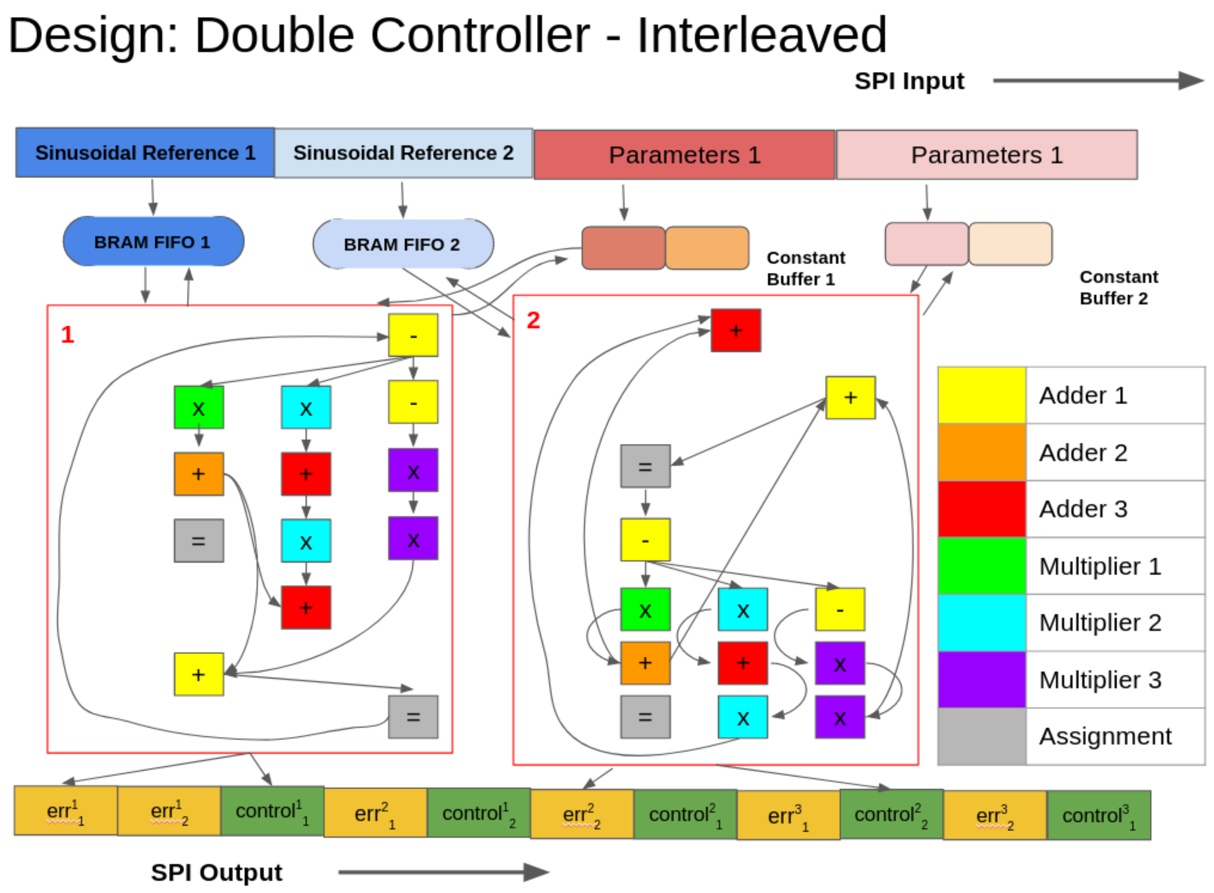

Figure 5.5: Architectural Design of an interleaved double PID accelerator.

execution and vice-versa. Additionally, there are skip connections (represented by dotted blue arrows in Figure 5.6) which are used when one of the controller has already converged while the other is still trying to do so. They are just requirements of implementation design to branch control flow suitably. The emitted error and control output signals for a particular time-step are consecutive over SPI Output channel unlike interleaved controller but there is interleaving across controllers for the same time-step. Similar to interleaved case, the superscripts denote time-step and the subscripts mark the identity of the controller to which an output belongs.

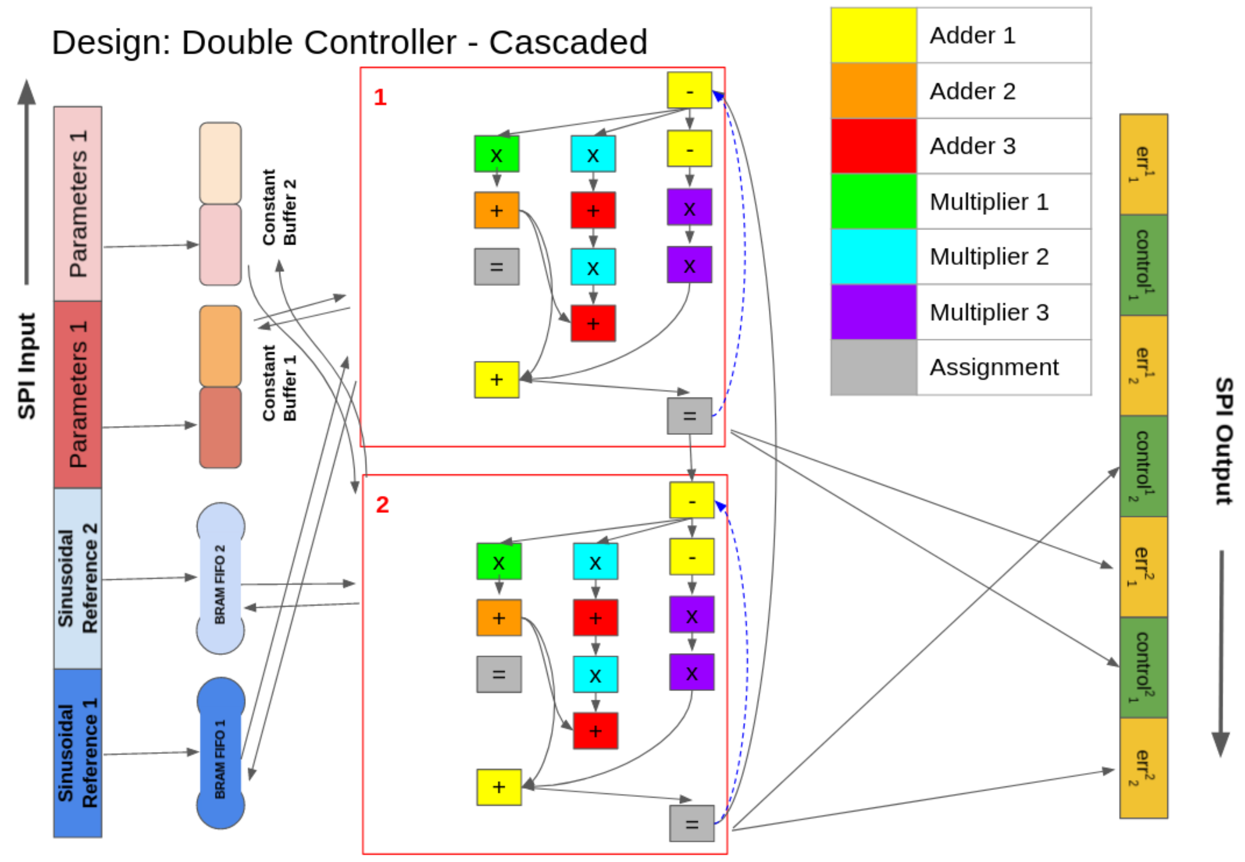

Figure 5.6: Architectural Design of a cascaded double PID accelerator.

5.2 Simplified Floating Point Core

To enhance performance efficiency, our design utilizes a simplified implementation of floating-point operations. These simplification methods can be broadly categorized into two classes: Approaches that affect accuracy (marginally), and approaches which do not.

For instance, our simplified floating point cores do not operate on special states like Infinity and Not-a-Number (NaN) explicitly. However, they do not have any significant impact on accuracy as these states would only be reached as a result of incorrect behaviour of the PID control algorithm which is not expected to happen.

On the contrary, the simplified floating point cores do not support operations on subnormal numbers and this may affect accuracy if an underflow occurs in normal floating-point representations due to exceptionally small fractional values generated by control arithmetic. Our operators handle such small fractional values by flushing them to zero, i.e., by explicitly setting to zero. The same optimization is applied by many SIMD and GPU implementations for efficiency purposes, including ARM NEON [27] and Intel SSE [5]. In practice, this optimization does not cause any significant impact on the accuracy as all computations ins our system involve values in the same scale, where a flushed zero is equivalent to an infinitesimally small value.

The one optimization that can result in slight loss of accuracy is the implementation of a floating-point multiplier operator using a 18-by-18 integer multiplier. While the bit-width of the mantissa part in a single-precision floating-point number is 23 bits, the actual maximum multiplication bit-width provided by the DSP blocks on the UP5K board is 18 bits. As a result, there is degradation in accuracy. Alternatively, for each multiplication operator, we could have utilized two DSP blocks to retain full accuracy, but based on the evaluation results of our implementation which confirmed the negligible impact of 5 least-significant bits on system behavior and for the sake of resource efficiency, we decided against it.

Chapter 6

Evaluation

6.1 Implementation Details

Our accelerator was implemented on the low-power, low-cost and small-sized Lattice iCE40 UP5K FPGA, which has many beneficial characteristics for the accelerator. It weighs a few milligrams [14] only and is quite cheap, at around $5 per chip. Moreover, it has a significantly low, mW-scale power budget, which will be highlighted with our accelerator in this section. Additionally, it also contains eight DSP blocks whose seasoned arithmetic logic is effective in executing 18-bit multiplication and accumulation in a single cycle. Such a capability is very rare among contemporary low-power FPGAs. Finally, there is additional support from open-source toolchains like Yosys and nextpnr [30] as the chip has already been reverse-engineered.

The major resource utilization of our accelerator and its sub-components on the UP5K FPGA chip is shown by Table 6.1. The entire single controller design including all the considered optimizations, fits easily into the chip and consumes 75% of available Logic Cells. As each floating point adder takes up 14% of the chip, our approach of reusing the three adders is validated as three extra adders cannot be accommodated on the chip. The comparison of resource utilization across different controller versions is listed in Table 6.2. It should be noted that there is no change in DSP utilization across the three controllers while the double controllers have 15% and 23% logic cell and memory overhead over single controller.

The results shown in the following sections are for sinusoidal reference signal. Experiments were carried out on constant reference signal as well but those results have not been covered here as they follow a similar trend albeit with a higher performance. Another reason for excluding the constant reference results is that sinusoidal signal is more challenging and has a large number of interesting applications compared to the constant signal.

Table 6.1: Chip resource utilization on the UP5K FPGA.

	1x Adder	1x Multiplier	Total System	UP5K Availability
Logic Cells	729 (14%)	343 (6%)	3998 (75%)	5280
BRAM	0	0	7 (23%)	30
DSP	0	1 (13%)	3 (37%)	8

Table 6.2: Chip resource utilization across different controller versions

	Logic Cells	BRAM	DSP
Single Controller	3998 (75%)	7 (23%)	3 (37%)
Double Controller (C)	4797 (90%)	14 (46%)	3 (37%)
Double Controller (I)	4778 (90%)	14 (46%)	3 (37%)
UP5K Availability	5280	30	8

6.2 Performance Evaluation

The PID controller accelerator, in entirety, uses **41** cycles for computation of a single control output which is the latency between sensor data input and control signal emission. For the sake of performance evaluation, we assume that sensor data for next computation is accessible as soon as the control output from previous computation gets emitted which entails that one control signal is produced every 41 cycles. This implies a throughput of over **645K** control outputs emitted per second for the single controller version while a net throughput of over **1032K** control outputs emitted per second for the interleaved double controller implementation. In spite of using many techniques to minimize on-chip resource usage, our work results in a latency of 41 cycles per emission which is quite competitive to a standard implementation from Xilinx [7], which obtained a latency of 37 cycles using a hand-optimised RTL design that utilized 5 hefty Xilinx DSP48 blocks.

We assess the performance of our accelerator in comparison with some popular and comparable embedded computing systems. The description of the platforms used for evaluation is provided in Table 6.3.

Table 6.3: Comparison of the evaluated computing platforms.

Processor	Clock Speed	Hardware Float Units	Cores	Power Consumption
AVR ATMega328	16 MHz	No	1	$\sim 20\ mW$
ARM Cortex-M4F	180 MHz	Yes	1	$\sim 200\ mW$
ARM Cortex-A72	1.5 GHz	Yes	4	$\sim 6\ W$
Lattice UP5K (S) **(This Work)**	26 MHz	Yes	N/A	$\sim 20\ mW$

The processors used for comparison which are listed in Table 6.3 are from the following systems: Arduino Uno, Arduino Teensy 3.6, and Raspberry Pi 4 B, respectively (from top to bottom). We take a note of the fact that amongst all the above processors, ATMega328 is the only platform that has power consumption comparable to our accelerator. Our target applications are insect-scale robotics and micro-aerial vehicles that require a total power budget of 10 mW to 100 mW [21, 20, 16]. **This power constraint is satisfied only by the ATMega328 processor and our accelerator**. The purpose of comparison with other power-hungry and more capable processors is only to provide a holistic picture of the performance and power efficiency aspects of our accelerator in the respective landscapes.

The comprehensive performance comparisons between these platforms are depicted in Figure 6.1. An important point to note is that for Cortex-A72.1, which is a 1.5-GHz processor, only single core performance is considered as otherwise its parallel performance will render the chart illegible due to an order of magnitude difference between its numbers and that of the rest. We observe that our accelerator surpasses the performance of ATMega328 processor which has a comparable power budget and size. Our accelerator also displays competitive performance in comparison to the Cortex-M4F processor equipped with a very rapid clock of 180 MHz which is an order of magnitude faster than ours. The performance of the Cortex-A72 processor is way ahead of other systems but its order of magnitude achievement can be attributed to its desktop-grade architecture. Thus, its comparison with other platforms is only for illustrative purposes.

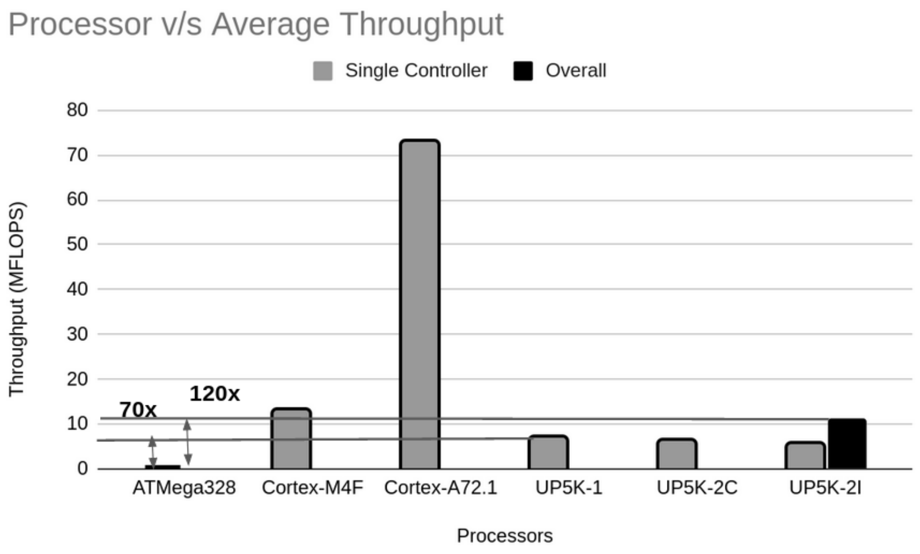

Figure 6.1: Comparison of Average Throughput across platforms.

The average throughput performance of the ATMega328 processor, meaning its rate of control signal emission, is 8.9K control outputs per second and fails to meet the 10-KHz threshold needed for some insect-scale robotic applications [15] even if its whole chip was utilized for implementing PID control.

6.3 Power Efficiency Evaluation

As far as power efficiency is concerned, our accelerator outperforms all other systems. The power efficiency comparison in terms of MFLOPS/W is presented in Figure 6.2. Our accelerator obtains almost two orders of magnitude better power efficiency than a comparable ATMega328, and over 5× the power efficiency of the Cortex-M4F, which has the second-best efficiency among all comparison platforms after our FPGA accelerator.

The Cortex-A72.4 results are obtained by multiplying the single-thread performance by 4 to get an estimate of its four-core performance. This four-core performance along with power consumption information of the whole Raspberry Pi 4B system is used to compute its power efficiency due to measurement limitations. Every other comparison system only measures the isolated chip power consumption.

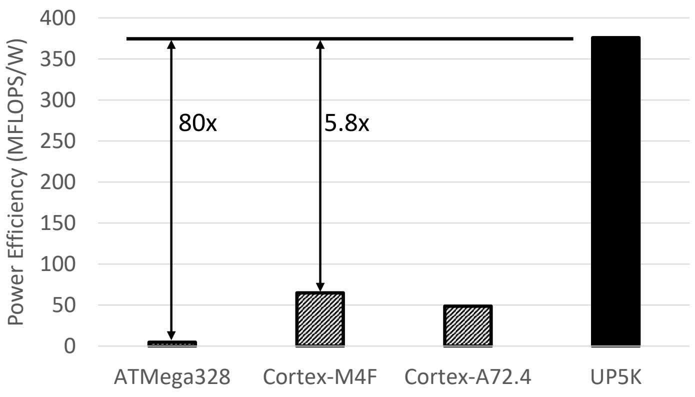

Figure 6.2: Comparison of Power Efficiency across platforms.

We want to highlight that in addition to our accelerator obtaining two orders of magnitude better efficiency when contrasted with a comparable power budget platform, after observing the power efficiency trend with more powerful processors like Cortex-A72.4, we expect with high likelihood that our accelerator will continue maintaining its power efficiency dominance over most standard general-purpose processors.

6.4 Analysis on Operator Re-Use and Pipe-lining

We take a note of the fact that the implementations of both multiplier and adder are fully pipelined, but can be used only in a blocking manner with a single request in-flight at any given time. We could use the same pair of multiplier and adder operators for all PID paths on a time-sharing basis without much reduction in performance. The removal of two pairs of multipliers and adders would have led to increased resource efficiency by reducing the utilization of Logic Cells by over 50% and this would have come at the expense of two additional cycles of latency. Since this does not have any considerable impact on power efficiency, we stick to our simplistic parallel approach which also fits snugly on the chip.

For the interleaved double controller, the computational throughput almost doubles when compared with the single controller. This is due to parallel execution of two single controllers in an interleaved fashion. There is an increase in resource efficiency due to decrease in idle time of floating-point operators as a result of pipe-lining architecture. Hence, the same DSP blocks are utilized for synthesizing the control design. However, there is a minimal overhead in terms of re-configurable fabric and block storage due to availing of two different sets of inputs. But, as there is still availability of more logic cells and storage resources, this overhead is irrelevant. For the cascaded double controller, resources are time-shared within the same controller as well as across the two controllers. On the downside, the throughput of individual controllers is halved while the overall throughput and resource utilization remain the same.

Chapter 7

Discussion

Our PID controller accelerator implemented on a Lattice iCE40 UP5K FPGA makes use of simplified floating-point operators, time-shares these operators for both single & multi-controllers and has support for pipe-lining of these operators for better on-chip resource reuse in case of multi-controllers. Although this system design has many advantages, it has many downsides of its own.

7.1 Benefits

There are many advantages of our accelerator architecture. It is implemented on a low-power FPGA which is quite small in size and is very cheap. It also meets the throughput requirement of 10 KHz which is needed for our target applications of insect-scale robotics. The chosen design works on periodic reference signals and also supports simplified floating-point operations for better accuracy as compared to fixed-pint arithmetic. It ensures very high resource efficiency through time-sharing and pipe-lining of floating-point operators at minimal cost to the complexity of the circuit logic.

7.2 Drawbacks

The prominent disadvantages of our approach include the requirement of task-specific tuning of the PID controller accelerator. Every time the domain changes, the controller needs to be adjusted to the new environment as its parameters are very sensitive to reference signal, domain-specific system model and other tuning characteristics like overshoot, steady-state error, etc. Our accelerator does not support tracking of a non-periodic reference signal as we store the entire input as a time-series for higher throughput. Real-time control still works in case of non-periodic inputs but can readily become a bottleneck for throughput. Lastly, the use of simplified floating-point cores result in a slight but insignificant loss of accuracy due to an upper threshold of 18 bits imposed by the chip on the multiplication operation.

Chapter 8

Use Cases

Our target applications for a single controller implementation are insect-scale robotics and micro-aerial vehicles. However, this work can be used for embedded control across multiple domains including but not limited to tiny ground robotics, little underwater autonomous vehicles, small-scale low-power cyber-physical system components, etc. In general, our work can be used for real-time control of applications that involve tracking of periodic reference inputs like sinusoidal signals, rectangular pulses, etc.

Multi-controller aspect of our work can be utilized to control multiple parts such as wings, tail, etc. of a single micro-aerial vehicle. Moreover, it can also be used to guide multiple agents in a swarm of drones/flock of unmanned aerial surveillance vehicles with one of the agents fitted with the control chip acting as a mobile command center for others. Additionally, control chips can be planted on each agent in a swarm to make the whole system fault-tolerant because there would be continued service in case of malfunction of an agent's chip as chip of another close-by agent can take over the control from the point of malfunction. Till that point, every agent would get benefit of exclusive control from their own chip as these chips are very cheap. Finally, PID accelerator along with a system identification model can be used as a simulator for generating simulation data for training of Machine Learning (ML) and Reinforcement Learning (RL) tasks.

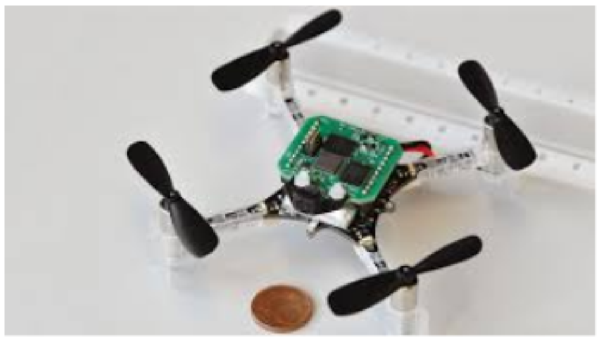
Figure 8.1: Micro-Aerial Vehicle

Source: https://rpg.ifi.uzh.ch/docs/IoT19_Palossi.pdf

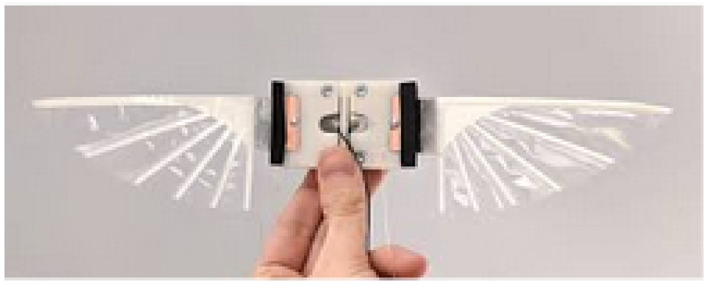
Figure 8.2: Insect-Sized Robot

Source: https://www.science.org/doi/10.1126/scirobotics.abi8189

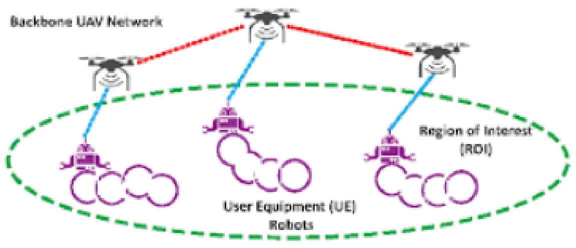

Figure 8.3: Swarm of Micro-Aerial Vehicles

Source: https://www.researchgate.net/publication/356026968 _CoCo_Games_Graphical_Game-Theoretic_Swarm_Control_for_Communication-Aware_Coverage

Chapter 9

Conclusion and Future Work

In this paper, we propose the design of a floating-point PID controller and present its implementation on the low-power, low-cost Lattice iCE40 UP5K FPGA. Through the use of a simplified floating-point operator which is optimized for the chips's DSP blocks along with an effective time-sharing and pipe-lining of these operators, single controller and interleaved double controller versions of our accelerator are able to surpass a comparable 8-bit microprocessor by a factor of over 70× and 120× respectively. On a significant note, our accelerator achieves superiority in terms of power efficiency over all general-purpose processors used for comparison, ranging from 8-bit comparable microprocessors to desktop-grade ones with clock speeds in GHz.

Subsequently, our accelerator is the only option available for our target applications of controlling insect-scale robots and micro-aerial vehicles, as it is the only alternative that meets both the performance and power budget requirements of those applications. In our future work, we intend to explore accelerators for more complex and potential control algorithms that are more suited for intelligent robotics and challenging tasks.

Bibliography

[1] I. A. Abdul Jamil and M. Moghavvemi. Optimization of pid controller tuning method using evolutionary algorithms. In *2021 Innovations in Power and Advanced Computing Technologies (i-PACT)*, pages 1–7, 2021.

[2] M. Aboelaze and M. G. Shehata. Implementation of multiple pid controllers on fpga. In *2015 IEEE International Conference on Electronics, Circuits, and Systems (ICECS)*, pages 446–449, 2015.

[3] V. Aggarwal, M. Mao, and U.-M. O'reilly. A self-tuning analog proportional-integral-derivative (pid) controller. In *First NASA/ESA Conference on Adaptive Hardware and Systems (AHS'06)*, pages 12–19. IEEE, 2006.

[4] M. P. Aghababa. Optimal design of fractional-order pid controller for five bar linkage robot using a new particle swarm optimization algorithm. *Soft Computing*, 20(10):4055–4067, 2016.

[5] M. Andrysco, D. Kohlbrenner, K. Mowery, R. Jhala, S. Lerner, and H. Shacham. On subnormal floating point and abnormal timing. In *2015 IEEE Symposium on Security and Privacy*, pages 623–639. IEEE, 2015.

[6] K. H. Ang, G. Chong, and Y. Li. Pid control system analysis, design, and technology. *IEEE Transactions on Control Systems Technology*, 13(4):559–576, 2005.

[7] D. Bagni and D. Mackay. Floating-point pid controller design with vivado hls and system generator for dsp. *Xilinx Application Note XAPP1163*, 2013.

[8] B. Borovic, A.-q. Liu, D. Popa, H. Cai, and F. Lewis. Open-loop versus closed-loop control of mems devices: Choices and issues. *J. Micromech. Microeng*, 15:1917–1924, 10 2005.

[9] Y. Chan, M. Moallem, and W. Wang. Efficient implementation of pid control algorithm using fpga technology. In *2004 43rd IEEE Conference on Decision and Control (CDC) (IEEE Cat. No.04CH37601)*, volume 5, pages 4885–4890 Vol.5, 2004.

[10] C. Chandni, V. S. Variyar, and K. Guruvayurappan. Vision based closed loop pid controller design and implementation for autonomous car. In *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pages 1928–1933. IEEE, 2017.

[11] R. Dedania and S. Jun. Very low power high-frequency floating point fpga pid controller. In *Proceedings of the* 12^{th} *International Symposium on Highly Efficient Accelerators and Reconfigurable Technologies (HEART)*, 2022.

[12] M. Documentation. Ieee floating point representation. `https://docs.microsoft.com/en-us/cpp/build/ieee-floating-point-representation?view=msvc-170`, 2021.

[13] J. C. Doyle, B. A. Francis, and A. R. Tannenbaum. *Feedback Control Theory*. Prentice Hall Professional Technical Reference, 1991.

[14] M. Electronics. Lattice ice40up5k-uwg30itr1k. `https://www.mouser.com/ProductDetail/Lattice/ICE40UP5K-UWG30ITR1K`, Accessed 2022-04-04.

[15] S. B. Fuller. Four wings: An insect-sized aerial robot with steering ability and payload capacity for autonomy. *IEEE Robotics and Automation Letters*, 4(2):570–577, 2019.

[16] B. Goldberg, R. Zufferey, N. Doshi, E. F. Helbling, G. Whittredge, M. Kovac, and R. J. Wood. Power and control autonomy for high-speed locomotion with an insect-scale legged robot. *IEEE Robotics and Automation Letters*, 3(2):987–993, 2018.

[17] P. Gopi, G. k. r. P V, M. V. Subramanyam, and K. Satyaprasad. Model based tuning of pid controller. *JOurnal of control and instrumentation*, 4:16, 01 2013.

[18] M. Graber. Practical pid tuning guide. `https://tlk-energy.de/blog-en/practical-pid-tuning-guide`, 2021.

[19] J. J. Gude and E. Kahoraho. Kappa-tau type pi tuning rules for specified robust levels: The frequency response method. In *Proceedings of 2012 IEEE 17th International Conference on Emerging Technologies Factory Automation (ETFA 2012)*, pages 1–8, 2012.

[20] V. Iyer, A. Najafi, J. James, S. Fuller, and S. Gollakota. Wireless steerable vision for live insects and insect-scale robots. *Science robotics*, 5(44):eabb0839, 2020.

[21] N. T. Jafferis, E. F. Helbling, M. Karpelson, and R. J. Wood. Untethered flight of an insect-sized flapping-wing microscale aerial vehicle. *Nature*, 570(7762):491–495, 2019.

[22] E. Joseph and O. OlaiyaO. Cohen-coon pid tuning method: A better option to ziegler nichols-pid tuning method. *Computer Engineering and Intelligent Systems*, 9:33–37, 2018.

[23] C. Kadu and C. Patil. Design and implementation of stable pid controller for interacting level control system. *Procedia Computer Science*, 79:737–746, 2016. Proceedings of International Conference on Communication, Computing and Virtualization (ICCCV) 2016.

[24] J.-S. Kim, H.-W. Jeon, and S. Jung. Hardware implementation of nonlinear pid controller with fpga based on floating point operation for 6-dof manipulator robot arm. In *2007 International Conference on Control, Automation and Systems*, pages 1066–1071, 2007.

[25] J. Lima, R. Menotti, J. M. P. Cardoso, and E. Marques. A methodology to design fpga-based pid controllers. In *2006 IEEE International Conference on Systems, Man and Cybernetics*, volume 3, pages 2577–2583, 2006.

[26] K. Lu, W. Zhou, G. Zeng, and W. Du. Design of pid controller based on a self-adaptive state-space predictive functional control using extremal optimization method. *Journal of the Franklin Institute*, 355(5):2197–2220, 2018.

[27] D. R. Lutz and C. N. Hinds. Novel rounding techniques on the neon floating-point pipeline. In *Proc. 39th Asilomar Conf. Signals, Syst. Comput*, pages 1342–1346, 2005.

[28] R. Namba, T. Yamamoto, and M. Kaneda. Robust pid controller and its application. In *1997 IEEE International Conference on Systems, Man, and Cybernetics. Computational Cybernetics and Simulation*, volume 4, pages 3636–3641 vol.4, 1997.

[29] A. L. Salih, M. Moghavvemi, H. A. Mohamed, and K. S. Gaeid. Flight pid controller design for a uav quadrotor. *Scientific research and essays*, 5(23):3660–3667, 2010.

[30] D. Shah, E. Hung, C. Wolf, S. Bazanski, D. Gisselquist, and M. Milanovic. Yosys+ nextpnr: an open source framework from verilog to bitstream for commercial fpgas. In *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 1–4. IEEE, 2019.

[31] R. Su and N. Kermiche. A learning scheme for open-loop and closed-loop control. In *Proceedings IEEE International Symposium on Intelligent Control 1988*, pages 523–528, 1988.

[32] M. V. Subramanyam, K. Satyaprasad, and G. k. r. P V. Study on pid controller design and performance based on tuning techniques. 07 2014.

[33] A. Trimeche, A. Sakly, A. Mtibaa, and M. Benrejeb. Pid controller using fpga technology. In V. D. Yurkevich, editor, *Advances in PID Control*, chapter 14. IntechOpen, Rijeka, 2011.

[34] J. Wadgaonkar, K. Bhole, and P. Singh. Floating point fpga architecture of pid controller. In *2015 International Conference on Industrial Instrumentation and Control (ICIC)*, pages 1259–1263, 2015.

[35] Y. Xu, K. Shuang, S. Jiang, and X. Wu. Fpga implementation of a best-precision fixed-point digital pid controller. In *2009 International Conference on Measuring Technology and Mechatronics Automation*, volume 3, pages 384–387. IEEE, 2009.

[36] J. G. Ziegler and N. B. Nichols. Optimum settings for automatic controllers. *Journal of Dynamic Systems Measurement and Control-transactions of The Asme*, 115:220–222, 1942.

[37] E. W. Zurita-Bustamante, J. Linares-Flores, E. Guzmán-Ramírez, and H. Sira-Ramírez. *FPGA Implementation of PID Controller for the Stabilization of a DC-DC "Buck" Converter*. Chicago: INTECH Publishing, 2012.