The Study of Field Programmable Gate Array Based Servos in Atomic, Molecular and Optical Physics Experiments

by

Shi Jing Yu

B.ASc., The University of British Columbia, 2014

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE

in

The Faculty of Graduate and Postdoctoral Studies

(Engineering Physics)

THE UNIVERSITY OF BRITISH COLUMBIA

(Vancouver)

April 2017

Abstract

The use of Field Programmable Gate Array (FPGA) is becoming increasingly popular in new designs for instrumentation tools. Among them, the FPGA-based servo is emerging as a replacement for the traditional analog servo as a more versatile, automated and remotely controllable alternative. Despite the demonstration of FPGA servos for the control of lasers in the literature, the practical constraints of an FPGA servo have not yet been fully investigated. This work presents an open-source FPGA servo design that is capable of reaching a total signal latency of 200 ns including both conversion delay and computation delay. This work also investigates various limitations inherent in a digital implementation of a servo arising from the computation precision of an Infinite Impulse Response (IIR) filter and the effect that signal quantization has on the transfer function that a digital servo can implement. These technical details are not widely discussed, but are important both for the design and the operation of the FPGA servo. Applying the FPGA servo in an intensity stabilization application allows direct tests of these limitations. In particular, this work compares the performance of the FPGA servo and a high-performance commercial analog servo with a focus on key specifications including the closed-loop bandwidth, noise floor and the resolution of the transfer functions. For closed-loop control scenario with a bandwidth below 1 MHz, we achieve better performance with the FPGA servo than the analog servo through the use of more complex transfer functions including a Proportional and Integral Cubed (PI^3) and a Proportional Integral and Integral (PII) with lag-lead.

Preface

All the work presented in this thesis was conducted at the Quantum Degenerate Gas (QDG) laboratory at the University of British Columbia. The author was heavily involved throughout the development of the FPGA servo and its application in a cold atom experiment, as described by this thesis. The early design of the analog front-end circuit (Section 2.3) and the investigation of the performance of the FPGA servo based on the Development and Education board 3 (DE3) (Section 4.2.6 and Section 4.2.6) was completed upon collaboration with Emma Fajeau; the investigation of noise floor of the two FPGA servos based on the Development and Education board 2 (DE2) and the DE3 (Section 4.1) was completed upon collaboration with Lin Qiao (James) Liu. The remaining work in Chapter 2, 3 and 4, include the hardware and the firmware design of the FPGA servo, the design of the intensity stabilization setup and the servo verification is independent and unpublished work by the author S. Yu.

The study on the relationship between the active intensity stabilization and evaporation efficiency of trapped atoms was carried out independently by the author but was based on a cold atom apparatus that have been built and maintained by members of the QDG lab. (Details on this apparatus can be found at Will Gunton's thesis [53]) The final result that demonstrates the use of passive stability of the trap laser as the final solution to improve evaporation efficiency (Section 5.4.4) is the result of the investigation by multiple members that worked on the apparatus, including Will Gunton, Gene Polovy and Mariusz Semczuk (a visitor at the time).

Table of Contents

List of Tables

List of Figures

Glossary

ECDL	External Cavity Diode Laser
AO	Analog Output
DDR2	Double Data Rate 2
LE	Logic Element
ASIC	Application-Specific Integrated Circuit
MCU	Microcontroller Unit
DC	Direct Current
LNA	Low Noise Amplifier
ADC	Analog to Digital Converter
DAC	Digital to Analog Converter
ROM	Read-Only-Memory
RAM	Random-Access-Memory
GPIO	General-Purpose Input Output
IO	Input Output
PWM	Pulse-Width Modulation
RC	Resistor-Capacitor
PLL	Phase-Locked Loop

HSMC	High Speed Mezzanine Card
ADDA	Analog-Digital Digital-Analog
ADA	Analog-Digital-Analog
SRAM	Static Random-Access Memory
SDRAM	Synchronous Dynamic Random-Access Memory
HPS	Hard Processor System
GBP	Gain-Bandwidth Product
AMO	Atomic, Molecular and Optical
QDG	Quantum Degenerate Gas
FPGA	Field Programmable Gate Array
PC	Personal Computer
AWG	Arbitrary Waveform Generator
DE0	Development and Education board 0
DE2	Development and Education board 2
DE3	Development and Education board 3
DE1-SOC	Development and Education board 1 System on Chip
SOC	System on Chip
PID	Proportional Integral and Derivative
PIID	Proportional double-Integral and Derivative
PD	Proportional and Derivative
PI	Proportional and Integral

HP	High Pass
LP	Low Pass
P	Proportional
I	Integral
D	Derivative
PII	Proportional Integral and Integral
PI^3	Proportional and Integral Cubed
IIR	Infinite Impulse Response
FP	Fixed Point
FIR	Finite Impulse Response
DFI	Direct Form I
DFII	Direct Form II
HDL	Hardware Description Language
FFT	Fast Fourier Transform
LSB	Least Significant Bit
LUT	Look-up Table
PDH	Pound-Drever-Hall
NIST	National Institute of Standards and Technology
AOM	Acoustic Optical Modulator
LC	Inductor-Capacitor
IC	Integrated-Circuit

PCB	Printed-Circuit-Board
TeO_2	Tellurium Dioxide
ESA	Electrical Spectrum Analyzer
NEP	Noise-Equivalent Power
RIN	Relative Intensity Noise
TOF	Time-of-Flight
MOT	Magneto-Optical Trap
VNA	Vector Network Analyzer
SNR	Signal to Noise Ratio
RF	Radio Frequency
VGA	Variable Gain Amplifier
CPU	Central Processing Unit
IO	Input Output
ROM	Read Only Memory
RAM	Random Access Memory
DSP	Digital Signal Processing
GUI	Graphical User Interface
HDL	Hardware Description Language
DDS	Direct Digital Synthesizer
SPI	Serial Peripheral Interface
NOP	No Operation

Acknowledgements

I would like to thank my supervisor prof Kirk Madison for his unconditional support towards my thesis project, prof David Jones for helpful conversations, Emma Fajeau, James Liu for collaborating on the FPGA servo project and William Bowden, Mandana Amiri and Pavel Trochtchanovitch for their insights on laser controllers, FPGA and analog circuits, respectively. I would also like to acknowledge the Altera (now Intel FPGA) University Program for providing free samples of the DE3 and the ADC/DAC that was used to build the DE3 servo.

Thanks to everyone at the lab who are involved in training me on tuning lasers and running the cold atom experiments. They are, my supervisor (of course), prof James Booth, Will Gunton, Gene Polovy, Janelle Dongen and Mariusz Semczuk. Also thanks to my peers, Kahan Dare and Kais Jooya for adding a touch of social aspect to graduate level courses.

I would also like to thank my boyfriend, now fiance and soon husband, Jonathan Fraser and his parents, Cindy and Peter, for their support, and most importantly, my parents, Wen Song (Vincent) Yu and Xiu Ju (Judy) Wang, for all the foresight and sacrifice that went into enabling me with a great education.

Chapter 1

Introduction

The Field Programmable Gate Array (FPGA) is a rapidly developing technology that has enabled a broad array of applications in both basic science and industry. An FPGA chip has the advantage of both speed and flexibility. It can achieve similar speeds to a Microcontroller Unit (MCU) or an Application-Specific Integrated Circuit (ASIC), but unlike the MCU and ASIC, the FPGA digital circuitry is not hardwired but can be programmed and re-programmed. In the infancy of this technology, implementing digital logic produced circuits that ran slower, consumed more space and were less energy efficient than competing technologies, but the advantage of flexibility and a shorter development cycle kept the FPGA in demand [1]. Over the years, significant improvements regarding speed, space consumption and energy consumption have been made. Additionally, more sophisticated routing mechanisms and Digital Signal Processing (DSP) computation resources have also been implemented on FPGAs (a survey on the FPGA technology up to 2008 can be found at [2]). These benefits and advancements have, in turn, been enjoyed by both industrial and the scientific communities.

Like most research communities, the Atomic, Molecular and Optical (AMO) experimental community has a strong demand for versatile instrumentation and computational tools. AMO experiments rely on precise control of lasers including their wavelength, phase, polarization and intensity. To accomplish this, an array of instrumentation tools are needed. The laser servo is an instrument used to form a closed feedback loop control system to make corrections to the laser with an actuator and a detector. (The term servo is in analogy to the use of the term "servo in motion control.)

Traditionally, laser servos, also sometimes referred to as loop-filters, are built from analog components like opamps and impedence networks. The design of an analog servo can be found as early as John Hall's original paper on the Pound-Drever-Hall (PDH) modulation method in

1989 [3]. In this paper, the phase detector is constructed as a mixer and the control mechanism (servo), in the form of a Proportional Integral and Derivative (PID) filter, is built from two cascaded op-amp stages, each implementing an integrator and a differentiator. Traces of John Hall's servo design can be observed in published works with references to the “Jila loop-filter” [4] and commercial stand-alone units based on the Hall design have now become available.

Migration of instrumentation systems from op-amp based analog systems to digital designs naturally follows from the advancement of digital technologies.

The development of an FPGA-based servo was reported in published work as early as 2002 [5]; however, the high cost of the FPGA hardware at the time meant that adoption of this approach was not immediate and widespread use of FPGAs for servos would happen much later. Today, phase locked loops, lock-in detectors, and arbitrary waveform generators have all been realized in the digital domain including, recently, FPGA based implementations [6], [7]. The question of how quantization error affects digital servos has not yet been fully addressed and is one of a main topics of investigation in this work. This question of quantization error is embedded in similar questions regarding the use of digital designs for feedback control: how does a digital servo implemented in an FPGA compare with a high-performance analog servo? Similarly, is there any trade off made with gaining versatility and automated control? The use of an FPGA servo spans may applications including the locking of an External Cavity Diode Laser (ECDL) in atom/ion trapping experiments [8], [9] for spectroscopy [10] and for atom interferometry [11], [12]. The advantage of digital system is its flexibility since, as it is often argued, it requires no soldering to be reconfigured with a different transfer function.

Naturally, the need to understand and quantify the role of quantization error follows the introduction of digital systems. In the implementation of lock-in logic in a digital form, it was clear that the sampling quantization error has an effect on the Signal to Noise Ratio (SNR) of the lock-in detector [13], and thus should be handled with care.

The question of how quantization error affects digital servos has not yet been fully addressed and is one of the main topics of investigation in this work. This question of quantization error is embedded in similar questions regarding the use of digital designs for feedback control: how does a digital servo implemented in an FPGA compare with a high-performance analog servo? Similarly, is there any trade off made with gaining versatility and automated control?

In the attempt to answer some of these questions, this work started with the development of an FPGA servo. The hardware developed by this work focuses on the high performance aspects offered by the FPGA technology. Similar work that focuses on high-performance FPGA servos are relatively few [14] and should be differentiated from digital servo designs with bandwidths <100 kHz, suitable for low bandwidth applications that also need servos in large quantities [6], [15], [16]. This work is also interested in understanding how to push the performance of an FPGA servo beyond what is capable with an analog servo, and this necessitates testing in a closed-loop application. In this work, we chose intensity stabilization as our application for closed-loop studies, and, with it, we achieved a closed-loop bandwidth of 1 MHz. This application allowed us to benchmark the FPGA servo in a high bandwidth setting and to explore some of its advantages over a high-performance analog servo including its ability to fine tune the transfer function to achieve optimal performance. It also allowed us to identify previously unrecognized limitations and areas of particular concern with FPGA servos

This thesis follows the development of our FPGA servo. Each chapter covers topics in the various stages of our development efforts in the following order: hardware design, firmware design, closed-loop performance tests and, finally, the application of the FPGA-based servo in a cold atom experiment. The theoretical background pertinent to the various topics is covered in each chapter. The thesis is meant to acquaint a new designer with all aspects in the development an FPGA servo, and, therefore, it is explicit with describing mistakes made during the development of the FPGA servo. This work may also be of interest to researchers that are interested in using recently developed FPGA servo products, such as Digilock by Toptica (200 ns total signal delay) [17], as this work investigates some of the considerations and limitations one should be aware of when using an FPGA based servo. Moreover, we believe this work provides some insight into the present state and the development bottle necks of FPGA based servo technology.

Chapter 2

FPGA Servo Hardware

The hardware of the FPGA is the development platform for all servo logic and signal processing related elements used in a closed-loop control application. Thus, the hardware plays an indispensable role in determining the performance of an FPGA based servo. The underlying hardware, therefore, affects whether an FPGA servo is suitable to replace existing analog servos in many applications. This chapter focuses on the design and specification of a complete set of FPGA servo hardware elements. Later chapters discuss the development and implementation of the servo logic and signal processing units.

Two design specifications are important for a servo design and thus are recurring throughout this thesis. They are the closed-loop bandwidth and the noise floor. The closed-loop bandwidth of the servo is important because it determines the highest frequency that the servo can correct for. As it is shown in Figure 2.1, a typically control loop consists of a servo/controller that is responsible for generating correction to a quantity under control, an actuator/plant responsible for carrying out the correction and a detector that monitors the quantity under control and provides feedback to the servo. It makes little sense to use a actuator/plant that has a extremely high closed-loop bandwidth of, for example, 10 MHz, if the servo can only respond to disturbances up to 100 kHz. In this scenario, the closed-loop bandwidth of the overall system would be dominated by the servo because the closed-loop bandwidth is determined by the sum of delay in all the component. Hence, the closed-bandwidth of the FPGA servo limits its use with actuators/plants and its applications.

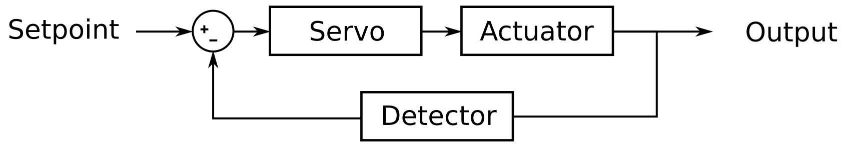


Figure 2.1: A typical control loop consists a servo (also referred to as a controller), an actuator (also referred to as a plant or a process) and a detector (also referred to as the feedback).

The other design specification, the noise floor of the servo, is important because it determines the signal level that the servo can correct at. We are rather familiar with the typical noise sources in an analog circuit such as the input-derived op-amp noise and Johnson noise. One initial concern in the addition of digital system to the servo is the quantization errors at the Analog to Digital Converter (ADC) and the Digital to Analog Converter (DAC). As later experimentation shows, quantization error at the ADC or the DAC is not neccessarily an issue for the noise floor of an FPGA servo, expecially when the signals can be over-sampled and averaged, as the effect is explained at some external sources [18]. It is nevertheless important to keep in mind that the quantization affects a digital servo in areas besides the noise floor, and the details are covered in more detail in Chapter 3.

Based on these general considerations, this work was carried out under the intuitive assumption that a faster and quieter servo is in general more useful. To explore the best FPGA servo that the current technology has to offer, we set out to design the FPGA servo "from scratch", while knowing that the final hardware needs to be competitive against existing analog servos with 10 MHz closed-loop bandwidth and 10 $\mathrm{nV}/\sqrt{\mathrm{Hz}}$ noise floor [19].

The design specification on speed and noise floor is addressed by each component of the servo. The chapter starts with the architectural overview and then describes the particular design challenges in individual sections in detail.

2.1 Architectural Overview

The FPGA servo is a mixed-signal system, as illustrated in Figure 2.2, that can be divided into a digital section, an analog front-end section and a signal conversion section. Reduction of noise and optimization in bandwidth are important in each section although subtly different design objectives are present in each module.

The digital section, mainly consisting of the FPGA, is responsible for high-speed signal processing and the implementation of the servo logic. This work relies on commercially available FPGA platforms to access high-performance FPGAs that would otherwise take too long to design and test on a Printed-Circuit-Board (PCB). It is nevertheless important to ensure that the FPGA can accommodate an on-board MCU and intensive signal processing. The

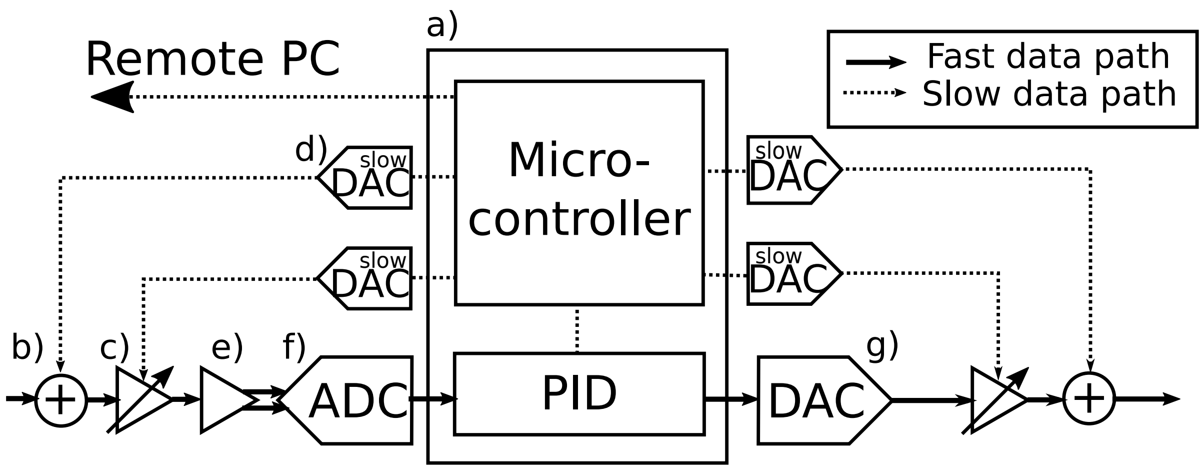


Figure 2.2: The design architecture of the FPGA servo. The servo is composed of the following modules, in the order of appearance in this chapter a) FPGA development board b) offset circuit c) gain circuit d) slow DAC e) ADC driver f) ADC g) DAC

requirement for an on-board MCU, whether it is in the form of a softcore MCU or a hardcore MCU, is advantageous for feature expansion. Additionally, the requirement for computation capability is necessary to implement the servo logic and other DSP tool in the FPGA, which is required to accomplished the primary objective of this work. The hardware description of the FPGA in Section 2.2 provides more details.

The analog front-end is responsible for conditioning the signals at the input/output of the servo to match the voltage range of the ADC and DAC. The design of the analog front-end produced by this work provides extensive flexibility with the use of a variable gain circuit and a variable offset circuit on each of the input/output of the servo. Digital control of these conditioning circuits is provided by an array of slow-updating DAC directly accessible by the FPGA. This design allows the FPGA servo to be remotely controlled in a wide range of control applications.

Finally, the signal conversion section, mainly consisting of the ADC and the DAC, is responsible for converting an analog signal to the digital domain and vice versa. In the signal conversion section, the optimization for closed-loop bandwidth is equivalent to reducing the conversion latency through the ADC and the DAC. In contrast to our initial impression, the latencies through many high-speed ADCs and DACs is much higher than what the clock speeds of these devices suggest.

Development Timeline

Two FPGA servos, based on the Development and Education board 3 (DE3) FPGA platform [20] and the Development and Education board 2 (DE2) FPGA platform [21], are first constructed as prototypes from modular pieces, as shown in Table 2.1. In the prototypes, the analog front-end is designed and tested as a PCB that consistings a variable offset circuit followed by a variable gain circuit. The slow DAC needed to control the variable gain and variable offset circuit is based on an existing 8-channel Analog Output (AO) design by Todd Meyrath [22]. The signal conversion sections of the prototypes are based on commercially available cards that have to be modified to accommodate Direct Current (DC) signals. The prototype servos allow us to verify the closed-loop performance of FPGA servos on two different FPGA platform, the DE2 and the DE3.

Table 2.1: A summary of various servo used in the study.

	FPGA	Conversion	Offset and Gain	Slow DAC
DE3-Prototype	DE3 [20]	AD/DA by Altera [23]	Custom PCB	PCB by Todd Meyrath[22]
DE2-Prototype	DE2 [21]	THDB-ADA by Altera [24]		
DE2-Final	DE2	Custom PCB as shown in Appendix A		

Once the performance of the servos is confirmed, we compiled the electronics design onto a custom daughtercard compatible with the DE2, DE3, a few of the other Altera FPGA development boards. The servo design based on the custom daughtercard is easily replicated and more convenient to use than the prototypes. The final servo design is characterized and provide most of the supporting data in this chapter unless it is otherwise specified.

2.2 FPGA Section

The FPGA section is responsible for high-speed signal processing and the implementation of the servo logic. This section describes the selection criteria of the FPGA in detail and the compatibility of our FPGA daughter card design with other existing FPGA platforms.

2.2.1 Selection Criteria

IO Interface

The Input Output (IO) interfaces of the FPGA development board need to support high-speed parallel signals for use with the ADC and DAC. Two types of IO are popular among the Altera development boards, with one being the High Speed Mezzanine Card (HSMC) high-speed connection and the other one being the 2×20 General-Purpose Input Output (GPIO) connectors. The HSMC connector is suitable for high speed signals and is available on the DE3 board among other high-speed research development boards. The GPIO connector is available on a wider range of development board, including the DE3, DE2, Development and Education board 0 (DE0) and Development and Education board 1 System on Chip (DE1-SOC). Both interfaces seem to be supported by many of Alteras boards including some data acquisition daughter cards. For the FPGA servo prototypes realized in this work, the DE3 platform is paired with the Analog-Digital Digital-Analog (ADDA) card with HSMC interface and the DE2 platform is paired with the Analog-Digital-Analog (ADA) card with the GPIO interface. The two types of interfaces are not directly compatible but conversion can be made using a commercially available converter.

MCU Unit

The second consideration is the support for a MCU in the digital system. It is becoming more common for FPGA designs to accommodate a MCU rather than co-processor as a separate chip. Less urgent tasks can be more quickly implemented in the MCU rather than in the FPGA, so the use of an softcore or a hardcore MCU reduces to development time needed for feature expansion. The more recent FPGA are equipped with Hard Processor System (HPS), but the DE2 and DE3 platform that this work uses do not contain HPS but rather contain sufficient number of Logic Element (LE)s to instantiate a NIOs II softcore MCU provided by Altera.

Besides sufficient number of LEs to instantiate an MCU inside the FPGA, other platform requirements include sufficient run-time memory in the form of Random Access Memory (RAM) and non-volatile storage of both the FPGA configuration binary and the MCU executable. The DE3 is equipped with 1 GB external RAM in the form of Double Data Rate 2 (DDR2) RAM, but the memory bits that are built-in to the FPGA of a total of 5,499 kbit and is sufficient to run the Nios II MCU. The DE2 FPGA doesn't have enough on-chip memory to run the Nios II MCU, so the design uses 512 KB of Static Random-Access Memory (SRAM) or 8 MB of Synchronous Dynamic Random-Access Memory (SDRAM) for run-time memory. In terms of keeping nonvolatile configuration of the FPGA and the MCU on the development board, the EEPROM serial configuration Integrated-Circuit (IC) on each of the platforms, EPCS16 (DE2) and EPCS128 (DE3), are sufficient, as the FPGA image and the MCU image can be combined into the same image and stored onto the same chip (the commands for doing this can be found here [25]).

DSP Units

The final consideration is the computation capability of the FPGA. In the case of the DE3, 384 18-bit by 18-bit multiplier is available on the Stratix III EP3SL150 FPGA on the development platform. However, in the case of the DE2 the number of multiplier is more scarce at 35 18-bit by 18-bit multipliers. This is why, after computation resources needed for a FPGA servo is investigated in Section 3.2.6, the study come to the conclusion that the DE2 isn't the most suitable for an FPGA servo design and rather recommends an FPGA development platform with more computation resources (refer to Section 3.2.6 for more details) for future development. While it is true that regular LEs in an FPGA can be configured as multipliers but are certainly slower due to the additional length of wires need to configure LEs into multipliers compared to a hard-wired multiplier.

2.2.2 Compatibility

Because a servo daughtercard is designed to be easily reproducible for future use, it is important to note its future-proof-ness and compatibility with other FPGA development board. Because Altera's line of FPGA development board has moderate compatibility with one another due to its common expansion header, a daughtercard designed to be compatible with one development board needs very little modification to be used with another development board. This is certainly true with the daughtercard originally designed for the DE2 and that means the DE3 and the more recent DE1-SOC is also compatible with the servo daughter card design by this work. This compatibility matrix makes it is rather effortless to upgrade to a newer FPGA platform from a hardware standpoint as long as Altera continues to support the format of the GPIO header. The servo firmware is also written with future migration in mind to the best of our ability. Information on compatibility of the servo logic with other FPGA can be found in Chapter 3.

2.3 Analog Frontend Section

The analog front-end is responsible for conditioning the signals at the input/output of the servo to match the voltage range of the ADC and the DAC. This section describes the design rationale, IC selection criteria and the characteristics of the analog front-end circuit in detail.

2.3.1 Design Rationale

An FPGA servo needs to work with a wide range of measurement devices and laser controllers, but voltage ranges of the ADC and the DAC are rather limited (0-3.3 V in this design). This motivates the design of the analog frontend. When designing the analog front-end, the variety of measurement devices and actuators, like the photodetector, frequency discriminator, laser current driver and piezo driver can all be simply regarded as an input/output voltage range, with common voltage ranges like 0-10V, ±10V, 0-3.3V and 0-5V.

Tunable gain and offset is useful when working with different voltage range and signal sizes. A tunable gain circuit can be used match the amplitude of the measurement signal to the ADC or the DAC voltage range to the laser controller. The tunable offset can be used to remove any DC component in the signal that contains little information about the signal but takes up valuable operation range of the ADC or the DAC. The offset circuit at the input of the servo also controls the setpoint that the servo locks to. Modulation of this input offset signal adds modulation capability to a closed-loop system.

Remote tunability of the analog front-end is accomplished by the use of slow DAC to control the offset and the gain at the input and the output of the servo. This tunablity is complimentary to the ability for the FPGA servo to implement various transfer functions (Chapter 3).

2.3.2 IC Selection Considerations

The choice of three ICs is specially important for the performance of the analog frontend. The ICs are the op-amp for the offset circuit, the amplifier used for the gain circuit and the slow DAC used for digitally controlling the offset and the gain. This section covers the selection criteria for each chip and some alternative designs that are rejected in the design process.

Variable Offset Stage

The variable offset circuit is simply an op-amp configured in the adder configuration, as shown in Figure 2.3. The adder configuration does not restrict the selection of op-amps, but requirement for speed and noise does. The design uses the AD829 for a bandwidth of 12 0MHz, a slew rate of 230 V/μs and low input noise of 1.7 nV/$\sqrt{\text{Hz}}$, as indicated by its datasheet. The more commonly used LM741 and OP37 were briefly considered although their were finally rejected due to to their lower slew rate, 0.5 V/μs and 17 V/μs respectively, and rather high input noise of 20 nV/$\sqrt{\text{Hz}}$ in the case of the LM741.

The AD829 opamp requires external compensation in the picofarad range to operate at full speed. It is therefore not wise to prototype this circuit on a breadboard/stripboard because the parasitic capacitance in a breadboard/stripboard can affect the slew rate of the op-amp. The schematics shown in Figure 2.3 shows the configuration for both voltage compensation (C307) and current compensation (C306), as the compensation schemes are defined by the AD829 datasheet. The external compensation scheme is finally decided to be the current compensation, (C306) after the AD829 is tested on a PCB.

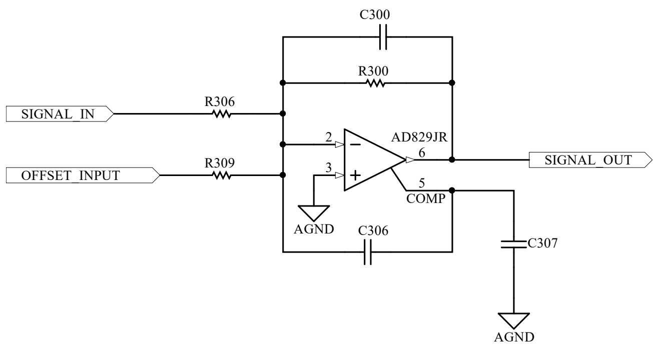


Figure 2.3: Schematics of the variable offset circuit. Both the current compensation scheme (C306) and the voltage compensation scheme (C307) are shown.

Variable Gain Stage

The variable gain stages are needed to scale the input/output signal of the servo to fit in the operating range of the the ADC or the DAC. A few implementation schemes for a variable gain stage are available, so some work went into searching for a suitable Variable Gain Amplifier (VGA) design.

Potential designs for the variable gain stage include the use of digital potentiometers or rheostats, which are resistors that can be digitally programmed. The limitation of this technology is resolution and bandwidth. The chipsets that represent the state-of-the-art technology are the Analog AD514x and AD512x series, with bandwidth of 3 MHz and resolution of 8 bit. It is possible to mitigate the lack of resolution with a network of digital potentiometers, but bandwidth limitation can not be overcome and that leads to the rejection of this design scheme.

Other rejected designs are VGAs with a discrete selection of gain and ADCs packaged with VGAs in single ICs. The former can be troublesome to tune as it does not provide a continuous range of gain. The later is rejected for its obscurity and a lack of high speed options.

Within the selection of VGA with continous range of tunable, we found it important to test the VGA to make sure that the IC is rated for DC. One unsuccessful example is the use of the AD8367, a VGA that represents a class of multi-hundred MHz VGAs that are not rated for DC signals. In the case of the AD8367, the gain is constant over two distinct frequency regions but has a change of 3 dB in gain between the two frequency regions. This renders the IC useless for closed-loop servo applications. Accordingly, we proceeded by carefully selecting and testing high speed VGAs from the selection table provided by manufacturer to avoid similar problems [26].

After surveying the whole design space, this work came up with the single variable gain op-amp AD603 that has 40 dB of tunability, a constant passband of DC - 91 MHz and a tunable gain of 40 dB. The design of the variable gain stage is very simple as shown in Figure 2.4. Besides the signal input to and output from the amplifier, the VGA takes a control input of ± 0.5 V. A voltage divider is installed to accept a voltage input range of ± 10 V. The voltage divider also doubles as a low-pass filter and is used to suppress noise in the slow-DAC at high frequencies.

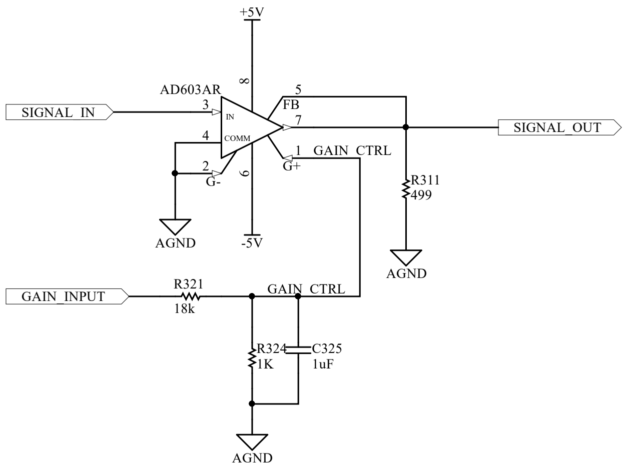


Figure 2.4: Schematics of the variable gain circuit.

Slow DAC Selection

The use of the slow DAC in controlling the gain and offset of the analog front-end allows the servo to be tuned digitally. The speed requirement on the slow DAC is rather relaxed compared to the ADC and the DAC that are directly responsible for the servo action. This allows the communication interface of the slow DAC to be serial to reduce the total IO usage on the FPGA.

The design consideration behind the slow DAC is an economic one because a total of 8 DAC channels is needed for the dual-channelled FPGA servo (one tunable offset and one tunable gain on each of the input/output of a servo channel).

The implementation of the 8-channel DAC is different in the prototypes and the final design of the servo daughter card. In the case of the servo prototypes, the slow DAC channels are controlled by a separate FPGA platform that interfaces with the 8-channel DAC card designed by Todd Meyrath [22]. The DAC card is based on 2 DAC7744 ICs and exposes a parallel interface and it is controlled by an USB-to-parallel converter that this work commissioned on another FPGA platform. A total of two FPGAs are used in constructing each servo prototype that we used to test out understanding about the FPGA servo. The construction is unnecessarily bulky and that motivated the design of the servo daughtercard.

In the case of the servo daughter card, the number of unused IOs on the GPIO header is small. This necessitates serial communication and sharing of signals between the slow DAC

ICs. The slow DAC IC used in the daughtercard design is a serialized version of the IC used in Todd's design, with ±10 V output range and a noise level of 60 nV/$\sqrt{\text{Hz}}$. This means that measurements made on the prototype servos apply to the daughtercard. This work also considered the use of AD5669, an 8-channel DAC, but the lower SNR of the IC (120 nV/$\sqrt{\text{Hz}}$ with an output range of 0-5 V) degrades the performance of the servo, so it is subsequently rejected.

Another feature of the slow DAC section is the synchronous latching of the control parameters, enabled the connection between the latching signals on the slow DACs and the FPGA. We speculate that updating all servo parameters at the same time causes less disturbance in the servo loop and can be very useful in feature developments in the future.

2.3.3 Noise

Offset Circuit

The work in reducing noise in the analog front-end circuit does not stop at the use of low noise amplifiers. A noise model for the offset circuit is helpful in identifying and eliminating noise sources.

The noise model for the offset circuit is based on the adder configuration, as shown in Figure 2.5. Noise sources include the noise in each branch of the adder and the input derived noise of the op-amp. The thermal noise of the feedback resistors (in the range of 1-10 kΩ which is responsible for 4-12 nV/$\sqrt{\text{Hz}}$ of thermal noise) is neglected which lead to a less accurate but still sufficient model. It is to note that the noise floor of the measurement can limit the quality of the noise measurement. The SR780 Vector Network Analyzer (VNA) used the study has a minimum noise floor of 10 nV/$\sqrt{\text{Hz}}$ but can have a higher noise floor when the sampling window is increased to measure a larger input signal.

The dominating sources of noise in the offset circuit is the offset voltage from the slow DAC, rather than the op-amp input noise. Reduction of noise from the slow DAC improves the noise floor of the offset circuit substantially. In the case of the offset circuit at the input of the servo, the slow DAC voltage of ±10 V is reduced to a range of ±0.6 V with a 19:1 voltage divider. This should in theory reduce noise of the slow DAC from 60 nV/$\sqrt{\text{Hz}}$ to 3 nV/$\sqrt{\text{Hz}}$. The result

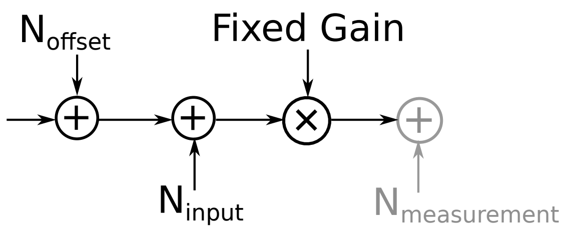


Figure 2.5: Noise model of the offset circuit. The input noise at the op-amp N_{input} is small at 1.7 nV/$\sqrt{\text{Hz}}$. The noise from the offset voltage varies. For the input offset circuit, N_{offset} is 3 nV/$\sqrt{\text{Hz}}$ and the noise level of the external offset. For the output offset circuit, N_{offset} is 60 nV/$\sqrt{\text{Hz}}$. $N_{\text{measurement}}$ varies between 10-100 nV/$\sqrt{\text{Hz}}$ depending on signal amplitude.

of this improvement is illustrated in Figure 2.6 where noise level after improvement is below the noise floor of the measurement.

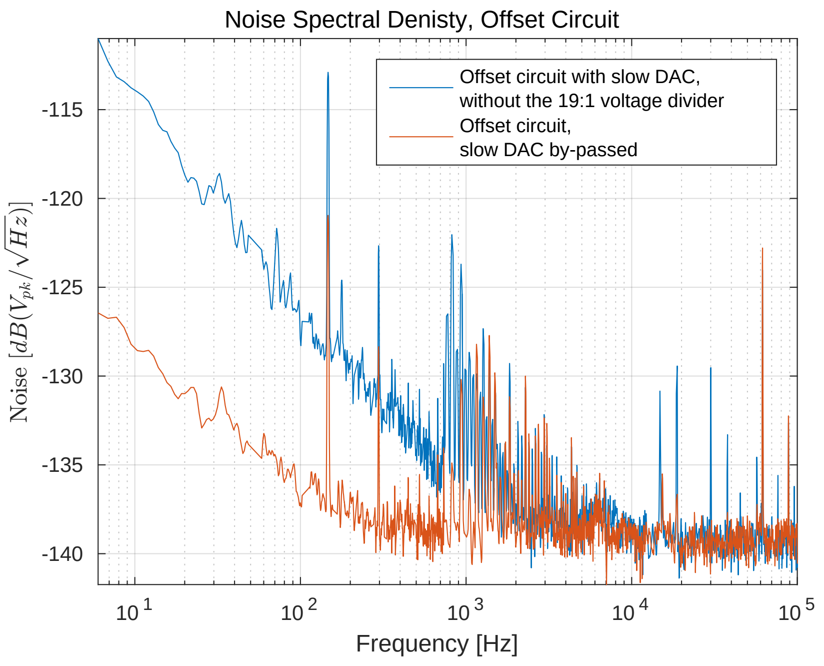


Figure 2.6: Noise level of the adder circuit highly depends on noise in the offset voltage, as it is shown in the change in noise floor in frequencies <1 kHz. In frequencies >1 kHz the noise floor is limited by the measurement noise floor.

In the absent of this input offset, the servo locks to 0 V. The final design of the servo allows the slow DAC to be turned off and it allows the use of an external offset. The user must then be careful with selecting the offset source, as any noise in the offset voltage can degrade the noise performance of the offset circuit.

In the case of the offset circuit at the output of the servo, the range of ±10 V in offset is needed to accommodate a wide range of laser applications. The divider is then absent in the output offset circuit. The decision is justified by the ability of a servo to suppress noise at its output, as it is demonstrated in Chapter 4.

Gain Circuit

A noise model is also built for the VGA to investigate noise sources in the variable gain circuit. As shown in Figure 2.7, noise is present at the signal input and the signal output of the amplifier and at the control input for the gain. The input noise of the amplifier is small at 1.3 nV/$\sqrt{\text{Hz}}$. The noise in the control signal is originally 60 nV/$\sqrt{\text{Hz}}$ from the slow DAC but is reduced after a voltage divider of 19:1. The control input is responsible for an error in gain of 120 ndB/$\sqrt{\text{Hz}}$. In the worse case scenario when the input signal or the output signal saturate at either rail of the AD603 (maximum output range of ±3 V), the error in gain converts to a theoretical maximum noise level of 41 nV/$\sqrt{\text{Hz}}$.

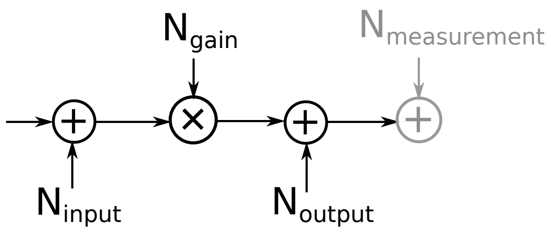


Figure 2.7: Noise model of the variable gain circuit. The input noise of the VGA is small at 1.3 nV/$\sqrt{\text{Hz}}$. The noise at the control input(N_{gain}) is 3 nV/$\sqrt{\text{Hz}}$ in voltage, which correspond to an error of 120 ndB/$\sqrt{\text{Hz}}$ in gain.

The noise source at the output of the AD603 is a significant source of noise as it is demonstrated by the set data shown in Figure 2.8. In the measurement, the input to the AD603 is terminated to ground and the gain is set to three settings that are 10 dB apart. The noise floor in frequencies >200 Hz is not dependent on the gain, and this suggests that the noise source is at the output of the amplifier. The possibility of the limiting factor being the measurement noise is carefully eliminated, which lead to the conclusion of the presence of about 80 nV/$\sqrt{\text{Hz}}$ noise at the output of the AD603 amplifier. We decided that this noise does not necessarily degrade the performance of the servo, because the noise is similar in amplitude as the noise level in the ADC region. We simply move on to different part the the servo design with the knowledge of this issue.

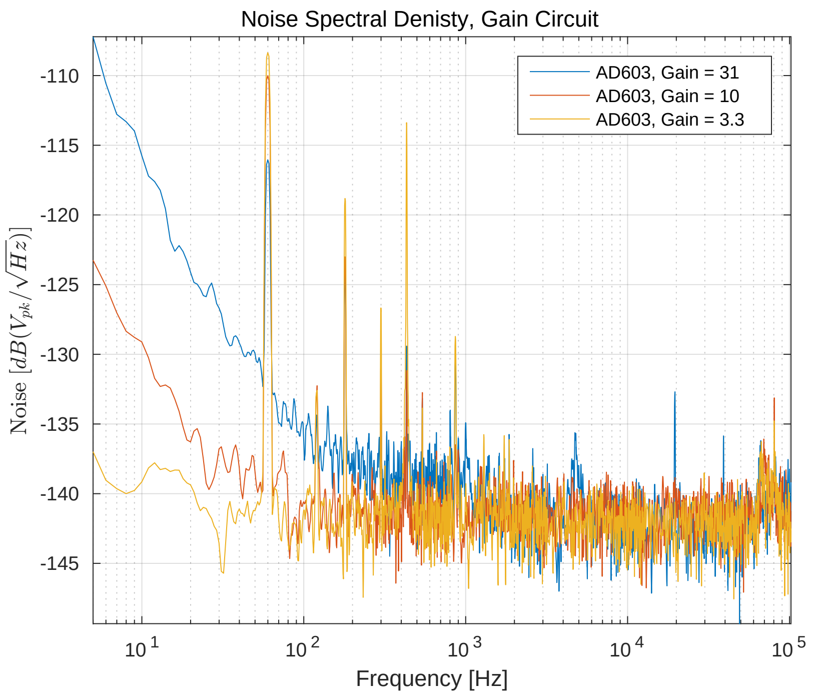


Figure 2.8: Noise at the output of the AD603 at various gain setting, with the input of the AD603 terminated to ground. The VNA input range is set to -44 dBV. In frequencies < 200 Hz, the effect of the input noise of the AD603 can be observed to vary with gain. In frequencies >200 Hz, the output noise of the AD603 can be observed as it is independent of the gain at a level of about -142 dBV/$\sqrt{\text{Hz}}$ (this is equivalent to 79.4 nV/$\sqrt{\text{Hz}}$).

2.3.4 Bandwidth

Offset Circuit

The AD829 opamp needs to be configured with external capacitance in the pidofarad range to operate at full speed. The optimal compensation scheme for a fixed gain of 1 or 4 is investigated in the first PCB design of the analog frontend design, which allows both external compensation schemes (current and voltage) to be tested. After the compensation scheme is decided, the speed of the offset circuit in the final design of the servo is characterized over a range of external capacitances. The purpose of the test is to optimize the speed of the offset circuit in the final servo design and to ensure that similar speed can be reproduced over the tolerance variance of the capacitors. The result of this test is shown in Figure 2.9.

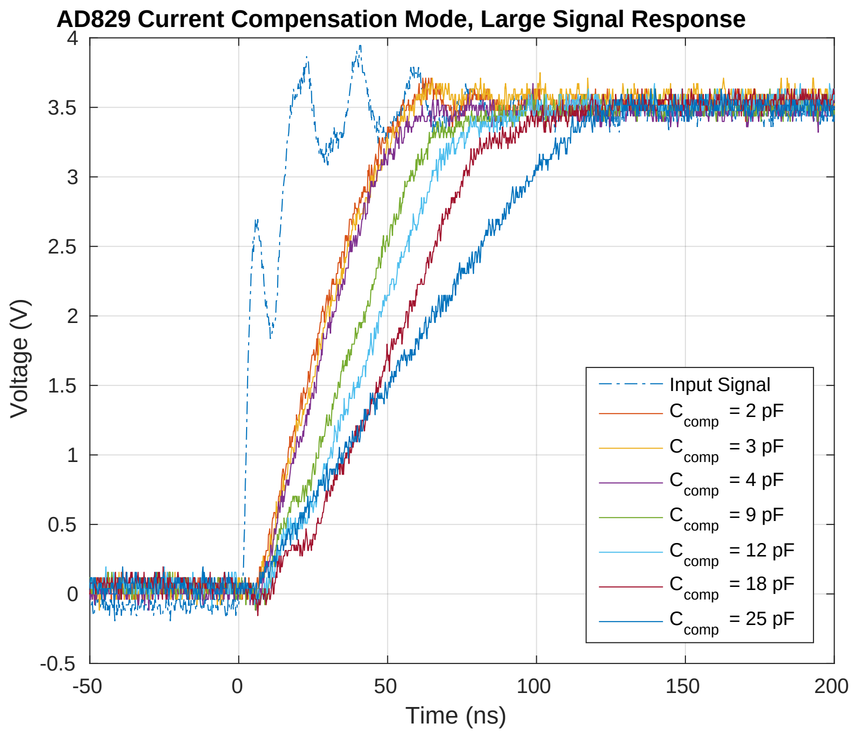


Figure 2.9: AD829 bandwidth as affected by external compensation capacitance in the current compensation mode. The optimal capacitance is 2-4 pF for the negative feedback configuration of AD829 with a gain of 4, feedback resistance of 4 kΩ and feedback capacitance of 1 pF, which agrees with manufacturer's recommendation. It is important to note that the capacitance in the feedback network affects the optimal compensation capacitance.

The input to the circuit is generated from a digital output of an FPGA, which approximates a step function with the exception of some ripples due to impedance mismatching at the terminals. The response at the output of the opamp is measured with an oscilloscope and the delay through coaxial cables to and from the circuit is calibrated away. A set of step response with variation in compensation capacitance (current compensation) is shown in Figure 2.9. The use of large signal response helps to reveal the slew rate of the op-amp. It is worth noting that all step response shown is the current compensation mode as we are not able to reproduce similar speed in the voltage compensation configure as described by the AD829 datasheet in any iteration of the analog front-end design.

When a gain of 4 and a feedback capacitance of 1 pF is used, the optimal compensation capacitance is 2-4 pF. The result agrees with manufacturer's recommendation. In the data, the variation of 2-4 pF in capacitance results in a very small variation in slew rate. This suggests that the bandwidth performance in a larger production of the servo daughtercard will not vary much from the results obtained by this work.

Gain Circuit

For designers who are familiar with the constant Gain-Bandwidth Product (GBP) of an op-amp, the ability for the AD603 to maintain a constant bandwidth of 90 MHz over a 40 dB range of gain can be surprising. This unique property of the AD603 is accomplished using high bandwidth digital potentiometer followed by an Low Noise Amplifier (LNA). The digital potentiometer is responsible for varying the gain by a range of 40dB when the gain through the LNA remains fixed. The gain in the LNA is determined by an externally configured feedback loop where in this design is set to the lowest gain setting to produce the highest bandwidth (DC-90 MHz).

The measurement of bandwidth shown in Figure 2.10 is accomplished with step input and an oscilloscope as described in the step response measurement of the offset circuit. The response of the AD603 over different gains confirms that a high bandwidth is maintain over the entire range of gain settings. The small amount of variation in response time with a spread of roughly 1 ns is likely due to the presence of ripples in the trigger signal and the shot-to-shot variation in the shape of the ripples.

2.4 Conversion Section

The conversion section is the bridge between the analog front-end and the FPGA in the design of an FPGA servo. It is also a main source of delay in the FPGA servo. This section describes the selection criteria for the various ICs that fulfill the function of the conversion and their overall characteristics, both in terms of noise and latency.

2.4.1 ADC and DAC Selections

The key selection criteria for the ADC and the DAC are signal latency and resolution. These two criteria are almost the opposite of each other since a high resolution ADC or DAC typically

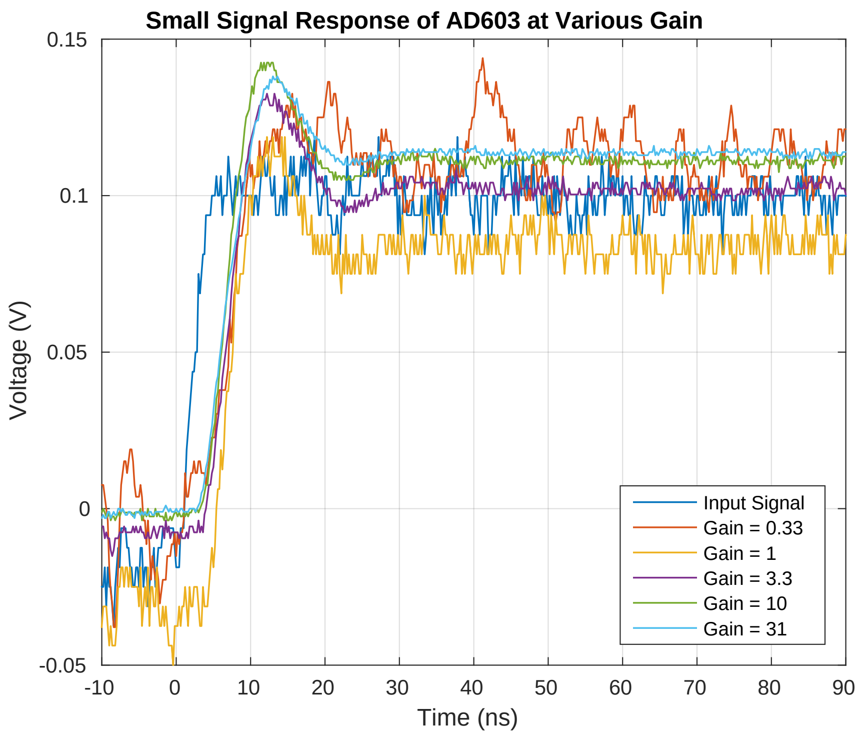


Figure 2.10: AD603 has a constant bandwidth over the whole range of gain, tested between -10 dB and 30 dB at 10 dB intervals.

has a high latency and and a low latency ADC or DAC typically has a smaller bit depth. This trend can be observed from the technology landscape of ADCs made by Analog Devices, as shown in Figure 2.11. The available ADCs requires choosing a resolution of 16-bit or lower to keep the sampling rate above 10 MHz, a must for a digital servo to compete with traditional analog servo technology with close-loop bandwidth of >10 MHz (laser servo D125 by Vescent). It is worth noting that the absolute delay through the ADC is not necessarily proportional to the inverse of the sampling rate since multiple clock cycles are often needed to completely convert an analog signal into the digital domain. For example, the AD9254 ADC used in DE3 prototype has a maximum clock speed of 150 MHz, but its 12 clock cycles of clock delay makes it less favourable than the ADC LTC2195 used in the National Institute of Standards and Technology (NIST) servo design [14] with a lower clock speed of 125 MHz but also a lower pipeline delay of 7 cycles. Unfortunately, the number of clock cycles required by each ADC or DAC to complete the conversion is often not part of the search criteria in search engines for ADCs and DACs. This makes it rather cumbersome to find suitable ADC and DAC to be used in an FPGA servo.

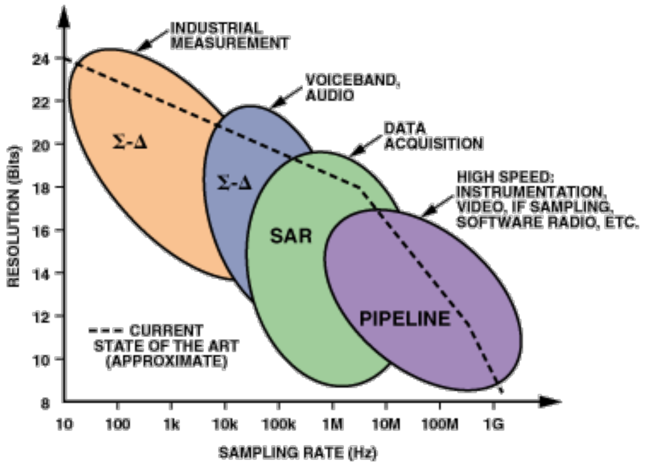


Figure 2.11: The technology landscape of ADCs, provided by Analog Devices

Since the latency through the conversion section is an important characteristic of a digital servo, a survey is conducted on suitable ADCs and DACs for use in a servo design. Part of the survey is shown in Table 2.2, where ADCs and DACs used in three FPGA servo platforms are listed, including a recently published work on high-bandwidth FPGA servo by NIST [14]. The minimum conversion latencies in each of the FPGA servos, in the order of the DE3 servo prototype, the DE2 servo prototype and the recently published servo design is respectively 95 ns, 108 ns and 70 ns.

The use of commercially available ADC and DAC daughtercards compatible with the DE2 and the DE3 in designing the FPGA servos greatly reduces the development time. This also means that the selection of ADC and DAC is limited to two daughtercards and their respective ADC and DAC options. It is worth noting that out of all the ADCs and DACs listed in Table 2.2, the ADC/DAC combination that would produce the minimum conversion delay is the ADC LTC2193 (NIST paper) and the DAC AD9766 (DE2 platform) with a total delay of 56 ns. Reduction of delay in the conversion section can be worthwhile as it lead to an increase of servo closed-loop bandwidth.

The Oversampling Effect

A higher clock speed has other advantages, although it may not indicate the the total conversion delay through an IC. With the appropriate DSP tools, precise control of the timing of a signal

Table 2.2: Viable ADC and DAC options, their pipeline delay, resolution, etc

	IC	Resolution	Type	Max Clock (MHz)	Latency (clk)	Total Pipeline Delay (ns)	Note
ADC	AD9254	14	SAR	150	12	80	DE3 Servo Prototype
	AD9248	14	Pipeline	65	7	108	DE2 Prototype Final Design
	LTC2195	16		125	7	56	NIST paper [14]
DAC	DAC5672	14	-	275	4	15	DE3 Servo Prototype
	AD9767	14	-	125	0	0	DE2 Prototype Final Design
	AD9783	16	-	500	7	14	NIST paper [14]

can be traded off for signal resolution and vice versa with the use of oversampling. In analog-to-digital conversion, oversampling and averaging can lead to higher accuracy in the data as long as sufficient amount of white noise is present in the input signal. Similarly, in digital-to-analog conversion, Pulse-Width Modulation (PWM) is the use of a 1-bit DAC (a digital output) with high-accuracy timing control to produce a high resolution analog output. Additional reading may be very helpful on this subject and this reference provides the reasoning on why for every 4 times in over-sampling the resolution of the sampled signal can be improved by a factor of 2 [18].

2.4.2 Single-Ended to Differential Conversion

ADCs and DACs with high resolution often have differential analog inputs and outputs for better noise immunity. Conversion between single-ended signals and differential signals needs to be carefully handled to prevent signal contamination.

The differential DAC outputs are relatively straightforward to convert to a single-ended signal. The key is to make sure that the DAC differential outputs have matched load (25 Ω resistors) to ground. The "+" signal of the differential pair can be used as an single-ended signal and the "$-$" signal can be disregarded. This is the recommended practice by the manufacturer for DC-coupling the differential DAC outputs [27].

On the other hand, an amplifier with differential output is required to interface with the differential inputs of an ADC. The ADC driver AD8137 is used and configured according to the schematics shown in Figure 2.12. The ADC driver is also used as the anti-aliasing filter before the ADC to suppress noise components that is higher in frequency than the Nyquist frequency (1/2 of the sampling frequency) of the ADC. The corner frequency of the anti-aliasing filter is determined by passive components, C301 and R301, in the feedback path of the ADC driver. They form a low pass filter around 25 MHz. It is to note that the exact Resistor-Capacitor (RC) time constant is often changed to match the Nyquist frequency of the system in the prototyping stage of the servo. Another low-pass filter formed by R305 and C302 at the output of the ADC driver is also used to suppress noise at high frequency, but the cut-off frequency of this low-pass filter is chosen to be a few times larger than that of the anti-aliasing filter to prevent additional decrease in loop bandwidth.

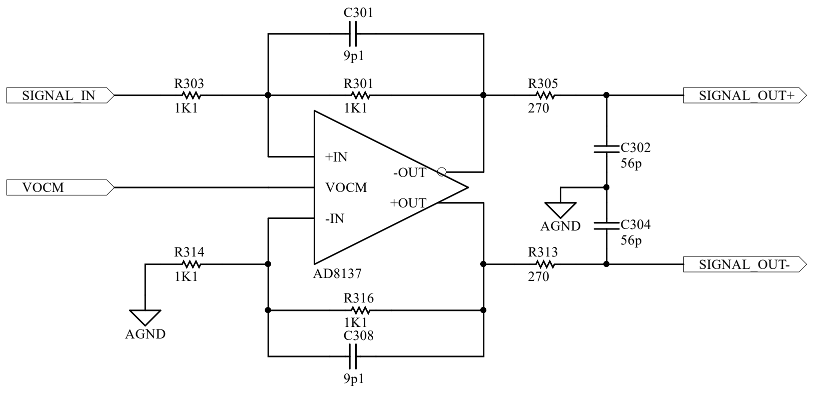


Figure 2.12: Schematics of the ADC driver.

2.4.3 Latency

The latency through the conversion section needs to be carefully optimized for a high loop bandwidth even when ICs with low latency are used. This section describes the techniques used in identifying the causes of delay and how to eliminate them.

Delay can be categorized into propagation delay and pipeline delay. It is helpful to distinguish between the two, because eliminating each type of delay takes a different strategy.

Pipeline delay can be used as an umbrella term for delays that scale with the clock frequency. The ADC, the DAC and the FPGA all have a moderate amount of pipeline delay. It is often helpful to measure the number of clock cycles in the pipeline delay in a system by making latency measurement at a few different clock frequencies. To ensure that the ADC and the DAC can operate at their rated maximum frequencies, any infrastructure carrying high-speed signals to and from the ICs also need to be properly rated. This ultimately motivated the use of commercially available FPGA platforms and ADC/DAC daughtercards in the prototyping stage of the servo. In the design of the custom servo daughtercard commissioned by the end of this thesis, signals in each 14-bit wide digital bus are carefully length-matched to the respective clock and latch signals, with the Altium trace length matching tool.

The master clock frequency of the servo also affects the total pipeline delay. One design constraint is the maximum clock speed of the ADC and the DAC, which are often not the same, as shown in Table 2.2. Clocking the ADC, the FPGA and the DAC at multiples of a master clock frequency can produce a better result than clocking them all at the maximum frequency of the slowest IC. For example, the DE3 servo prototype produces a lower delay of 97 ns when the ADC and the DAC are each clocked at 150 MHz and 225 MHz than if both ICs are clocked at 150 MHz. A second constraint is the capacity of the Phase-Locked Loop (PLL) in the FPGA because a PLL has a defined set of clock multipliers and dividers. In the case of the DE2, 65 MHz cannot be derived from a 50 MHz source. This prevents the DE2 servo from achieving the theoretical minimum conversion delay of 108 ns.

The other type of delay, the propagation delay, can be used as an umbrella term for delays that do not scale with the clock speed. Both analog signals and, somewhat surprisingly, digital signals can produce propagation delay. The typical method to reduce propagation delay is to reduce any unnecessary internal or external signal paths. This applies to coaxial cables, PCB traces and etc although the propagation speed in each median is different from one another. The delays of a few ns can quickly add up to 10-20 ns even after extensive optimization.

2.4.4 Bandwidth

The bandwidth of ADC driver dominates the bandwidth of this section as its corner frequency is deliberately set to be low to act as the anti-aliasing filter. The optimal corner frequency of this filter changes as the clock frequency of ADC changes, so modification to this corner frequency is typically made after the clock frequency of the ADC is fixed.

2.4.5 Noise

Compared to the noise level of < 10 nV/$\sqrt{\text{Hz}}$ in the analog front-end section, the conversion section is noisier at the 100 nV/$\sqrt{\text{Hz}}$ level and it is susceptible to noise spikes introduced by poor grounding practices. In this work, the noise level in the conversion section is measured at the DAC output with a low frequency spectrum analyser SR780, and the conversion section is configured in three different ways to reveal information about the circuit.

The first configuration, as shown in the inset of Figure 2.13, establishes the noise level at the output of the DAC at 32 nV/$\sqrt{\text{Hz}}$ by holding the DAC output at a constant value. The second configuration measures the noise floor of the ADC with terminated inputs, by configuring the FPGA in the "follower" mode (where the DAC relays the ADC data). This measurement reveals that the ADC noise floor is higher than the DAC and that some frequency spikes appear between 60 Hz and 10 kHz, as shown in Figure 2.13. The frequency spikes worsens in the final configuration, where the ADC is driven by the ADC driver with a terminated input. The frequency spikes can be seen in many of the noise spectrum data throughout this work. We investigated in this issue extensively.

Since the amplitude of these frequency spikes observed in the signal conversion section can potentially make the use of these FPGA undesirable, we thoroughly investigated in the matter and developed a few techniques to eliminate the spikes. First, it is often helpful to improve the ground connection of the circuit. That means to eliminate ground loops and to reduce the impedance to ground by, for example, connecting the optical table to the power supply chassis. Secondly, we find that the ground setting on the spectrum analyser used to make the measurement greatly affects the appearance of these spikes. Setting the ground configuration on the spectrum analyser to "floating" typically helps. Finally, the output of loop measurement of a closed-loop controlled quality almost never show these spikes. This observation strengthens our speculation that the spikes are associated with the grounding configuration, as the out-of-loop measurement provides electrical isolation between the servo circuit and the measurement circuit. Although we often use these techniques to eliminate the frequency spikes, this thesis did not go into great length to retake all existing data. So some inconsistency in the appearance of these spike is present in this chapter and Chapter 4.

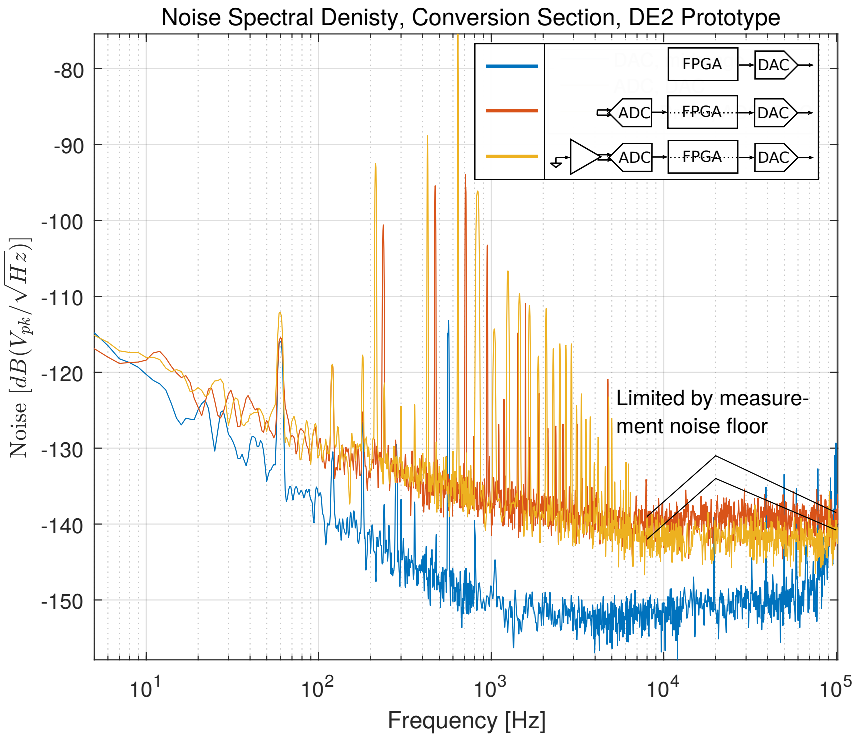


Figure 2.13: Investigation in the noise of signal conversion (ADC/DAC) section of the DE2 servo prototype.

This work takes the indirect approach to identify the noise sources in the ADC and the ADC driver. The direct approach would be to make Fast Fourier Transform (FFT) analysis of the ADC data directly. This can be accomplished either by the FPGA or off-line in a computer. The development of an FFT module in the FPGA can add a useful tool in the FPGA servo.

2.5 Servo Transfer Function

The overall servo transfer function is characterized with a network analyser. As illustrated the transfer function in Figure 2.14, the FPGA servo has constant gain at low frequency and a high-frequency drop off at 10-20 MHz. The phase response is also constant at low frequencies and phase response of a constant latency of about 200 ns becomes visible at >100 kHz.

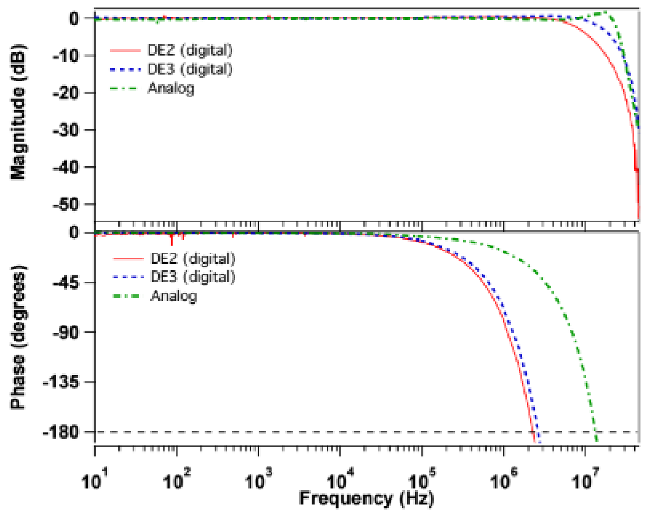


Figure 2.14: Servo transfer function.

The closed-loop bandwidth of the servo is at the 180 degree phase shift of the servo. In the comparison between the FPGA servos and the analog servo in Figure 2.14, the bandwidth of the FPGA servos are significantly lower at 2-3 MHz compared to the 10 MHz of the analog servos. The difference in closed-loop performance is visible when the total latency of the plant and the detector is very small. Despite a slightly reduced bandwidth, the FPGA servo is still useful in closed-loop applications that involve Acoustic Optical Modulator (AOM), fast piezos and current modulation of lasers, where the closed-loop bandwidth of the plant is typically <500 kHz.

Chapter 3

FPGA Servo Firmware

The FPGA servo design is motivated by the opportunity to replace an analog servo controller based on op-amps with a digital servo that can be controlled and monitored remotely. In order to replace analog servos already in use in AMO experiments, the digital FPGA servo needs to achieve a similar transfer function and performance as the analog servo. The bandwidth and noise floor is addressed by the previous chapter on servo hardware. This chapter focuses on the FPGA firmware with the goal of maintaining a good stability and noise level.

The servo logic is implemented in the form of a Infinite Impulse Response (IIR) filter similar to what was described in various published work [12], [14]. The work of this thesis is aimed at a demonstration of viability and the exploration of the practical constraints of implementing servo logic in the form of an IIR filter. The issues addressed in this work include the precision with which the poles and zeros can be placed, the computation resources needed in an FPGA to implement IIR filters of various precision, and the effects of computation delay. This work also discusses the details of the verification of an IIR filter.

This work also looks into various ways of implementing an Arbitrary Waveform Generator (AWG) as an added feature to the FPGA servo. The FPGA servo hardware can be used as a high-speed waveform generator without needing any circuit modification, with similar data rate (50 MHz - 135 MHz) as the DS345 function generator (30 MHz).

The implementation of other useful servo tools such as a lock-in detector input stage, an auto-locking algorithm and a transfer function identification system are discussed at the end of this chapter as potential future expansion features to the servo. Besides the proposed feature expansion, the FPGA servo hardware is a high performance platform and many more features can be added without hardware changes.

3.1 Overview

In realizing the FPGA servo as a toolbox for control applications in AMO experiments, the work produces a number of System on Chip (SOC) solutions applicable to common experimental applications. This section covers the design of the SOCs and the organization of the FPGA firmware. Here with the word “firmware” we refer to all designs that reside in the FPGA including both the configurable circuitry and the assembled C program inside the Central Processing Unit (CPU) built into the FPGA. This is different from the vocabulary used in white pages by FPGA manufacturers such as Altera, where FPGA circuitry is referred to as hardware and the assembled C program is referred to as software. In this work, however, the term hardware is reserved for the hardware design described in Chapter 2; the term software is reserved for programs on the Personal Computer (PC) which is discussed in Appendix D.

This thesis investigates the use of two FPGA platforms, the DE3 and the DE2, in designing laser servos. The DE3 is a high-performance FPGA and the DE2 is an economic option, both offered by Altera. For both FPGA platforms, this work included SOC designs with the structure illustrated in Figure 3.1. Each SOC was composed of a CPU in the form of a Nios II processor and several specialized signal processing units like the IIR filter, the Finite Impulse Response (FIR) filter and the AWG. The work conducted in this thesis adheres to the principle of modularity so it aims to produce individually testable modules. The advantage of this design is that it produces not only well tested code but also designs that can be rather effortlessly transferred into other applications. The following sections discuss the design of servo logic in the form of the IIR filter (Section 3.2), implementation of the FIR filter (Section 3.2.9) and the AWG (Section 3.3) in detail. These modules share a common control interface that includes a clock, an asynchronous reset, a parallel bidirectional data port, an address port for accessing internal registers and simple read/write signals. This control interface is compatible with not only the master bus of the Nios II processor controls (the Avalon bus) but also the AXI bus of the hardcore ARM processor embedded into the DE1-SOC FPGA among other CPU bus standards.

The softcore CPU, implemented as a NIOS II MCU provided by Altera, is tasked with handling communication to the host computer, interpretation of commands and passing down

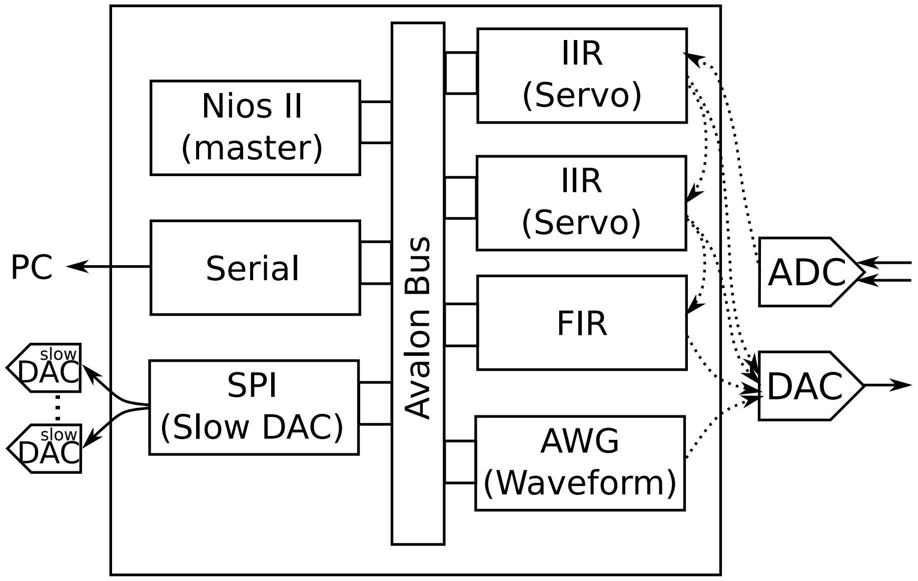


Figure 3.1: The general structure of the SOC design in an FPGA servo. The exact composition of an SOC design and the interface between the FPGA and the ADC and the DAC (represented in dotted lines) vary on a per platform basis.

commands to the specialized signal processing units that interact more closely with the electronic hardware. These messaging tasks are insensitive to delay and can be generated from the Altera toolbox or implemented in C directly. These design tools allow effortless implementation of human readable commands like “set 1 1024” through the serial port and allow the production of designs that can be easy replicated on another platform like the DE1-SOC. When the CPU and the specialized signal processing units are assembled together to form an SOC, the internal addresses of all the signal processing units are also combined to form a coherent memory space. The memory layout as seen by the CPU is detailed in Appendix C on a per module basis.

We note here that the Quartus 13.1 design suite was used to design the firmware in the DE3 whereas Quartus 13.0 was used for the DE2 firmware design. In the case of the DE2, the associated FPGA Cyclone II was depreciated in later versions. This also prompted our interest in porting DE2-related development into a more recent platform, the DE1-SOC. This thesis tries to make explicit the FPGA used for each study, and it also comments on the compatibility of the design with other FPGA platforms, especially the DE1-SOC whenever applicable.

3.2 IIR Module

The servo logic of the FPGA servo is implemented an IIR filter or set of filters. This is because an IIR filter can describe transfer functions of various levels of complexity and has a mathematical representation that is straightforward to implement in an FPGA. This section covers the use of IIR filters in FPGA servos in detail. It starts with the representation of controller logic in the IIR format and subsequently covers the computation precision, delay and footprint of an IIR filter.

3.2.1 Controller Representation

PID control is a common form of controller. The PID controller and the variations of a PID controller, such as the Proportional and Derivative (PD) controller and the Proportional and Integral (PI) controller, cover a large range of applications. Extension of PID into a more complex controller such as a Proportional Integral and Integral (PII) controller, a Proportional and Integral Cubed (PI^3) controller or a PI controller with lag-lead compensation is possible and often offers improvement in control performance. The derivation of PID into its IIR form is covered in detail here, although the FPGA implementation of the servo, an IIR filter, isn't limited to the PID alone.

The Proportional (P), Integral (I) and Derivative (D) parameters used in various PID tuning procedures usually refer to one of the three most common PID forms. They are the standard form, the series form or the parallel form [28, p.159-162], with the following expressions.

$$C_{standard}(s) = K_p\big(1 + \frac{1}{T_i s} + \frac{T_d s}{\tau_D s + 1}\big) \tag{3.1}$$

$$C_{series}(s) = K_s\big(1 + \frac{I_s}{s}\big)\big(1 + \frac{D_s s}{\gamma_s D_s s + 1}\big) \tag{3.2}$$

$$C_{parallel}(s) = K_p + \frac{I_p}{s} + \frac{D_p s}{\gamma_p D_p s + 1} \tag{3.3}$$

Here, the letters I and D refers to the integral gain and the derivative gain in the respective PID form with the subscript "p" for parallel and "s" for series. The letter K represents either the overall gain (in the case of the standard PID form and the series PID form) or only the gain for the proportional component (parallel PID). As an alternative expression, integral and derivative components are expressed by their corresponding time scales as the integral (reset) time (T_i) and the derivative time (T_d) in the standard form. The PID forms are all second order systems each representing two poles, with one of them at $s = 0$ and two zeros. The non-zero pole is omitted in some references for brevity [29], equivalent to setting τ_D, γ_s, and γ_p in equation 3.1, 3.2 and 3.3 to zero (these parameters are technically needed to limit the gain of the transfer functions as frequencies approach infinity). From a practical standpoint, the variety of PID forms are motivated by their similarity with various underlying implementation structures. The series form is often associated with an analog PID circuit based on op-amps, where each component implements part of the transfer function and are cascaded in stages to form the PID. The parallel form is often seen in MCU implementation of PID where integral, derivative of error and the error itself are weighted separately and summed at the end of each "for" loop.

The PID form most suitable for the IIR implementation in the FPGA servo can be described by the following relationship.

$$C_{FPGA}(s) = K\frac{(s-\omega_{z_0})(s-\omega_{z_1})}{(s-\omega_{p_0})(s-\omega_{p_1})} \tag{3.4}$$

Here, the poles and zeros are explicitly stated as $\omega_{p_{0,1}}$ and $\omega_{z_{0,1}}$. The poles and zeros are labelled such that $|\omega_{p_0}| \leq |\omega_{p_1}|$ and $|\omega_{z_0}| \leq |\omega_{z_1}|$. This implies that the low frequency pole, ω_{p_0} is at $s = 0$ and it refers to the integral term. The expressions for ω_{z_0} and ω_{z_1} can be obtained in the limit when τ_D, γ_s, and γ_p are close to 0, and is given by the following relationships.

$$\omega_{z0,1(standard)} = \frac{-T_i \pm \sqrt{T_i^2 - 4T_iT_d}}{2T_iT_d} \tag{3.5}$$

$$\omega_{z0,1(series)} = -I_s, -\frac{1}{D_s} \tag{3.6}$$

$$\omega_{z0,1(parallel)} = \frac{-K_p \pm \sqrt{K_p^2 - D_pI_p}}{2D_p} \tag{3.7}$$

The assumption of τ_D, γ_s, and γ_p being close to 0 does not affect the shape of the transfer function in the region $s \ll |\omega_{p_1}|$. This is because the assumption is equivalent to omitting the high frequency pole ω_{p_1}, and in a PID controller, the corner frequency of the poles and zeros are typically arrange to be $0 = |\omega_{p_0}| \ll |\omega_{z_{0,1}}| \ll |\omega_{p_1}|$. In this approximation, the high-frequency pole ω_{p_1} in equation 3.1, 3.2 and 3.3, responsible for limiting controller gain at high frequency, are removed. The value of this high-frequency pole can be directly obtained from equation 3.1, 3.2 and 3.3 as the following,

$$\omega_{p_1} = \underbrace{-\frac{1}{\tau_D}}_{\text{standard}} = \underbrace{-\frac{1}{\gamma_s D_s}}_{\text{series}} = \underbrace{-\frac{1}{\gamma_p D_p}}_{\text{parallel}} \tag{3.8}$$

resulting in a transfer function gain of the following.

$$K = \underbrace{\frac{K_p T_d}{\tau_D}}_{\text{standard}} = \underbrace{\frac{K_s(\gamma_s + 1)}{\gamma_s}}_{\text{series}} = \underbrace{\frac{K_p}{\gamma_p}}_{\text{parallel}} \tag{3.9}$$

When τ_D, γ_s, and γ_p are zero, the gain take on the value of the following.

$$K = \underbrace{K_p}_{\text{standard}} = \underbrace{K_s \cdot D_s}_{\text{series}} = \underbrace{D_p}_{\text{parallel}} \tag{3.10}$$

In the above expressions, the zeros and poles are explicitly written to have a negative real part, indicating that all the poles and zeros of the PID controller are on the left-hand-side of the s-plane. The collection of poles and zeros in the s-domain can be mapped into the z-domain, with the following relationship.

$$z = e^{sT} \tag{3.11}$$

Here, T is the sampling time, inversely related to f_s, the sampling frequency. The sampling frequency is also sometimes referred to as the clock frequency f_{clk}. This relationship is derived from the definition of Laplace transformation and Z-transformation. Reference can be found in various texts [30], [31]

Because the PID controller has 2 poles and 2 zero in the s-domain, its corresponding z-domain representation should also have 2 poles and 2 zeros, in the format of the following relationship,

$$C_{FPGA}(z) = K\frac{(z - z_0)(z - z_1)}{(z - p_0)(z - p_1)} \tag{3.12}$$

This relationship can be expanded like the following,

$$C_{FPGA}(z) = K\frac{1 - (z_0 + z_1)z^{-1} + (z_0 z_1)z^{-2}}{1 - (p_0 + p_1)z^{-1} + (p_0 p_1)z^{-2}} \tag{3.13}$$

Here, the z-domain poles and zeros are related to the s-domain poles are zeros with the following relationships.

$$z_{0,1} = e^{\omega_{z_{0,1}}/f_s}, p_{0,1} = e^{\omega_{p_{0,1}}/f_s} \tag{3.14}$$

Here, the s-domain poles and zeros, $\omega_{z_{0,1}}$ and $\omega_{p_{0,1}}$, have been defined previously in Equations 3.5 - 3.8.

To convert the z-domain representation of the PID controller in the form of an IIR filter, a final transformation is needed to cover the z-domain relationship into a discrete-time domain relationship. The transformation converts z^{-1} in z-domain to a single clock delay in the discrete time domain. This transform the previous z-domain relationship into the following.

$$Kx[n] - K(z_0 + z_1)x[n-1] + K(z_0 z_1)x[n-2] = y[n] - (p_0 + p_1)y[n-1] + (p_0 p_1)y[n-2] \tag{3.15}$$

Here, $x[n]$ and $y[n]$ are the time domain input and output for $C_{FPGA}(z)$. By introducing the IIR coefficients, we arrive at the typical form of an IIR filter.

$$b_0 x[n] + b_1 x[n-1] + b_2 x[n-2] = y[n] + a_1 y[n-1] + a_2 y[n-2] \tag{3.16}$$

Here, the IIR coefficients, b_0, b_1, b_2, a_1 and a_2, are defined according to the convention of the Direct Form I (DFI) and the Direct Form II (DFII) as shown in shown in Figure 3.2.

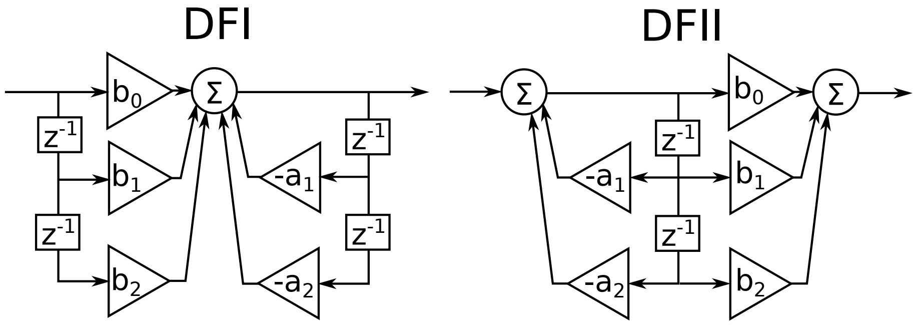


Figure 3.2: The DFI (left) and DFII (right) representation of a second order IIR.

The IIR coefficients has the following relationship to z-domain transfer function.

$$\begin{aligned}
b_0 &= K \\
b_1 &= -K(z_0 + z_1) \\
b_2 &= K z_0 z_1 \\
a_1 &= -(p_0 + p_1) \\
a_2 &= p_0 p_1
\end{aligned} \tag{3.17}$$

The graphical representation of an IIR in DFI shown in Figure 3.2 corresponds to the FPGA implementation almost exactly. Details of the IIR implementation are discussed in section 3.2.3.

Bilinear Transformation

In general, an arbitrary transfer function in the s-domain can be converted into the z-domain by first extracting the gain, poles and zeros as illustrated in Equation 3.4 and then applying the transformation relationship between the s-domain and the z-domain described by Equation 3.11. Other variations of this transformation exist and the most widely used variation is the bilinear transformation. The bilinear transformation is the first order approximation of the s-to-z transformation described by Equation 3.11. The bilinear transformation has the following expression.

$$s = \frac{1}{T}\ln(z) \approx \frac{2}{T}\frac{z-1}{z+1} \tag{3.18}$$

This expression can be used to perform a "substitution of variable" to convert an s-domain transfer function into its z-domain equivalent. The approximation has good accuracy when all poles and zeros in the s-domain are much smaller than the sampling frequency. [32, p.221226]. The bilinear transformation has the advantage of being much less computationally intensive than the exact transformation and is available in both MATLAB's control system tool box and Scipy's Signal Processing library. Both tools can be helpful for generating and verifying IIR coefficients.

In addition to the exact pole-zero mapping and the bilinear transformation method, other ways of generating IIR coefficients are investigated for the use in a servo system. Section 3.2.9 elaborates on the varying degrees of success of this endeavour.

3.2.2 First-Order IIR Filters

Many commonly used filters can be realized with a first-order IIR filter. This section describes the formulation of 4 filters: the pure integrator,the PI filter, the Low Pass (LP) filter and the High Pass (HP) filter. The IIR representation of these filters can can be obtained from their s-domain transfer functions followed by an s-to-z transformation; however, this section provides an alternative interpretation for each filter in an attempt to promote an intuitive understanding of the IIR filters.

The pure integrator is a filter that integrates continuously. The following is an IIR filter that implements a pure integrator.

$$y[n] = b_0 x[n] - a_1 y[n-1] \; (b_0 = 1, \, a_1 = -1) \tag{3.19}$$

Here, the definition of the IIR coefficients is according to Equation 3.16, with the coefficients b_1, b_2 and a_2 set to 0. Equation 3.19 can be treated as a recursive relationship between the input and the output of the IIR filter for a more intuitive perspective. By converting the recursive relationship into a direct relationship between the input and the output of the IIR filter, Equation 3.19 can be expanded like the following.

$$\begin{aligned} y[n] &= x[n] + y[n-1] \\ &= x[n] + x[n-1] + y[n-2] \\ &= x[n] + x[n-1] + x[n-2] + y[n-3] \\ &\dots \\ &= \sum_{i=0}^{n} x[i] \end{aligned} \tag{3.20}$$

This leads to the expression of an accumulator, where the output of the filter is the sum of all past inputs.

A PI filter is a variant of the PID controller without the D component. The PI filter is a single order system because it contains a pole at DC and a zero at a finite frequency. The PI filter can be implemented in a first-order IIR filter in the following way.

$$y[n] = x[n] + b_1 x[n-1] + y[n-1], \ -1 < b_1 < 0 \tag{3.21}$$

The PI filter resembles the pure integrator, with the exception of a non-zero b_1. A negative b_1 coefficient places a zero at a finite frequency and it effectively reduces the effect of the integrator at high frequencies.

Another example, the LP filter, is a filter that attenuates high frequency components in a signal. The following is a LP filter in its IIR form.

$$y[n] = b_0 x[n] - a_1 y[n-1] \ (b_0 = 1, \ -1 < a_1 < 0) \tag{3.22}$$

Here, we can see a similarity between this expression and an averaging technique in signal processing called the “weighted sum”. By converting the recursive relationship into a direct relationship between the input and the output of the filter, Equation 3.22 can be expanded like the following,

$$\begin{aligned} y[n] &= x[n] + \alpha y[n-1] \ (\alpha = -a_1 \in (0,1)\ ) \\ &= x[n] + \alpha\Big(x[n-1] + \alpha y[n-2]\Big) \\ &= x[n] + \alpha x[n-1] + \alpha^2\Big(x[n-2] + \alpha y[n-3]\Big) \\ &\ldots \\ &= \sum_{i=0}^{n} \alpha^i x[n-i] \end{aligned} \tag{3.23}$$

Here, the components in the sum are exponentially weighted based the amount of time elapsed since that sample was taken.

The final example, the HP pass filter, can also be considered a DC block with finite gain at high frequencies. A HP filter has a zero at DC and a pole at a finite frequency. It has the following IIR relationship.

$$\begin{aligned} y[n] &= b_0x[n] - b_1x[n-1] - a_1y[n-1] \ (b_0 = 1,\, b_1 = -1,\ -1 < a_1 < 0) \\ &= x[n] - x[n-1] - a_1y[n-1] \ (-1 < a_1 < 0) \end{aligned} \tag{3.24}$$

The DC blocking behaviour of the IIR form of an HP filter can be observed from the coefficients $b_0 = 1$, $b_1 = -1$, where the sum of a static input weighted with the coefficients b_0 and b_1 is 0. The additional requirement of $-1 < a_1 < 0$ is equivalent to adding a LP filter, which limits the gain at high frequency.

3.2.3 Implementation Details

Fractional Resolution Representation

In this work, the IIR filter is implemented in the form of Fixed Point (FP) arithmetic due to the forbidding high cost of floating point arithmetic in FPGAs used in this project. All the IIR coefficients are implemented as FP numbers with the fractional resolution of R bits. The FP number satisfy the following relationships.

$$b_n = \frac{B_n}{2^R} \text{ , R=fractional width} \tag{3.25}$$

$$a_n = -\frac{A_n}{2^R} \tag{3.26}$$

Here the IIR coefficients in capital letter are the FP numbers, and the IIR coefficients in lower letters are defined Section 3.2.1. The programming interface between the the FPGA servo and the PC also adopt the FP format. It is important to note the negative sign in Equation 3.26 when programming the IIR coefficients. This negative sign is the result of reducing negation oprations in the IIR implementation in the FPGA. Programming the A_n coefficients with the wrong sign will result in poles being placed on the right-hand-side of the s-domain (oscillatory) rather than the left-hand-side of the s-domain (stable).

Since the implementation of arithmetic operation in FPGAs requires explicit declaration of all underlying operations (as opposed to designing within a selection of arithmetic operations that have been previously defined in an MCU), some design pitfalls exist in the FPGA implementation of an IIR filter. One such mistake is illustrated in Figure 3.3. Here, the IIR coefficients are 16-bit in width and the width of all operands are explicitly annotated like it would be in an Hardware Description Language (HDL) description of an IIR filter. At a first glance, the IIR filter appears to implement 10 fractional bits in all the coefficients correctly. This can be confirmed by tracing the gain through each data path in the IIR filter. What goes hardly noticed in this IIR filter is the loss of computation precision. We can identify this issue by setting $B_0 = 1$, $B_1 = 0$ and $A_1 = 2^{10} = 1024$, which should configure the IIR as a pure integrator with a very small gain. When the input is set to a small value, 1 for example, the output of a pure integrator should produce a shallow ramp, but the implementation shown in Figure 3.3 produces a sustained 0 at the output.

The correct implementation, as shown in Figure 3.4 is similar to the previous implementation, but many of the internally stored values in the correct implementation are wider in width. The filter does not have the computation issue described above due to the additional computation precision.

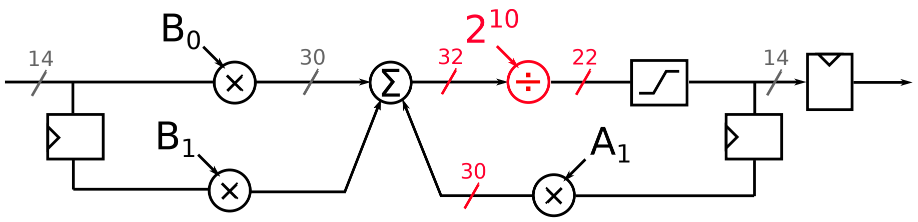


Figure 3.3: An incorrect implementation of a first order IIR filter with FP coefficients (B_0, B_1 and A_1) with 10-bit fractional resolution. The FPGA implementation is made up of simple operations like multiplication ($\times$), addition ($\sum$), division ($\div$) by the power of 2 implemented as a shift operation, saturation logic (◪) and clock delays (▭). the width of internal path is noted in grey next to the path. The placement of the $\div 2^{10}$ operation has a large effect on the effectiveness of the implementation. The difference between the incorrect implementation and the correct implementation shown in Figure 3.4 is highlighted here in red.

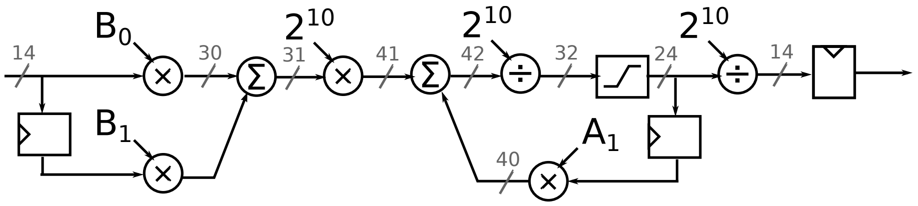


Figure 3.4: A correct implementation of a first-order IIR filter with FP coefficients (B_0, B_1 and A_1) with 10-bit fractional resolution. The placement of $\times 2^{10}$ and $\div 2^{10}$ has a large effect on the width of the internally stored values and subsequently the correctness of the IIR implementation. The FPGA implementation is made up of simple operations like multiplication ($\times$), addition ($\sum$), division ($\div$) by the power of 2 implemented as a shift operation, saturation logic (◪) and clock delays (▭).

IIR Implementation for Different FPGA Servos

It is worthwhile to summarize the variety of IIR filters commissioned in this work. In broad categories, as listed in Table 3.1, two IIR filters are commissioned, one for the DE3 FPGA exclusively and the other one for the DE2 FPGA but the latter is not platform dependent. Table 3.1 also summarizes other differences between the two implementations. One such difference is that the DE3 version implements 32-bit coefficients while the DE2 version implements 16-bit coefficients. We note that the DE2 version is designed to have adjustable coefficient width upon synthesis of the HDL.

Table 3.1: A summary of filters implemented in various FPGA platform in this work. The format Qx.y means that the FP representation of the coefficient has x integer bits and y fractional bits. In the case of the Q3.28 format, a total of 32 bits are needed to represent 1 sign bit, 3 integer bits and 28 fractional bits. In the case of the Q5.10 format, a total of 16 bits are needed.

	Platform	Portability	Coefficient Format	Complexity (Single)
IIR	DE3	No	Q3.28	3
IIR	DE2	Yes	Q5.10	3

The flexibility of the DE2 implementation means that little effort is needed to adopt this implementation onto a different FPGA like the DE1-SOC, or to change the coefficient width with from 16-bit to 32-bit. This portability, combined with compatibility of the daughter-card with DE1-SOC (described in Chapter 2) means that the DE2 servo design can be ported to DE1-SOC and most other development education platform by Altera.

Integral Anti-Windup

In control systems, integral anti-windup is often implemented to limit the maximum contribution the integral term has to the total control gain. The purpose is to prevent the controller from over compensating as the closed-loop system recovers from an out-of-lock state. In the FPGA servo, integral anti-windup is implemented as a saturator block at the output of the filter, as shown in Figure 3.4 with the symbol of $\boxed{\diagup}$, that clamps the IIR output to the DAC range. We note that this particular implementation of integral anti-windup has a problem with the high order of integration as discussed in Section 3.2.8, but is nevertheless chosen due to its negligible computation delay and modest resource footprint.

Less Than Unity Servo Gain

Adapting an existing IIR filter, as it is shown in Figure 3.4 to implement a gain of 2^a, with $a < 0$ calls for modification of the filter. Changing the input coefficient values to implement a lower gain is not advisable because the coefficients are coupled to the frequency resolution (refer to Section 3.2.5 for the relationship between the coefficients and the frequency resolution of poles and zeros). Alternatively, implementing the lower gain can be done by dividing the output of the IIR filter; however, this must be done with care because an incorrect placement of the division can reduce the output range of the IIR filter. A reasonable way to implement logic gain of 1/2 is illustrated in Figure 3.5. In the case of a gain that is less than unity, the internally stored past IIR output needs to be expanded to have more fractional bits to compensate for the attenuation at the output of the IIR filter.

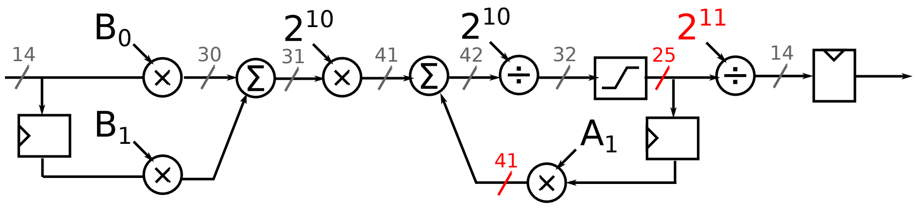


Figure 3.5: Implementation of a 1st order IIR with a built-in gain of 1/2. The coefficients are FP with 10-bit fractional resolution. The difference between the gain of 1/2 implementation and the unity gain implementation is highlight in red. The division of 2 is placed at the output of the filter but the width of the internally stored values are increased to compensate for the potential loss of the output range.

This particular filter implementation is used in the study of the optimal distribution of gain in a closed-loop system in Section 4.1.5. The gain distribution is investigated because of its effect on noise, and the results of this study are presented in Chapter 4. Our findings show that increasing the input gain and reducing the gain through the FPGA logic can improve the noise suppression capability of the closed-loop system. However, the IIR filter with 16-bit coefficients is found to have a limited frequency resolution for the placement of poles and zeros.

How Higher Order Filters are Implemented

Transfer functions with a higher number of poles and zeros are implemented by adding depth to each IIR or by cascading them as illustrated in Figure 3.6. An IIR filter of high-order can implement a maximum number of zeros or poles that is equal to the depth of the IIR filter. A few high-order IIR filter when cascaded implement a transfer function in the z-domain like the following.

$$C_{FPGA}(z) = \left(\frac{b_0 + b_1 z^{-1} + ... + b_{n-1} z^{-(n-1)} + b_n z^{-n}}{1 + a_1 z^{-1} + ... + a_{n-1} z^{-(n-1)} + a_n z^{-n}}\right)\left(\frac{b'_0 + b'_1 z^{-1} + ...}{1 + a'_1 z^{-1} + ...}\right)... \tag{3.27}$$

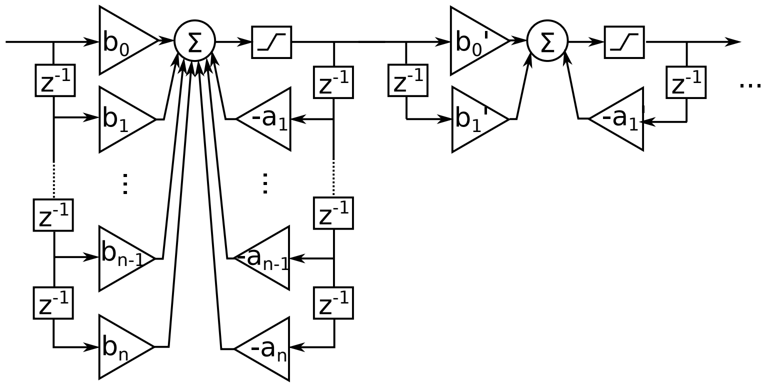


Figure 3.6: Two methods for expanding the order of IIR, depth-wise (left) and length-wise in cascaded form (right)

Both methods of implementing multiple poles and zeros have some disadvantages. When using a single high-order IIR filter to implement multiple poles and zeros, the frequency resolution for pole or zero placement diminishes as more poles of a half of or zeros are implemented in the same IIR filter. In the case of the cascaded configuration the disadvantage is a larger delay. When implementing complex filters such as the PII with lag-lead filter used in Chapter 4, both techniques are used. For the PII filter with lag-lead, the 6 poles and 6 zeros in the transfer function are divided up into two third-order IIR filters with 32-bit coefficients in a cascaded configuration. The poles and zeros are distributed in such a way that both IIR configurations require similar amount of computation precision.

3.2.4 Verification and Simulation

Verification at the basic level is essential for correctly designing a moderately complex system. The importance of verification at the HDL level was often overlooked in this FPGA servo project and this often leads to detours. Three levels of verification were implemented by the end of the project. The simulation includes testing of individual data paths of an IIR filter, the open-loop simulation of IIR filters and the closed-loop simulation of IIR filters.

As the simplest verification method illustrated here, the data path verification of an IIR filter is sufficient to confirm the correctness of an IIR implementation. The open-loop and closed-loop simulation serves more as way to strengthen our understanding of the IIR filter. More specifically, the result of an open-loop simulation resembles the transfer function measurement of the FPGA servo made by a network analyzer (as shown in Section 3.2.5). Similarly, the result of a closed-loop simulation result allows us to predict the closed-loop performance of the FPGA servo (which is evaluated in Chapter 4).

Issues encountered in implementing the FPGA servo are investigated with the verification and simulation tools described in this section. An example of an issue encountered is the loss of computation precision due to the incorrect handling of the Least Significant Bit (LSB)s as illustrated in Figure 3.4 in Section 3.2.3. Another issue is the saturation of signals at the input or output of an IIR filter, similar to the saturation of the error signal at the ADC described in Section 4.4. Finally, a set of closed-loop simulations are useful to investigate the maximum level of white noise that a closed-loop IIR filter can suppress, a study that is much more time-consuming if done with the FPGA servo hardware.

All the HDL simulations are made using the Altera Edition of Modelsim provided in the Version 13.1 of the Quartus II design suite. The compatibility between the simulation tool and the Cyclone II FPGA is not affected by the removal of Cyclone II from the list of supported devices in Quartus 13.1. The HDL simulation is checked against a separate implementation of an IIR filter in C++. The rest of the tasks, including signal processing, analysis and plotting of simulation results are made in MATLAB.

IIR Data Path Verification

The verification of all data paths in an IIR filter insures the correctness of the IIR filter. The test values, as listed in Table 3.2, cover the most commonly made mistakes in implementing an IIR filter in an HDL. The tests are designed such that if any of the test conditions were to fail, the tester should be able to quickly isolate the problem to a few lines of code.

The correct gain and delay in an IIR filter insures that the filter implements the correct transfer function. An HDL implementation of an IIR filter is particularly susceptible to unintended

Table 3.2: Test values for the IIR filter used in the DE2 servo, where FP representation of the the IIR coefficients, B_0, B_1, B_2, B_3, A_1, A_2 and A_3, all have 10 bits (the relationship between the FP numbers and the IIR coefficients can by found in Equation 3.25 and 3.26). The implementation details of a first order IIR filter can be found in Figure 3.4. The input width and the output width of the IIR filter are both 14-bit.

Test	Data In	B_0	B_1	B_2	B_3	A_1	A_2	A_3	Data Out
Gain, Signed and Delay	100	1024	0	0	0	0	0	0	100
		0	2048	0	0	0	0	0	200
		0	0	3072	0	0	0	0	300
		0	0	0	4096	0	0	0	400
		1024	0	0	0	1024	0	0	100, 200, 300, ...
		1024	0	0	0	0	1024	0	100 x 2 cycle, 200 x 2 cycles, 300 x 2 cycles, ...
		1024	0	0	0	0	0	1024	100 x 3 cycle, 200 x 3 cycles, 300 x 3 cycles, ...
Bound	4096	16384	0	0	0	0	0	0	8192
		-16384	0	0	0	0	0	0	-8191
LSB	1	1024	-1023	0	0	1024	0	0	0 x 1024 cycles, 1 x 1024 cycles, ...
Low-pass	0 $\downarrow$ 100	16	0	0	0	1008	0	0	Reaching 76 in 64 cycles, approaching the steady state of 100

latches which can modify the relationship between the IIR coefficients and the frequencies of poles and zeros that an IIR filter implements. The gain and delay in an IIR filter can be tested by monitoring the IIR output while activating one IIR coefficient at a time. The forward path coefficients, B_0, B_1, B_2 and B_3, are assigned different values and the output is expected to be updated with the gain and the corresponding delay of 1, 2 or3 clock cycles. The feedback path coefficients, A_1, A_2 and A_3 are tested by configuring the IIR filter as an integrator, by setting B_0 to 2^R (R is the fractional resolution of the coefficient in bit) and one of A_1, A_2 or A_3 to a positive value. The outputs of these integrators are expected to be ramps that update at an interval of 1, 2 or 3 clock cycles that correspond to the data path tested.

Testing the IIR filter around bound condition verifies the saturation and truncation logic located at the output of the IIR filter, and this prevents overflow or under-utilization of the IIR output. Whenever the bound condition fails the IIR filter can still implement the correct transfer function for a sufficiently small input, but a larger input triggers overflow and causes instability. In the tests for bound conditions, the IIR input is sufficiently large to cause the output to clip. The output is expected to clip at the maximum or minimum of the IIR output. Any other clipping behaviours need to be investigated.

Careful LSB handling is important for reaching a low noise floor when the IIR filter is used in a closed-loop system. Unwanted LSB truncation can manifest either as insensitivity to small fluctuation at the IIR input or inability to accurately address the frequency of a pole or a zero. Both scenarios can be tested by configuring the IIR as an integrator with the lowest frequency zero that the system can implement and setting the IIR input to 1. If the IIR filter produces a very slow ramp at the output then the LSBs are likely handled correctly. Otherwise, a sustained 0 at the IIR output indicates that the LSBs are handled incorrectly.

Open-Loop Simulation

The open-loop simulation of the IIR is a set of tests where the HDL implementation of IIR filters are given an input waveform generated from MATLAB, as opposed to the static test values described in the previous section. The most useful test waveform is the sinusoidal waveform which can be used to measure the transfer function of the HDL implementation of a IIR filter when the period of the sinusoidal wave is varied. The open-loop simulation of the HDL is compared against an C++ implementation of an IIR filter. As the C++ implementation uses double float operands, any mistakes frequently made in HDL with fixed point mathematics in sign extension, numerical saturation, numerical rounding can also be detected

Closed-Loop Simulation

The closed-loop simulation of IIR filters is very helpful for a better understanding of the IIR filter. The simulation is implemented by configuring an IIR filter in a closed loop in the HDL testbench and allowing a disturbance to be injected into the loop with a waveform generated from MATLAB. Figure 3.7 shows the similarity between the open-loop and closed-loop simulation of an IIR filter as they can be configured from the same simulation resources.

The test waveform used in the closed-loop simulation is typically a white noise source rather than the sinusoidal disturbance used for the open-loop simulation. The use of white noise has

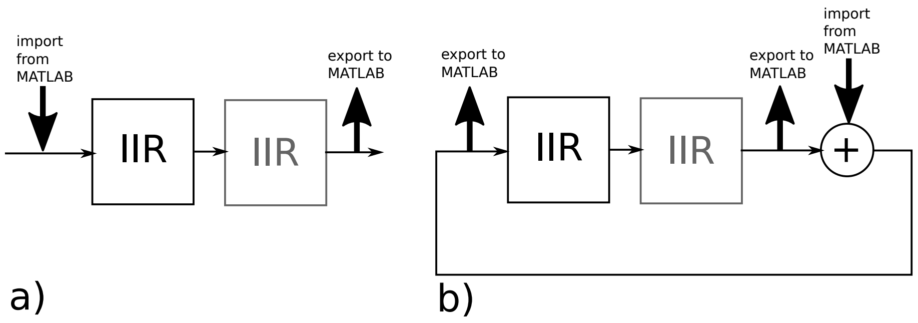


Figure 3.7: Simulation scheme of IIR in a) open-loop and b) closed-loop. Both simulation are implemented in HDL and in C++ to 1) verify the implementation of the IIR in FPGA and 2) to provide better understand of IIR filters

the advantage of probing a range of frequencies rather than a single one. The simulation can optionally export signals at various points of the loop. When the signal is analyzed with FFT, result is equivalent to taking the transfer function of the IIR filter when it is configured in closed-loop.

The method of extracting the transfer function from an IIR filter with the FFT and a white noise input is computationally intensive. This is because the resolution of transfer function is limited by the bandwidth of the white noise. This means that to obtain the transfer function of the simulated system at low frequency (with high resolution) the IIR filter needs to be simulated against a long duration of white noise. In quantitative terms, probing the frequency response of the IIR at 100Hz requires the IIR to be simulated over 10 ms worth of white noise waveform of the same duration. That is 200,000 frames of data if the IIR logic is clocked at 50 MHz. Both the HDL simulation in Modelsim and the C++ implementation can process a similar amount of data in under 15 minutes, although such a simulation in MATLAB would be less practical.

To demonstrate the usefulness of the closed-loop simulation, Figure 3.8 shows the effect of loss of computation precision. In this simulation, two cascaded IIR filters are configured in closed-loop and the noise spectrum at the input of the first IIR filter is computed. It is important to note that the amplitude of white noise is evaluated in spectral density with the quantization step at the input of the IIR filter used as the unit. In this simulation, a white noise disturbance with spectral density of 140 $/\sqrt{\text{Hz}}$ is injected into the loop at the output of the cascaded filters. As shown in Figure 3.7, the IIR implementation with LSB error has limited ability to suppress noise at low frequency. In comparison, the IIR performance without the LSB error reaches far below the quantization level of the IIR input, giving a maximum noise suppression of 10^4 times. It is important to note that a noise suppression of 10^4 times is equivalent to 80 dB, a level of suppression that is never achieved with the FPGA hardware (refer to Chapter 4). We speculate that other factors, such as input noise floor, limits the performance of the real FPGA servo.

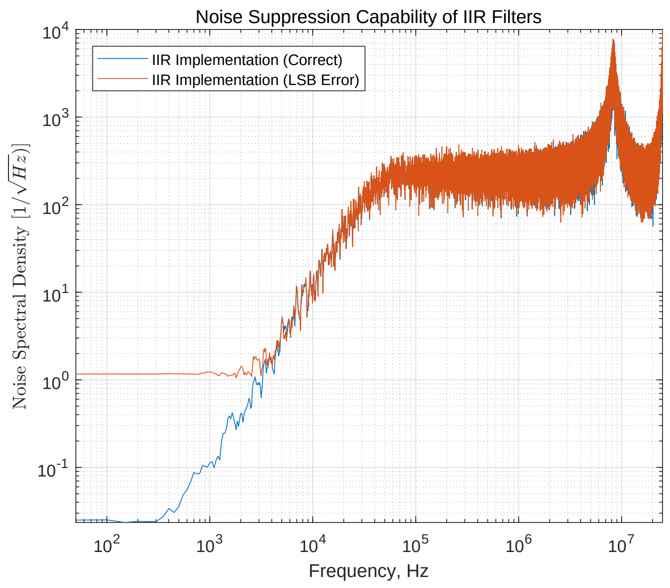


Figure 3.8: The closed-loop simulation with white noise as disturbance. This data shows the spectrum at the input of the closed-loop for two versions of IIR filter: the version of the IIR with the error in LSB handling (in red) and the correct version (in blue).

Another set of simulations demonstrates the versatility of the closed-loop simulation. As shown in Figure 3.9, this set of simulations reveals the effect of a limited IIR input and output range as the amplitude of disturbance increases. This set of simulations is implemented in C++ to emulate a single IIR filter with 14-bit wide input and output, configured in closed-loop. The level of noise injected into the loop is varied. It is important to note that amplitude of white noise is evaluated in spectral density with the quantization step at the input of the IIR filter used as the unit. For white noise with spectral density of 140 $/\sqrt{\text{Hz}}$ the peak-to-peak amplitude of the error signal becomes larger than the input range of the IIR module ($-8191 \sim 8192$ for a input width of 14-bit). This clipping causes the closed-loop system rapidly breaks down and amplifies noise rather than suppressing it. This behaviour is similar to the amplification of noise observed in Section 4.1.3 when the gain in the FPGA servo is over-tuned. However, when the input and output range of the IIR filter is extended, the amplification effect lessens. This allows us to conclude that the amplification of noise observed in Section 4.1.3 and this simulation is due to clipping at either the IIR input or the IIR output.

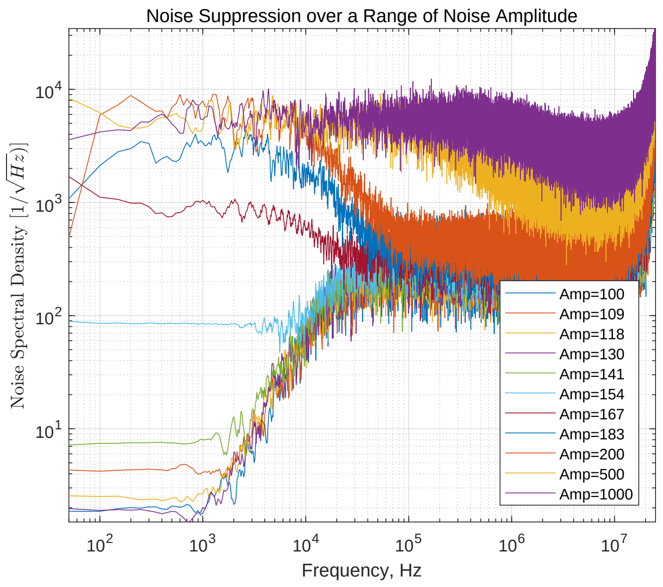


Figure 3.9: The effect of noise amplitude on noise suppression. This shows simulation of different level of noise injected until the servo break out of lock.

3.2.5 Characterization: Pole and Zero Resolution

In analog servos, the frequencies of poles and zeros are determined by the set of RC constants implemented in the feedback circuit of op-amps. The RC corner frequencies are often carefully selected to implement poles or zeros with even spacing in logorithmic scale at frequencies such as 10 Hz, 30 Hz, 100 Hz and etc [33]. The mechanism of enabling the poles or zeros in analog servos is through mechanical switches. In comparison, the FPGA servo can be configured with different frequencies of poles and zeros by simply updating the parameters of the IIR filter inside the FPGA servo. The added flexibility of the FPGA servo is nevertheless constraint by the frequency resolution of poles and zeros that the IIR filter can implement. This section explores the origin of these limitations in a simple first-order IIR filter.

Derivation of Frequency Resolution

Through experimenting with an IIR filter, we found that frequencies at which the poles and zeros can be placed in an IIR filter is quantized with a resolution set by the clock speed of the IIR filter ($\mathrm{f_{clk}}$) and the fractional width of the coefficients (R), like in the following proportionality relationship.

$$\Delta f \propto \frac{f_{clk}}{2^R} \tag{3.28}$$

This means that the frequency resolution of the IIR filter scales linearly with the clock frequency and increases exponentially as the fractional width of the coefficients increases. This intuitive scaling relationship can be derived from the mapping relationship between the s-domain and the z-domain as shown in Equation 3.11. The scaling factor in Equation 3.28 can be found by following the derivation provided in the following text.

In this derivation, ω_{p_0} and ω_{z_0} are used to represent radial frequency of a pole and a zero in the s-domain, and z_0 and p_0 are used to represent the z-domain equivalent of the pole and the zero. According to the mapping relationship between the z-domain and the s-domain as the following.

$$sT = \ln z,\, T = 1/f_{clk} \tag{3.29}$$

Here, ω_{p_0} and p_0 has the following relationship.

$$\omega_{p_0} T = \ln{(p_0)},\, T = 1/f_{clk} \tag{3.30}$$

Here T is the sampling time and is the inverse of clock frequency $\mathrm{f_{clk}}$ of the IIR filter.

It is important to note that the value of the pole in radial frequency is different from its value in ordinary frequency by a factor of 2π. The radial frequency is the quantity that s-domain poles or zeros are evaluated in, and the ordinary frequency is the quantity that is typically referred to when describing the periodicity of a waveform in the time domain. Here, f_{p_0} is used to represent the pole in ordinary frequency (in Hz), and it is related to ω_{p_0} by the following relationship.

$$\omega_{p_0} = 2\pi f_{p_0} \tag{3.31}$$

This means that the ordinary frequency of a pole is related to its z-domain equivalent by the following relationship.

$$f_{p_0} = \frac{f_{clk}}{2\pi} \ln(p_0) \tag{3.32}$$

Here, the term $\ln(p_0)$ can be evaluated by taking the linear approximate of $\ln$ as p_0 approach 1 from below. This approximation is equivalent to implementing a pole or a zero with frequency much lower than the clock frequency of the IIR filter ($z \to 1^-$ is equivalent to $s \to 0^-$). In the typical usage of a PID controller, the corner frequency of a pole or a zero is lower than 1 MHz while the IIR filter is clock between 50 MHz and 75 MHz in this work. This suggests that it is valid to approximate p_0 and z_0 as approaching 1 from below.

To reflect the hardware implementation in the FPGA, the z-domain pole p_0 is written in terms of the FP number like the following.

$$p_0 = \frac{A_1}{2^R} \tag{3.33}$$

Here A_1 is an FP number with R fractional bits (refer to Section 3.2.3 for definition of the IIR coefficients). Using this relationship, the frequency resolution of f_{p_0} can be expanded like the following.

$$\Delta f_{p_0} = \frac{f_{clk}}{2\pi} \ln\left(\frac{A_1}{2^R}\right) - \frac{f_{clk}}{2\pi} \ln\left(\frac{A_1 - \Delta A_1}{2^R}\right) \tag{3.34}$$

Here, the smallest change that is allowed in A_1 is $\Delta A_1 = 1$. By expanding the $\ln$ around 1, the frequency resolution can be found to be the following,

$$\begin{aligned} \Delta f_{p_0} &= \frac{f_{clk}}{2\pi}\Big(\frac{A_1}{2^R} - \frac{A_1 - \Delta A_1}{2^R}\Big),\ \Delta A_1 = 1 \\ &= \frac{f_{clk}}{2\pi \cdot 2^R} \end{aligned} \tag{3.35}$$

This concludes that the scaling factor in Equation 3.28 is 2π. This is confirmed by checking the transfer function of various IIR filters in MATLAB. When f_{clk} is to 50 MHz, the frequency resolution of a single pole or zero that implemented by a first order IIR filter is about 7.1 kHz when the coefficients has 10-bit fractional resolution (16-bit coefficients). Details on the measurement of frequency resolution can be found in the next section. When coefficients with 28-bit fractional resolution (32-bit coefficients) are used, the frequency resolution of a pole or a zero can be as low as $\Delta f = 48$ mHz.

The same scaling factor does not necessarily hold true for the frequency resolution of a zero in a first order IIR filter. This is because the z-domain implemented in the FPGA has the following relationship,

$$z_0 = \frac{B_1}{B_0} \tag{3.36}$$

as indicated by the definition of the IIR coefficients in Section 3.2.3. When the B_0 coefficient is fixed at $B_0 = 2^R$ (the FP equivalent of $b_0 = 1$), the frequency resolution of the zero has the same relationship as the frequency resolution of a pole as shown by the following relationship,

$$\Delta f_{p_0} = \frac{f_{clk}}{2\pi \cdot 2^R},\ B_0 = 2^R \tag{3.37}$$

However, when B_0 is not fixed at 2^R, the frequency resolution of the zero is $< f_{clk}/2\pi \cdot 2^R$ for $B_0 > 2^R$ and $> f_{clk}/2\pi \cdot 2^R$ for $B_0 < 2^R$.

In a more general case, an IIR filter of higher order can implement multiple poles and zeros. It is possible to show that the resolution of the pole placement in a multi-order IIR filter, depends on the value of all the other poles implemented in the same IIR filter, by factoring and reducing Equation 3.27 in Section 3.2.3. In summary, it is important to keep in mind that the resolution of a pole in an IIR filter that implement multiple poles is always worse than if a single pole is implemented (for further reading, please see [34, p.383]).

When working with IIR filter of higher order, it is important to note that small adjust to poles and zeros in z-domain can lead to dramatic change in the transfer function. In some cases, the phase response of the transfer function can be modified by as much as 180 degree, which can cause instability in a closed-loop system. While IIR filter with high computation precision can implement a more complex transfer function without introducing instability, for any IIR filter with finite fractional resolution, this problem can occur if the transfer function becomes too complex. As a solution to this issue, this work develops a MATLAB routine (refer to Appendix D) that compares the s-domain transfer function and the closest transfer function that the hardware can implement in both amplitude and phase. Ultimately, it is up to the user to make sure that the phase response of the s-domain transfer function is not compromised when converted to a hardware implementation.

Measurement of Frequency Resolution

The frequency resolution of poles and zeros in an IIR filter can be characterized as a series of transfer function with small variation in the IIR coefficients. A network analyzer like the SR780 can be used to take the transfer function. The network analyzer generates a chirp signal and when it is used as the input to the FPGA servo, the cross correlation (computed by the SR780) between the output of the FPGA servo and the chirp signal gives the transfer function.

A typical result of this measurement is shown in Figure 3.10. Here, the IIR filter is configured as a PI filter with a fixed corner frequency and is tested against chirp signals of different amplitudes. In all of the measurements, the transfer function agrees with the ideal transfer function of a PI (in the dotted line) at intermediate frequencies, between 10 kHz to 1 MHz (only the regions between 10 kHz and 100 kHz is shown). At low frequencies, the amplitude response of the measured transfer function saturates, while the phase response does something completely unrecognizable. The non-ideal behaviour of the PI filter at low frequencies is not visible in its z-domain transfer function where the input and the output signals are unlimited in amplitude. As it can be observed in Figure 3.10, the transfer function measurement approaches the ideal PI transfer function, as the input amplitude is lowered and the IIR output saturates around 200 mV regardless of the input amplitude. This suggests that the non-ideal behaviour is rather caused by saturation at the output of either the IIR filter or the FPGA servo. This saturation voltage may differ in other testing scenarios if any DC component is present in the chirp signal or if the gain and the offset circuit is part of the measurement.

Figure 3.11 shows the lowest 3 non-zero zeros implementable by a first order IIR with coefficients with 10 fractional bits. The coefficients are programmed as $B_0 = 1024$ and $A_1 = 1024$ for all three filter, and $B_1 = -1021$, $B_1 = -1022$ and $B_1 = -1023$ respectively for each

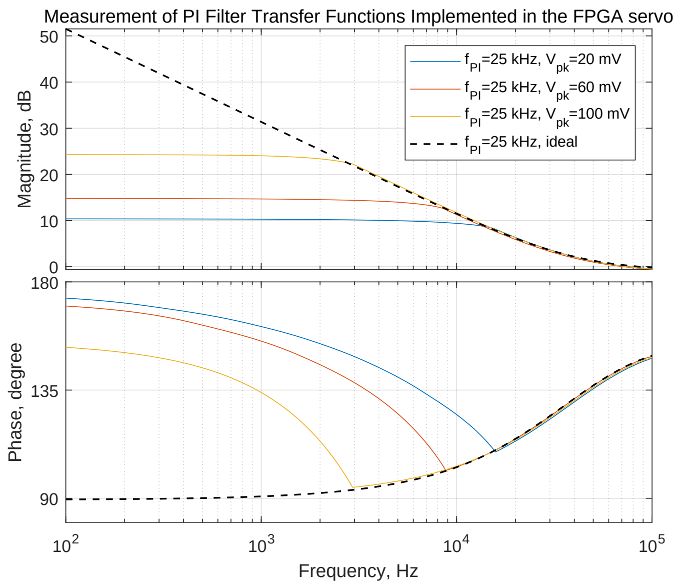


Figure 3.10: Transfer function measurement (with SR780) of a PI filter with chirp input of different amplitude. The measured transfer function approach the ideal shape as the chirp signal becomes smaller in amplitude. This suggests that the non-ideal behaviour of the implemented PI transfer function is due to clipping at the output of the IIR filter or the FPGA servo.

filter. The corner frequencies of these PI filters are, 22 kHz, 14 kHz and 7.1 kHz, indicating a frequency resolution of roughly 7.1 kHz. When compared against an IIR filter with high fracitonal resolution, a higher resolution of zero can be found.

An alternative method for measuring the frequency resolution of a single pole or zero is to set the IIR filter to the lowest pole or zero that is allowed by the IIR filter and use a function generator to scan over a small frequency range until the 3 dB (corner) frequency is found. This method do not verify the shape of the transfer function everywhere else in the frequency range, but is an extremely fast sanity check for a first order IIR filter, and it should be used with care on a higher order IIR filter.

3.2.6 Characterization: Resource Allocation

The computation footprint of IIR filters is an important consideration in selecting the right FPGA in an FPGA servo design. Before investigating this issue, we would like to identify the computation resources in an FPGA. The simplest type of configurable logic unit in an FPGA is the LE (by Altera convention) or Look-up Table (LUT) (by Xilinx convention). Although

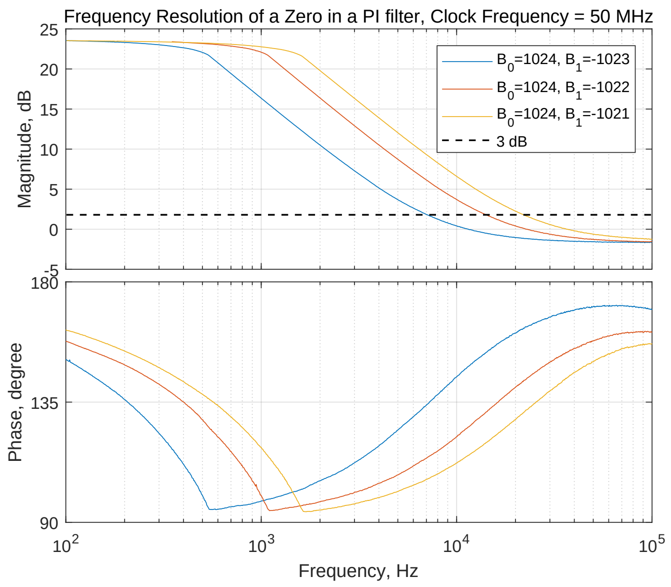


Figure 3.11: Transfer function measurement that demonstrates the frequency resolution of a zero in a PI filter, where coefficients has 10 bit fractional resolution. The coefficients are set to $B_0 = 1024$, $A_1 = 1024$ and $B_1 = -1021$, $B_1 = -1022$ and $B_1 = -1023$ for each filter. The corner frequency of the PI filters can be found as the 3 dB frequency (intersection with the dashed line) and are respectively 22 kHz, 14 kHz and 7.1 kHz.

LEs can be configured to form any kind of logical devices, arithmetic operations are best implemented with the FPGAs DSP resources. The DSP based computation resources in the two FPGAs used in this thesis, the high-performance Stratix III in DE3 and the economic Cyclone II in DE2, are different in complexity, as illustrated by the comparison in Figure 3.12.

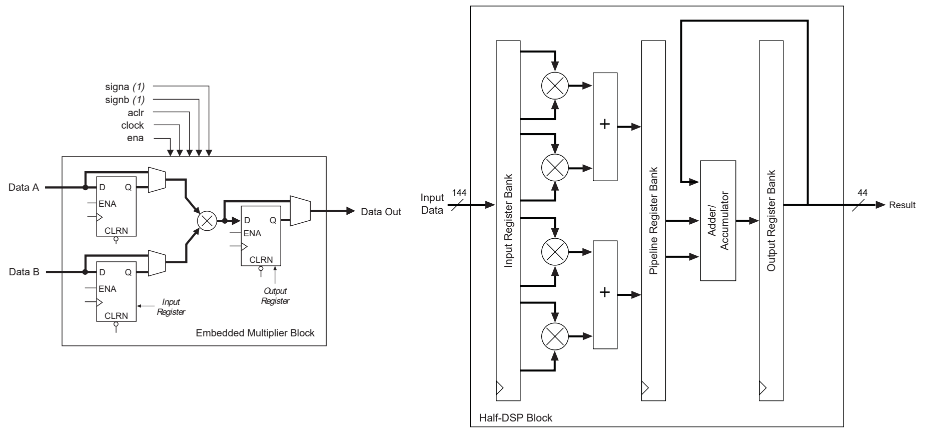


Figure 3.12: Comparison between DE2 (left) and DE3 (right) computation resources

The DSP resources in an Cyclone II FPGA are simple multipliers with optional input and output latches as illustrated in Figure 3.12 (left). In comparison, the DSP resources in an Stratix III

FPGA are DSP engines which contain multipliers followed by two stages of accumulators, with the first accumulator responsible for summing the output of multipliers and the second accumulator responsible for summing of output several DSP engines, as illustrated in Figure 3.12 (right). It is important to note that the HDL synthesizer shipped in Quartus 13.1 does not provide full support to all necessary features of the DSP engines in the Stratix III FPGA, which necessitates declaration of the DSP explicitly. Although using specific accumulator features in the DSPs helps to lower the computation delay in the IIR filter, reference to the DSP features reduces the portability of this HDL implementation. Thus, specific references to DSP structures are avoided unless necessary.

In order to accurately assess the computation footprint of IIR filters in FPGAs, we use 18-bit by 18-bit multipliers as the unit of the assessment. Among the IIR filters implemented in this work (Table 3.1), the most suitable IIR implementaion for this study is the filter used in the DE2 servo because it can be easily configured to implement a variety of coefficient width and various IIR depth. This study also assumes that the accumulators absent in the Cyclone II FPGA are to be constructed out of LEs. Table 3.3 shows that the number of 18×18 multipliers needed to implement IIR filters of various coefficient widths and depths.

Table 3.3: A summary of the computation footprint of various IIR filters with different depth and coefficient widths, quantified by the number of 18 × 18 multipliers.

	16 bit coeff., with 10 bit fractional resolution (18×18 multipliers)	32 bit coeff., with 20-28 bit fractional resolution (18×18 multipliers)
1st order IIR	4	8
2nd order IIR	7	13
3rd order IIR	10	18

With the computation footprint of IIR filters in Table 3.3 and the computation resources in FPGAs in Table 3.4, it is possible to devise a strategy on distributing DSP resources in an FPGA servo. For example, the IIR filter needed to implement the PII + lag-lead control is in the form of 2 third-order IIR filters in cascaded configuration both with 32-bit coefficients. The computation footprint of this IIR filter is 36 multipliers per channel, an impossibility for the DE2 FPGA (with 35 multipliers in total), but is manageable for both the DE3 FPGA (384 multipliers) and the DE1-SOC FPGA (174 multipliers).

Table 3.4: Comparison of computation resources in various FPGA platforms

	DE3	DE2	DE1-SOC
18×18 Multiplier	384	35	174

Since the DE2 FPGA is limited in computation resources, a reasonable way of distributing the DSP resources is to implement in each channel 2 first-order IIR filters in a cascaded configuration. One of the IIR filters can be configured with 16-bit coefficients and other with 32-bit coefficients to allow implementation of PII with one high frequency zero and one low frequency zero. Many variations exist, but they all have to work within the resources available in the DE2 FPGA. The ultimate solution is to port existing DE2 implementation to the DE1-SOC to allow a more complex filter.

3.2.7 Comments on Computation Delay

The computation delay through the IIR logic can be combined with the delay through the analog frontend and the ADC/DAC described in Chapter 2 to compute the total latency through the FPGA servo (typically around 200 ns in total). This computation delay is not absolute as a lower computation delay can often be achieved by additional computation resources and additional development time. In an effort to minimize the computational delay, the IIR filter commissioned in this project is optimized to have a delay of 1 cycle at either the maximum frequency allowed for the ADC and the DAC or at twice of this frequency. For example, a single DE2 IIR filter with 16-bit coefficients has 1 cycle of delay at $f_{clk} = 130$ MHz while the ADC and the DAC are clocked at 65 MHz. The f_{clk} for the DE3 32-bit coefficients IIR is lower and various versions of the design are clocked with a single cycle of delay at either 50 MHz or 75 MHz although more optimization would allow an additional increase in the clock frequency. As the result of this optimization, the computation delay is similar or less than the ADC/DAC delay, even when up to 3 IIR filters are cascaded to realize a more complex filter.

The main tool used in reducing the IIR computation delay is Altera's TimeQuest timing analyser. The timing analyzer helps by identifying the limiting factor in computation delay and providing clues to reduce the total delay. Another look at the IIR implementation illustrated in Figure 3.4 in Section 3.2.3 should reveal that the forward paths (the signal paths between the IIR input and the IIR output) and the feedback paths (the signal paths between the past IIR output and the current IIR output) has different computation complexity. The feedback paths have a larger computation load because the scaling factors for the feedback paths, A_1, for example, need to multiple against internally stored values that are wider. This is why it is often the feedback path that has trouble satisfying the timing constraint. As a concrete example, the total timing budget for one IIR filter in the DE2 implementation is 7.7 ns. This is assuming the IIR filter is clocked at 130 MHz. Each feedback path (A_n) is composed of two 18-bit by 18-bit multiplications that takes 3.5 ns (specification is obtained from [35, 5-21]), while working in parallel. An additional summation step must follow the two 18-bit by 18-bit multiplication operation, and this summation step is not present in the case of the forward paths (B_n).

A few optimization techniques were found to be helpful. In one case, the FPGA synthesis planner is allowed to place redundant buffers for a few internally stored values which allow the synthesis planner more freedom in placing the slowest signal paths, and eventually leads to reduction in delay. In another instance, the design of three cascaded IIRs are allowed 4 total clock delays, with the additional clock cycle designated to allow the signals to travel from the IIR output to the FPGA pins. Having an additional delay allows the maximum frequency of this block (derived from the same master clock as that derived for the ADC and DAC driver) to be higher than if the IIR filters were designed to have 3 clock delay (each IIR filter takes up one clock delay). This allowed the servo logic to be clocked at twice of the clock frequency for the ADC and the DAC rather than at the same frequency, so as a result the total delay is reduced by 1/3. Admittedly, these examples of reducing the computation delay may not be feasible in all optimization scenarios. Many optimization strategies are discussed in text on reducing delays in FPGA design [36].

3.2.8 Comments on PII

This section comments on some observations made during the use of a PII filter. Some of these observations, regarding the offset at the output of the PII filter specifically, is inherent to the PII filter and should be present regardless of the underlying implementation (whether is digital or analog). The observations regarding the strange saturation behaviour of the PII is specific to the implementation of the IIR filter by this work and is not present in an analog implementation or likely IIR filters with a different saturation logic.

PII Offset

The IIR filter produces an offset on its output when configured as an integrator. This offset can be observed at the output of a single PI filter as illustrated in Figure 3.13 . Similarly, when two PI filters are cascaded together to form an PII filter, a ramp appears on its output as it is also evident in Figure 3.13. The offset and the ramp at the output of the IIR filters may seem counter intuitive as the input sinusoid contains no offset. However, this behaviour is expected from the integral of a sinusoid over a finite interval. The first order integration of a sine wave has a DC offset dependent on phase of the input. The second order integration subsequently integrates this offset and produces a ramp with a slope that is dependent on the phase of the input sine wave. The simulation shown in Figure 3.13 produces this behaviour after it was first discovered with the FPGA servo hardware. The details of the simulation method are discussed in Section 3.2.4.

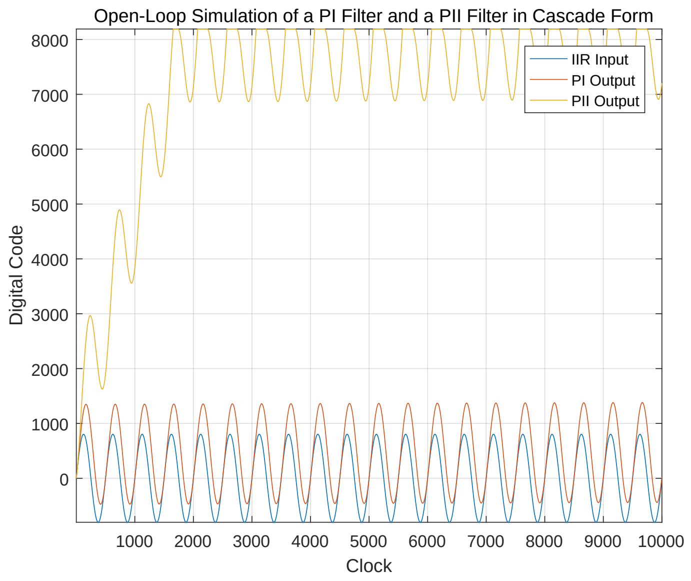


Figure 3.13: The open-loop simulation of the PI filter and the PII filter in cascaded form, showing the offset at the output of the PI filter and the ramp at the output of the PII filter in cascade. As it can be observed in this figure, the input to the filter is a sine wave with no offset.

The ramp at the output of the PII causes the IIR output to quickly “rail to either the minimum or the maximum of the IIR output. This behaviour complicates the open-loop measurement of the transfer function of the PII. In contrast to the inconvenient open-loop behaviour, when the PII is used in closed-loop this ramp is suppressed by the servo logic and does not affect the closed-loop performance of the servo.

An Unusual Saturation Behaviour of the PII Filter

The IIR filter commissioned by this work (as illustrated in Figure 3.4 in Section 3.2.3) has an edge case where instability exists. The instability occurs when a single IIR filter implements a PII filter in the form of a second-order IIR filter. The instability occurs when the output of the IIR filter saturates but the filter fails to hold the output in saturation as illustrated in Figure 3.14 starting roughly at $6.6 \cdot 10^4$ clock in the simulation. The spectral measurement of the output the filter may appear normal if the instability does not frequently occur. However, this is a serious problem in a closed-loop system because when the plant drifts further away than the range of the actuator the servo would fail to hold the actuator at its limit and result in losing the lock. As this instability does not occur when a single IIR filter implements more than one non-zero pole/zero (as opposed to the PII containing 2 poles at $s = 0$), this work resorts to avoiding the instability by limiting each IIR to implement a single order of integration and add IIR in cascade if needed.

To examine the cause of this instability, the exact instability scenario is emulated in simulation, which then provides the values of internal registers at the moment when instability occurs, as shown in the inset of Figure 3.14. The simulation reveals that the problem occurs at the moment when the saturation logic is activated. Activating the simple saturation logic is equivalent to introducing an impulse at the IIR output, with a sufficient amplitude to prevent the IIR output from overflowing. The record of this impulse, remains in the IIR filter and manifests itself as a ramp at the IIR output over time.

A better saturation logic is to modify the internal states of the IIR filter in a coherent manner to avoid introducing this impulse. One way is to scale all the internal states by the same amount that the IIR output is changed by. This is equivalent to modifying internal states of the IIR filter as if all past inputs were smaller in amplitude. We were able to simulate this implementation in C++ and to confirm that this saturation logic fixes the unusual saturation

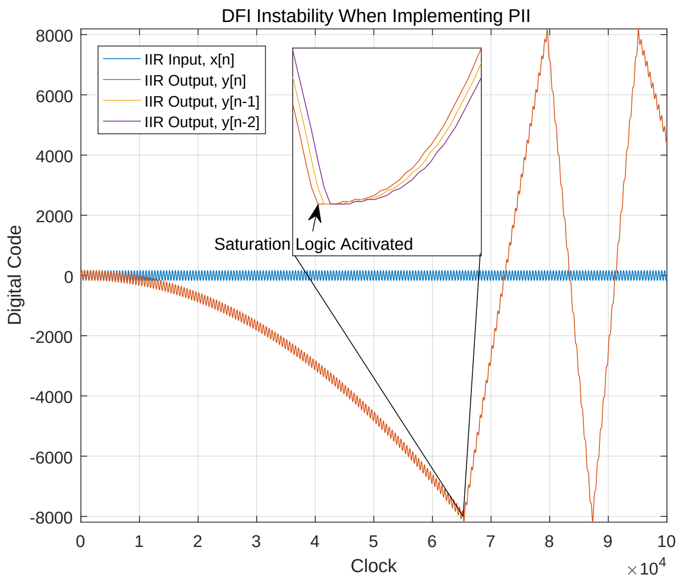


Figure 3.14: The open-loop simulation of a second-order IIR filter configured as a PII filter. The PII in this configuration fails to saturate at its boundary, as opposed to its expected behaviour in Figure 3.13 with a PII filter in cascaded form. Instabilities occur at the clock frame $6.6 \cdot 10^4$ (shown in the inset), $8.0 \cdot 10^4$, $8.8 \cdot 10^4$ and $9.5 \cdot 10^4$.

behaviour for a PII filter or a PI^3 filter when implemented in a single IIR filter.

3.2.9 More on Filter Design

IIR and FIR are two broad classes of digital filters that can be designed with many different strategies. Two such strategies of designing an IIR filter are described in Section 3.2.1 including the use of an exact s-domain to z-domain transformation and the use of a bilinear approximation. This section describes two other methods of designing filter, and it also comments on whether these alternatives are suitable for use in a closed-loop system.

Higher Order of Approximation for an Integrator

In Section 3.2.1, the controller is designed in the s-domain, transformed into the z-domain and then converted into the time domain to produce the IIR coefficients. Another method, less commonly used, is to design the controller directly in the time domain. This method is based on transforming the differential equation that describes the relationship between the filter input and the filter output into a difference equation that can be directly implemented with an IIR or an FIR filter. For example, a pure integrator controller can be described by the following expression where $x(t)$ is the input and $y(t)$ is the output.

$$y(t) = \int_0^{t_0} x(t)dt \tag{3.38}$$

This continuous-time relationship can be converted into a discrete-time relationship using various numerical integration methods including the rectangular integration rule, the trapezoidal integration rule, or other rules (see, for example, ch.6 of ref [37]). The continuous-time expression evaluated to the lowest order using the rectangular integration rule yields the following difference relationship,

$$y[n] = Tx[n] + y[n-1] \tag{3.39}$$

where T is the sampling period of the discrete-time system. Similarly, the next higher order approximation using the trapezoidal integration rule results in the following equation.

$$y[n] = T(\frac{x[n] + x[n-1]}{2}) + y[n-1] \tag{3.40}$$

It is worth noting that this method yields the same result as the s-domain equivalent of the differential equation followed by a transformation into the z-domain using the bilinear approximation (refer to Section 3.2.1).

Like the rectangular integration rule, the trapezoidal integration rule has a residual error when used to integrate smooth functions. The residual error can be reduced with a second order approximation of integration, called the Simpson's Integration rule. By following this rule, the differential relationship in Equation 3.38 can be transformed into the following relationship.

$$y[n] = T(\frac{x[n] + 4x[n-1] + x[n-2]}{3}) + 0 \cdot y[n-1] + y[n-2] \tag{3.41}$$

In preliminary tests of the FPGA servo, this Simpsons Integration rule relationship was combined with a zero around 70 kHz to form a PI filter and was used to form closed-loop lock with the FPGA servo hardware. The servo system was able to lock with this PI filter but exhibited no obvious advantage over the trapezoidal integration rule. Rather, compared to the PI filter that follows the trapezoidal integration rule, a PI filter that follows the Simpson's integration rule increases the loop delay by one clock cycle and takes up more computation resources.

To conclude, this study is meant to illustrate that transforming a continuous-time relationship into a discrete-time relationship by converting the integral or differential equation (Equation 3.38) into difference equations (Equation 3.39, 3.40 and 3.41) is a valid method of producing IIR coefficients for use in the servo logic. This study also shows that a closed-loop control system is rather forgiving with integration error and that reducing the integration error does not seem to result in a significantly better closed-loop performance. The later observation is reminiscent of the observation made in Section 3.2.8 where the PII offset error does not have a detrimental effect on the closed-loop control performance.

IIR vs FIR in Implementing a Phase Compensator

To reduce the impact of the loop resonance in a closed-loop system, the use of a lag-lead filter is investigated in detail in Section 4.2.6. Before arriving at the decision to use the lag-lead filter, the work also considered other types of digital filters to form a bandstop filter to be used along with a PII filter to cancel the loop resonances. Two other types of bandpass filters are considered. They are an elliptic bandstop filter that has an equivalent form as an IIR filter and an FIR bandpass filter designed via the windowing methods. To keep the comparison fair, all the bandstop filters are designed with a centre frequency of 0.1 f_{clk} where the f_{clk} is the clock frequency of the filter. Their frequency domain response (transfer function) is shown in Figure 3.15 along side with the lag-lead filter, which is designed via the bilinear transformation.

One filter candidate is a second order elliptic bandstop filter with a target stop-band between $0.05f_{clk} - 0.2f_{clk}$. The filter is computationally equivalent to a fourth order IIR filter. One unacceptable feature of the elliptic filter is the sign change at each node of the filter, as shown in Figure 3.15 at approximately $0.07f_{clk}$ and $0.14f_{clk}$. This means that the filter has different signs in frequencies between $0.07f_{clk} - 0.14f_{clk}$ than all other frequencies, making it impossible to avoid positive feedback for all frequency. This positive feedback is undesired for closed-loop systems as it often leads to instability.

The other candidate is a 30th order FIR bandstop filter with a target stop band between $0.05f_{clk} - 0.2f_{clk}$. Although the computation complexity of FIR filter is not directly compa-

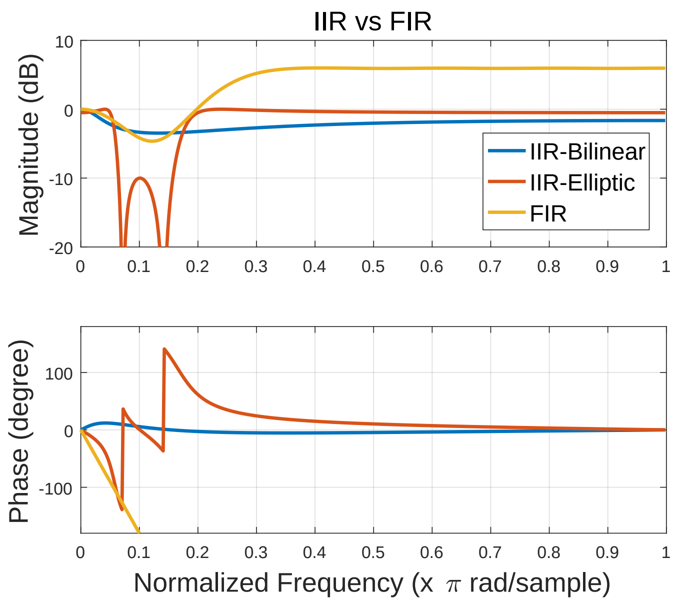


Figure 3.15: Comparison of three different digital filters all implementing a notch filter with center frequency at $0.1 f_{clk}$. Only the bandstop filter designed with the bilinear method (lag-lead) is suitable for used in closed-loop, because the elliptic bandstop filter changes sign at 0.7 f_{clk} and 0.14 f_{clk} and the FIR bandstop filter has excessive delay.

rable with an IIR filter, the comparison can be made by assessing the amount of computation resources used by the two filters. In this way, a 30th order FIR filter in this study is found to use a similar number of 18-bit by 18-bit multipliers as a 9th order IIR filter. Besides the large computation footprint, the FIR bandstop filter also has a large delay that scales with the order the of the filter (the order is 30 in this case). Despite these disadvantages, the FIR filter can, nevertheless, be used in a closed-loop controller and should be considered if a better alternative is not available.

As the remaining candidate, the IIR filter designed via the bilinear method is composed of two poles, at 1 MHz and 8 MHz and two zeros, at 2 MHz and 4 MHz and converted to z-domain with a clock frequency of $f_{clk} = 50$ MHz. Although the IIR filter obtained via the bilinear method has a wide attenuation band and attenuates much less within the band, as shown in Figure 3.15, it has a stable phase response and uses a reasonable amount of computation resources as only a second order IIR filter. As a result of this comparison, the lag-lead filter designed with the bilinear method was chosen, and it is used as the compensator to reduce loop resonances. A detailed discussion of its closed loop performance is provided in Section 4.2.6.

3.3 Arbitrary Waveform Generator

Waveform generation is a ubiquitous and versatile tool in an AMO laboratory. The need for waveform generation ranges from dithering laser frequencies to modulating laser intensities. It is convenient feature for a laser servo to combine the function of stabilizing the laser with waveform generation and this can be easily implemented in an FPGA servo. Additionally, the waveform generation capability of an AWG can be used to implement new features, such as lock-in detection and system identification. This section describes the design of an AWG produced by this work, how the AWG can be program, its residual error, and other characteristics of the module.

3.3.1 Implementation

With flexibility and speed as the main design requirement, the waveform generator is implemented as a fourth order polynomial with programmable coefficients as illustrated in Figure 3.16. The timescale of the polynomial is controlled by a ramp generator (left module). The shape of the waveform can be controlled by the coefficients to the fourth order polynomial (right). All the ramp parameters and the polynomial coefficients can be controlled from a PC via USB. A soft-core MCU in the FPGA receives the parameters through the serial link and then transfers the parameters to the AWG. The address where the parameters can be accessed is provided in Appendix C.

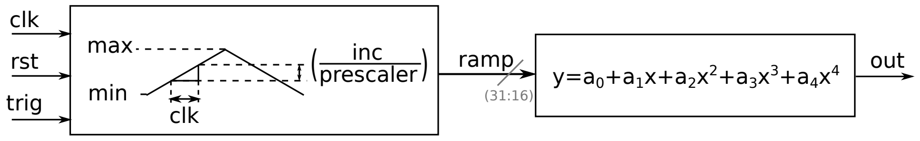

clk
rst
trig
max
min
clk
$\int \left(\frac{\text{inc}}{\text{prescaler}}\right)$
ramp
(31:16)
$y=a_0+a_1x+a_2x^2+a_3x^3+a_4x^4$
out

Figure 3.16: A block diagram of the arbitrary waveform generator, implemented as a counter followed by a 4th order polynomial. The counter produces a 32-bit triangular waveform and can be controlled through internal registers: “max”, “min”, “inc” and “prescaler”. The triangular wave at the output of the first module has a range between “max” and “min” control and a slope of $f_{clk} \cdot (inc/prescaler)$. The polynomial coefficients are implemented as 16-bit FP numbers with 10 fractional bits. Signal responsible for external trigger are “rst” and “trig”.

Synchronous control of the AWG is accomplished with an external trigger signal. This allows the waveform generator to operate in synchronization with the rest of the experiment.

3.3.2 Coefficients, Residual Error and Verification

A simple configuration of the AWG is an triangular wave generator, where the coefficients are programmed as $a_1 = 1$ and $a_n = 0$. The slope and the amplitude of the ramp is controlled by the counter parameters, like “ramp min”, “ramp max”, “increment” and “prescaler”. Configuration of an arbitrary waveforms is accomplished by adjusting the AWG coefficients A_0, A_1, A_2, A_3 and A_4. They are the FP representation of the coefficients of a fourth polynomial and are defined like the following.

$$A_n = a_n \cdot 2^R, \text{ R=fraction resolution=10} \tag{3.42}$$

The coefficients can be found by a least square fit to fourth order polynomial. The sample coefficients for implementing sinusoidal or Gaussian waveform are listed in Table 3.5. Because the underlying implementation in the AWG is in FP arithmetic, the coefficients here are also in the format of FP number with 10-bit fraction resolution.

Table 3.5: The best fit parameters to implement a sine or a Gaussian waveform. The AWG is implemented as 4th order fixed point polynomial. The parameters related to the ramp (“min”, “max” and “increment”) are 32-bit FP numbers with 16-bit fractional resolution. The parameters for the polynomial, A_0, A_1, A_2, A_3 and A_4, are 16-bit FP numbers with 10-bit fractional resolution

	Ramp Min	Ramp Max	Increment	Pre-scaler	A_0	A_1	A_2	A_3	A_4
Gaussian	0	0x800000	0x10000	1	13	899	-1522	1825	-372
Sine	0	0x4000000	0x10000	1	4096	1000	-27415	18277	0

A small but significant residual error remains when fitting the target waveform to a polynomial, as it is shown in Figure 3.17 for both the sine wave and the Gaussian wave. When a fourth order polynomial is used, the residual error in fitting to a sine or a Gaussian waveform is about 2% of the total amplitude of the waveform. Although increasing the order of the polynomial decreases the residual error, the cost of computation resources increases rapidly as the benefit of a smaller residual error diminishes.

Another potential issue is that the residual error is not smooth at the boundary of the polynomial. This can be observed at the 0th, 500th, 1000th clock frame in Figure 3.17. This is because the least square method is used for fitting the waveform to a polynomial. If a fitting

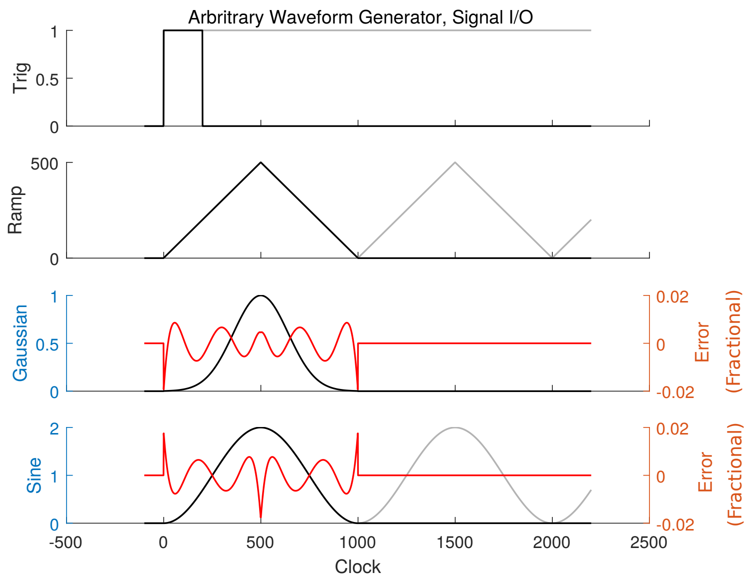


Figure 3.17: The sine and Gaussian waveform output of the AWG, with their respective residual error. The AWG has two trigger mode, a single trigger modes (in black) and a continuous-trigger mode (in grey).

method is based on the Laguerre polynomials with smooth boundary conditions, the AWG output would not exhibit this behaviour.

The AWG implemented in this work has a verification scheme, where the HDL description of the AWG is compared against another description of the same algorithm programmed in MATLAB. The MATLAB description of the AWG anticipates the computation precision of the HDL implementation and no discrepancy is expected from the two sets of implementation. The verification scheme is in the spirit of modular testing already described in previous sections and it is meant to produce self-documenting and reusable code.

3.3.3 Alternative Designs

An AWG with an underlying polynomial implementation has the advantage of needing very little bandwidth to program since the waveform is generated within the FPGA. Another approach is to use the FPGA as a memory device. This implies generating the waveform on a PC and then uploading it into the FPGA. This work explored this approach and comments are provided below on what was found.

The very first iteration of the waveform generator simply stored the waveform in the on-chip memory bits in an FPGA in the form of Read Only Memory (ROM) and used a counter to step through the data (The syntax of doing this in Verilog can be found at this reference [38, p.295]). This design is incredibly easy to make, but the waveform is read only. Changing the waveform requires access to the FPGA design program and this creates a barrier to update the waveform for other experimenter in the lab.

Another possible implementation of an AWG is to store the waveform in random access memory of the FPGA and to program it from a PC through a softcore MCU. This immediately turns out to be a bad idea on the DE2 FPGA platform, due the lack of a fast communication option between the FPGA and the PC. It is important to keep in mind that the AWG consumes 14-bit data at 50 MHz. The constraint of slow communication interface between PC and the FPGA platform is lifted on newer FPGA platforms and should allow the implementation of a memory-based AWG.

3.3.4 Future Improvements

In general, implementing an AWG in the form of polynomial function has the advantage of being quick to re-configure. However, as the complexity of the waveform increases additional computation complexity is needed. Implementing a polynomial function with higher complexity is exponentially more work as the complexity increases. In the case when truly arbitrary waveform with lots of discontinuities is needed, the memory-based AWG implementation described in Section 3.3.3, may become more favourable.

3.4 Proposal on Feature Expansion

The FPGA servo designed by this work has the potential to become a versatile, control toolbox with wide adoption in AMO experiments. So far, this work has explored the implementation of two of these tools, the IIR filter and the AWG. More features can be added to the FPGA servo over time, and this section is meant to discuss potential implementations for some of these new features, including lock-in detection, auto-lock detection and plant detection. A few of these features have already been demonstrated in published work by NIST [14], and while this can serve as reference design it may not be easily transferable to an Altera platform.

3.4.1 Lock-in Detection

The lock-in technique is a detection technique for weak signals that is widely used in AMO experiments. Some example applications are the PDH technique [39] and the use of lock-in detection in saturated absorption spectroscopy [33]. The theory underlying the operation of lock-in amplifiers is discussed in various texts [40]. In essence, a lock-in amplifier is a modulation and demodulation technique that is based on the orthogonality principles of sine waves. The outcome of this technique is a very sharp notch filter at the modulation/demodulation frequency and this helps to recover signal from a high level of background noise. The implementation of a lock-in amplifier is illustrated in Figure 3.18. In this design, both the in-phase and the 90 degree out-of-phase component is recovered from the lock-in input. Some comments on this type of digital lock-in implementation can be found at a white paper by Zurich tech [41]. Some lock-in implementation does not include the 90 degree out-of-phase component, but we suspect that those types of implementation are suitable for modulation frequency that are low enough that the signal latency through the plant is not sufficient to cause a significant phase shift.

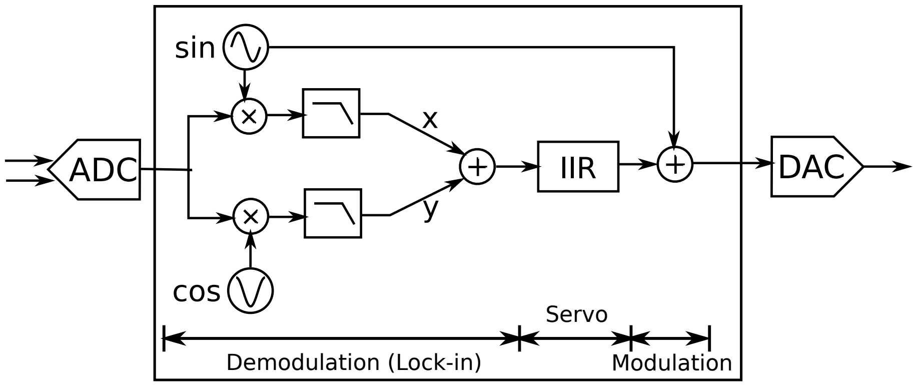


Figure 3.18: An implementation proposal of lock-in amplifier in the FPGA servo. The servo logic is divided in to demodulation, servo and modulation (what is not shown here is the rest of the servo and the laser configured in closed-loop. Both in-phase component (x) and 90 degree out-of-phase component (y) are recovered from the input signal and they must be combined as complex numbers (rather than as simple addition of the amplitudes) to generate the error signal.

The lock-in detection logic can be implemented in the FPGA servo and be used in conjunction with the servo logic without the need for additional electronics. Some of the basic building blocks of a lock-in an amplifier are already implemented in this work. The sine and cosine waveform can be handled by the AWG described in Section 3.3, or by the alternative design of the AWG where the waveform is preprogrammed in ROM. The period of the sine and cosine can be controlled with clock prescalers to the AWG module. In addition, the LP filter that follows the multiplier can be implemented in the form of IIR. The remaining pieces are simple arithmetic and pose no obstacle to implementing lock-in detection.

3.4.2 Auto-Lock Logic

The auto-lock logic, sometimes referred to as the lock-recovery logic, is the mechanism to automatically lock a plant to a pre-programmed setpoint. The use of an auto-lock mechanism in an experiment reduces the manual work need to lock lasers and is very beneficial in an complex AMO experiment. The typical locking method is to manually adjust an offset to the laser controller until the frequency of a free-running laser is within detectable range of a frequency discriminator. This manual adjustment can be replaced by ramping the slow-DAC in the FPGA servo slowly until an error signal is detected. Once the error signal is detected, the FPGA servo can engage the lock by setting the IIR coefficients. The trick to this implementation is to keep the auto-lock logic in the MCU that is built-in to the FPGA. The auto-lock routine is a sequential process and is more intuitive to be implemented in C/C++.

3.4.3 Plant Detection

Plant detection, or sometimes referred to as system identification, is a set of control techniques, widely used in identifying the optimal control parameters in applications such as motor servos.

Some system identification techniques measure the transfer function of a plant in an open-loop configuration. In some implementations, a chirped sine wave is used as the control input of the plant and a cross-correlation or an FFT is used to identify the transfer function.

Another set of techniques configures the plant in closed-loop when identifying the optimal control parameters. The second type of detection techniques is sometimes the preferred method when locking the frequency of a laser or a cavity, because it is often the case that the laser or cavity to be locked is not passively stable enough to allow a thorough open-loop investigation.

Despite the widespread use in other applications, the technique of plant detection in a closed-loop configuration is not common practice in AMO experiments. The addition of closed-loop plant detection would therefore be a valuable tool in an AMO researcher's toolbox.

Chapter 4

Servo Performance and Intensity Stabilization

The hardware design and the firmware design of the FPGA servo is discussed in previous chapters. In both designs, the noise level and the delay of the servo are optimized for a better closed-loop performance. This chapter focuses on the resulting closed-loop performance of the FPGA servo with an emphasis on noise suppression.

For users only previously exposed to analog servos, the closed-loop performance of a digital servo is not obvious. The signal quantization introduced by the ADC and DAC can be especially worrisome. For example, the quantization level of the ADC used in the DE2 system is in theory around 60 uV, with 14-bit resolution and an input range of 1 V peak-to-peak. This level of correction isn't acceptable for most applications, but in practice the computation precision allows the servo to correct to level much lower than the quantization step of the ADC or the DAC. Precision below the quantization step is the result of oversampling and signal averaging and this has been reported in [14], [18].

In the following section, we describe various techniques that are useful in maximizing the level of noise suppression in a closed-loop system. The closed-loop performance is investigated with two setups, one with the FPGA servo forming a closed-loop system onto itself, referred to as the "self-lock configuration" and the other with the FPGA servo installed in an intensity stabilizing setup. The self-lock investigation focuses on exploring the limit of the FPGA servo performance while the intensity stabilization setup focuses on demonstrating the additional improvement that an FPGA servo can offer when compared to an analog servo.

4.1 Self Lock Benchmark

The self-lock test is a convenient method for determining the performance limits of a servo. When used in control applications, the self lock performance of the servo can be combined with the noise floor and the bandwidth of the detector and the actuator to predict the closed-loop performances in a specific application.

4.1.1 Locking Performance

Before using the FPGA servo in any application, it is necessary for the user to confirm that the servo performs as expected. The self-lock test, illustrated in Figure 4.1, provides a convenient benchmark. In the self-lock test shown in Figure 4.1, the FPGA servo is configured as a PII controller with negative feedback and is configured with two corner frequencies at 200 kHz. It is important to set the corner frequencies high to reveal the input noise of the servo for a large range of frequencies but not too high to worsen the loop resonance. Here, the PI corner frequencies (200 kHz) are set to be roughly a decade below the loop bandwidth (2-2.5 MHz) of the servo. The typical input noise level of the servo and the bandwidth of the servo can be observed from the data.

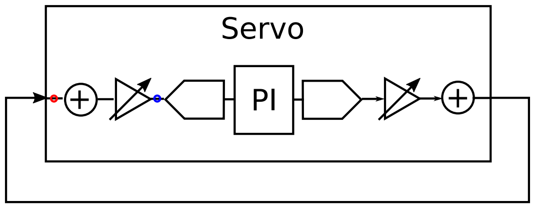


Figure 4.1: The self-lock configuration is a closed-loop feedback system formed by the FPGA servo and no other detectors or actuators. This is a convenient method for evaluating the performance of a servo.

The spectrum can be examined in two frequency ranges, for frequencies less than 200 kHz where noise suppression is active, and for frequencies larger than 200 kHz where the noise suppression is not effective. In the frequency range less than 200 kHz, the spectrum is limited by the input noise of the servo, shown in the self-lock data in Figure 4.2 to be around 11 $\mathrm{nV}/\sqrt{\mathrm{Hz}}$ for the DE3 prototype and around 25 $\mathrm{nV}/\sqrt{\mathrm{Hz}}$ for the DE2 prototype. In the high frequency range, the frequency of the lowest resonance reveals the total delay of the servo, with the following relationship.

$$f_{\text{bandwidth}} = \frac{1}{2T_{\text{latency}}} \tag{4.1}$$

It can be observed from the data, that these two servo prototypes has a loop delay of 277 ns and 227 ns respectively. Additionally, a small resonance can be observed at 200 kHz and it is the closed-loop behaviour of two zeros place at the same frequency. This effect can be avoided if the two zeros of the PII filter is spaced a decade apart.

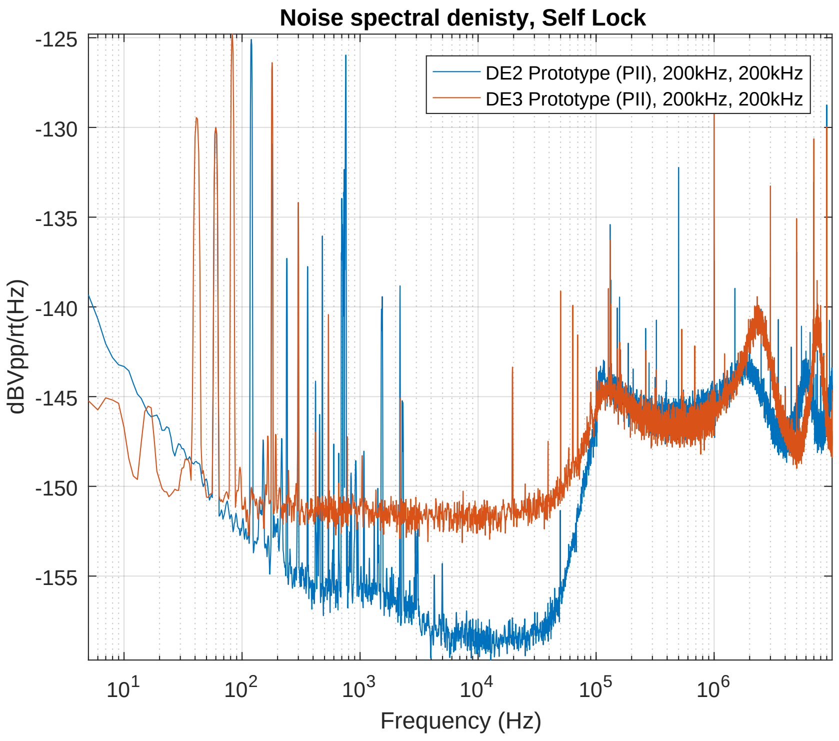


Figure 4.2: The self lock data of the DE2 servo and the DE3 prototype servo. The minimum noise floors of the two servos are -152 dBV/$\sqrt{\text{Hz}}$ and -159 dBV/$\sqrt{\text{Hz}}$. They are equivalent to 25 nV/$\sqrt{\text{Hz}}$ and 11 nV/$\sqrt{\text{Hz}}$ respectively. The self-lock spectrum is optimized according to techniques discussed in sections 4.1.3 - 4.1.6. Additionally, the origin of the spikes in frequencies between 60 Hz and 40 kHz more visible on the DE2 spectrum is covered in Section 2.4.5.

We note that even without a spectrum analyser (eg SR780 and RSA3303A), it is possible to check the performance of a servo in closed-loop. The delay of the servo can be investigated by increasing the loop gain to purposely introduce an oscillation at the loop resonance. The frequency of the oscillation follows the same relationship shown in Equation 4.1. With careful planning, the noise floor of the servo can be investigated with an oscilloscope and analysed either with a built-in FFT function on the scope or downloaded to a computer and analysed off line. It is important to note the noise floor of the oscilloscope in this case, and it is possible that only high resolution oscilloscopes with deep memory buffers are suitable for this task.

If the performance of the servo deviates from normal, then each circuit components can be investigated further with information in the hardware and firmware chapter.

4.1.2 Noise Suppression

Before discussing the practical side of noise suppression, we will first discuss a theoretical model for noise in a closed-loop control system. Figure 4.3 shows a schematic of the FPGA servo configured in closed-loop, with the variable gain amplifiers modelled by frequency independent gain factors G_1 and G_2 and with the offset circuit omitted. It is important to note that although not shown in the diagram, other factors of amplification are present in the loop. Examples include the logic gain, any fixed amplification or attenuation in the servo and amplification in the actuator or the detector. These factors are not included in this model as they only clutter the argument on noise suppression that is presented here. In a more general sense, G_1 can be regarded as the total gain between the input of the closed-loop system and the servo logic, and G_2 can be regarded as the total gain between the servo logic and the output of the closed-loop system.

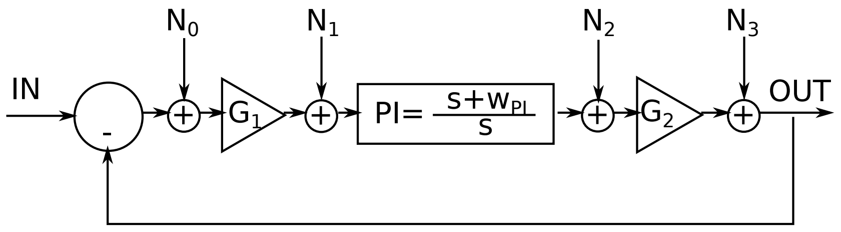


Figure 4.3: The model used for analysing the noise suppression in a closed-loop servo system based on an FPGA servo. For a simpler analysis on noise suppression regarding gain distribution, the non tunable gain in the circuit.

Noise sources N_0, N_1, N_2 and N_3 model noise introduced at different locations in the loop. The output of the loop (including the contribution from each noise) has the following transfer function.

$$\begin{aligned} OUT = IN & \frac{G_1 G_2 PI}{1 + G_1 G_2 PI} \\ & + N_0 \frac{G_1 G_2 PI}{1 + G_1 G_2 PI} \\ & + N_1 \frac{G_2 PI}{1 + G_1 G_2 PI} \\ & + N_2 \frac{G_2}{1 + G_1 G_2 PI} \\ & + N_3 \frac{1}{1 + G_1 G_2 PI} \end{aligned} \tag{4.2}$$

The denominator for any of the terms in the transfer function remains the same regardless of the location of the noise source, but the numerator of each term in the transfer function is the forward path between the location of the noise and the output of the closed-loop system. The transfer efficiency of the noise to the closed-loop output in two limits is summarized in Table 4.1. Understanding the behaviour in the two limits helps with the interpretation of the data and with identifying the location of the noise sources.

Table 4.1: The expected output noise from sources in the loop in two limits assuming $G_1G_2 = 0.5$ and all other components in the loop have unity gain.

$OUT=$	$\omega \ll \omega_{PI}$	$\omega \gg \omega_{PI}$
	N_0	$\frac{N_0}{3}$
	$\frac{N_1}{G_1}$	$\frac{N_1 G_2}{1 + G_1 G_2}$
	$0 \cdot N_2$	$\frac{N_2 G_2}{1 + G_1 G_2}$
	$0 \cdot N_3$	$\frac{N_3}{3}$

The suppression of noise at frequencies $\omega >> \omega_{PI}$ is not complete regardless of where the noise is injected into the loop. This is consistent with our empirical experience of a servo having a limited bandwidth. In contrast, the noise level at frequencies $\omega << \omega_{PI}$ depends on the location of the noise. Any noise introduced after the servo logic, whether it is N_2 or N_3, is completely suppressed regardless of the loop gain. On the other hand, noise injected at the input of the servo (N_0), is written directly onto the output of the closed-loop system. This means that the noise at the first stage of the servo is the most influential on the overall servo performance. The noise N_1 (injected after the variable input gain) has a transfer efficiency onto the output of the closed-loop system that depends on the distribution of the gain inside the loop. In particular, increasing the input gain G_1 while keeping the product of $G_1 G_2$ constant decreases the transfer efficiency of N_1 onto the output, at no cost to the other noise sources. This means that increasing G_1 while keeping $G_1 G_2$ constant beneficial to the noise performance of the closed-loop system, when N_1 is substantial.

Equipped with these conclusions, we are able to continue the investigation in an informed fashion. In the studies that follow, the effect of loop gain, corner frequency and gain distribution on noise suppression is investigated.

4.1.3 Loop Gain

Based on empirical tuning experience, we know that there is an optimal loop gain for suppressing noise. Figure 4.4 illustrates the behaviour of the noise spectrum when the loop gain is varied. We observe an optimal loop gain of 0.5 (equivalent to $G_1 G_2 = 1.2$ due to other factors for gain in the loop) because when the gain is lowered the width of the suppression band shrinks, but when gain is increased the height of the resonance due to loop delay increases. The loop delay resonance is not visible in this plot as it is above 1 MHz. Nevertheless when the height of the loop delay resonance increases, the peak-to-peak amplitude of the error signal also increases until it exceeds the peak-to-peak range of the ADC. When the error signal exceeds the ADC input range, the servo loop adds noise to the quantity being controlled rather than suppressing it. As seen in Figure 4.4, the range of gain where the closed-loop controller switches from noise suppression (at $G_1 G_2 = 1.2$) to noise amplification (at $G_1 G_2 = 1.76$) is small.

4.1.4 Corner Frequency

Similar to the effect of increasing the gain, increasing the corner frequency of the PI filter widens the suppression band, as shown in Figure 4.5. The upper bound of the corner frequency of the PI filter is limited by the bandwidth of the closed-loop. Because a PI filter introduces a 90 degree phase delay in the output for frequencies $\omega << \omega_{PI}$, when corner frequency is too close to the bandwidth of the closed-loop, the delay introduced by the PI filter interacts

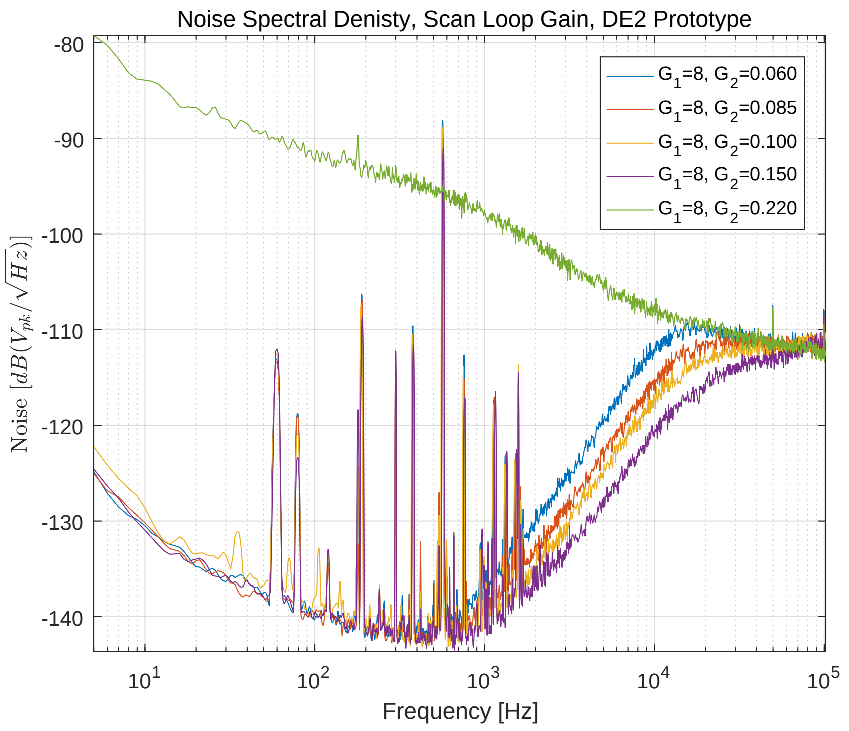


Figure 4.4: The effect of loop gain on noise suppression. As loop gain increases the width of the suppression band also increases. However, once the amplitude of the loop delay resonance (at frequencies above 1 MHz) exceed the input range of the ADC, the closed-loop system amplifies rather suppresses noise.

with the bandwidth of the loop and exacerbates the loop delay resonance. A worse loop delay resonance can cause the closed-loop system to amplify noise when the amplitude of the error signal exceeds the ADC input as it is already seen in the case of an excessive loop gain in Section 4.1.3. To avoid this issue, it is often safe to first set the corner frequency at a tenth of the loop bandwidth and to then make fine adjustment to the corner frequency.

4.1.5 Gain Distribution

Input Gain vs Output Gain

Section 4.1.2 makes the argument that the gain distribution has an effect on the noise suppression through the evaluation of the transfer function of a closed-loop control system, and this section presents data on this effect. Figure 4.6 shows a higher gain at the input of the servo (G_1) while keeping the total loop-gain constant improves the noise floor of the closed-loop system.

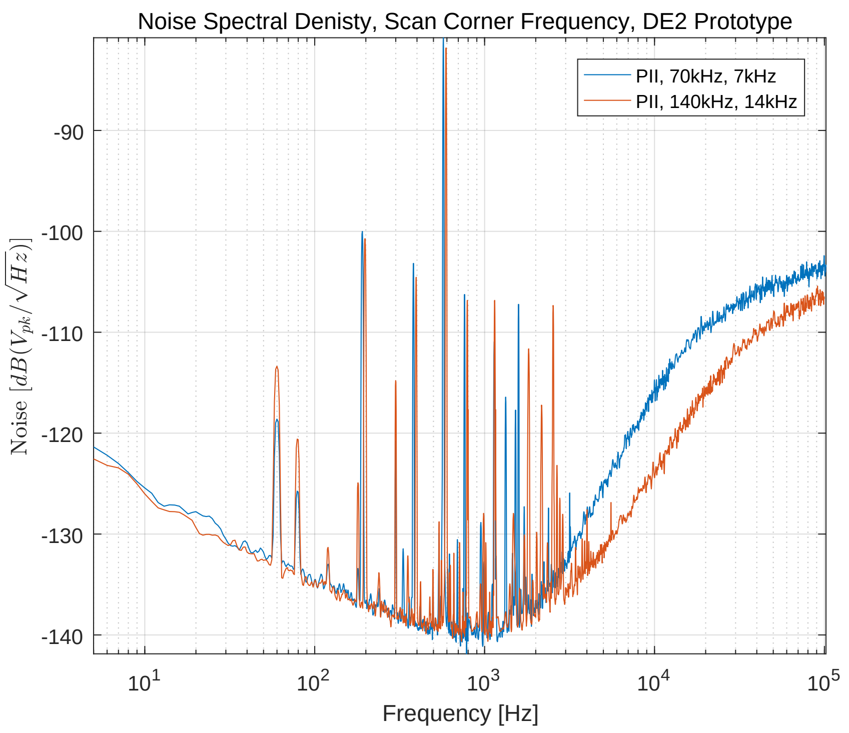


Figure 4.5: The effect of corner frequency on noise suppression. As the corner frequencies of the PII filter increase the width of the suppression band also increases. The limitation of this optimization technique is covered in the text.

This behaviour indicates that the performance of the servo is limited by a noise source after the variable gain amplifier (refer to Section 2.3.3 and Section 2.4.5 for details on the noise floor of the FPGA servo hardware). Unfortunately, G_1 cannot be increased indefinitely. As the input gain G_1 increases, the amplified error signal eventually becomes larger than the input range of the ADC and causes the servo to lose lock. This gain distribution effect on the servo noise floor is also present for an analog servo as well, but the FPGA servo needs special consideration due to the limited the input range of an ADC compared to that of a standard opamp.

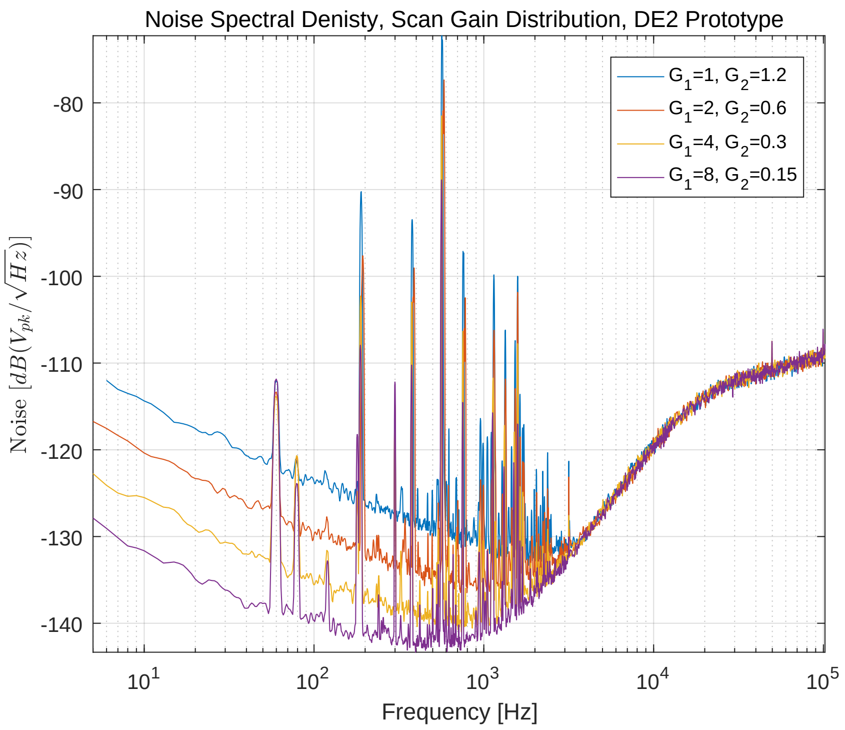


Figure 4.6: The effect of G_1 and G_2 on noise suppression. When loop gain is kept constant, a higher G_1 leads to an improvement in noise suppression in frequencies where noise suppression is limited by N_1 (the ADC) in the noise model.

FPGA Gain vs Output Gain

Similar to the limitation of ADC range, the DAC is also limited and it needs sufficient amplification to correct the noise in the system. As seen before, trading off output gain for input gain improves noise performance; however, the actuation range of the closed-loop system is reduced in the process and this in turn reduces the maximum amplitude that the system can correct for. This scenario can be avoided by decreasing the gain through the FPGA logic rather than at the variable gain stage following the DAC. Reducing the gain through the FPGA logic is simpler with 32-bit IIR coefficients as the pole-zero resolution is high and a reduction of the pole-zero resolution is unlikely to affect the ability of the system to lock. On the other hand, reducing the gain through the FPGA is not as straightforward with 16-bit IIR coefficients as the pole-zero resolution placement is already 7 kHz (details of this implementation is discussed in Section 3.2.3). Figure 4.7 shows a scenario where the lock was able to suppress a higher level of error when the FPGA gain is traded off for output gain.

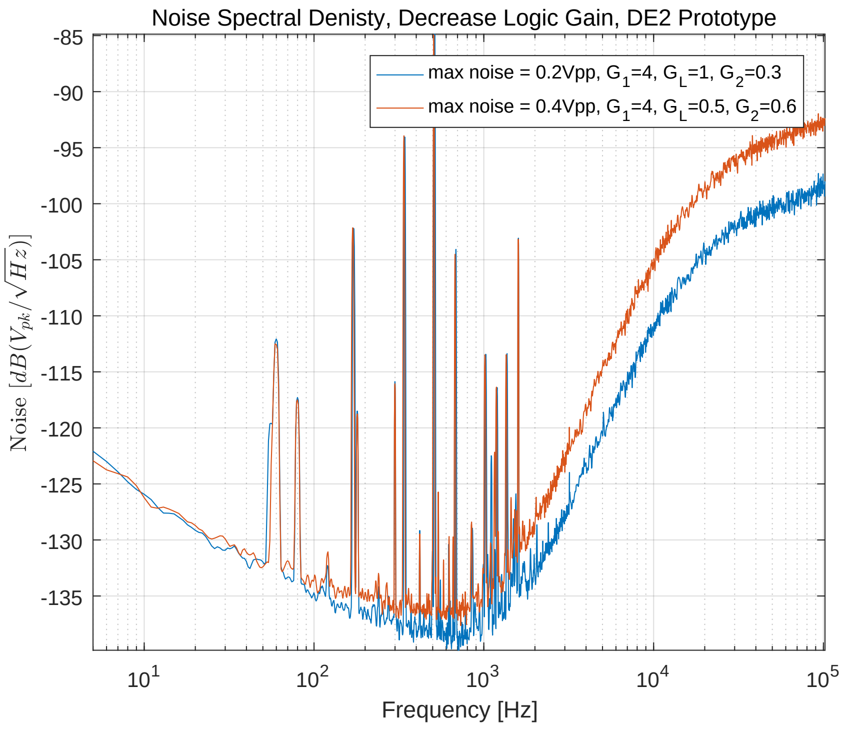


Figure 4.7: The effect of G_2 and logic gain on noise suppression. When loop gain and G_1 are kept constant, decreasing gain through the servo logic in the FPGA and increasing G_2 (the gain between the DAC and the servo output) increases the maximum amplitude of noise that the closed-loop system can correct.

4.1.6 The Optimal Strategy

Based on the study, we can conclude that the optimal tuning strategy is to choose the total loop gain and the poles and the zeros suitable for the laser system and then redistribute the gain within the servo for the optimal noise performance. (We follow this practice in the later stages of this project.) The tuning example discussed is for a PI filter but all the same principles apply for a different transfer function. Thus, compared to an analog servo, the FPGA servo has an additional step of choosing the gain distribution. Here, the optimal distribution can be obtained by first deciding on the output amplification needed to give the actuator a sufficient range and then maximizing input amplification without letting the amplified error signal saturates the ADC. The gain through the FPGA can then be used as a free parameter to keep loop-gain constant.

The best experimental result for suppressing white noise with a 30 MHz bandwidth, as shown already in Figure 4.7 is in excess of 40 dB. The suppression of single tone noise within the suppression band can be much better and is estimated to be as high as 160 dB. This is under the assumption that sufficient gain follows the DAC to amplify the correction to 10V and the input gain is at least 8 to obtain a input noise floor of 100 nV/$\sqrt{\text{Hz}}$. The difference in the ability of a servo to suppress white noise and single-tone perturbation can illustrate the constraints of a digital servo. In the case of white noise, a large band of the noise can not be suppressed so the residual error is large which in turn limits the input gain. The single tone perturbation is another extreme where the perturbation can be completely suppressed and little residual noise is left to limit the input gain so the maximum perturbation that the servo can suppress is set by the DAC range and the amplification that follows. A realistic application will likely involve a noise spectrum somewhere between the two extremes, and either the ADC range or the amplification after the DAC can be the limiting factor for the noise performance.

4.2 Intensity Lock

In this section, the FPGA servo system is applied to an intensity stabilization application. This is a different application from the use of an FPGA to provide PDH based frequency control of a laser previously demonstrated in various published works [9, 14]. The intensity stabilization builds upon the FPGA servo discussed in previous sections, with the addition of intensity modulation and measurement hardware. This section first covers the design of the hardware and then describes some optimization techniques used to improved the closed-loop performance of the intensity stabilization system.

This study also focuses on demonstrating the performance of the FPGA servo in comparison to and beyond that of a high-performance analog servo. The ease for such improvement on the same piece of hardware is all thanks to the ease of reprogramming the FPGA and its ability to emulate complex transfer functions. Three servos, the Vescent D2-125, the DE2 servo prototype and the DE3 servo prototype are used to make the baseline comparison. The DE2 prototype is equipped with a 16-bit IIR and the DE3 prototype is equipped with a 32-bit IIR (refer to Chapter 3 for details on IIR implementation). Two aspects of the servo performance can be observed from the study. First, the two servos have similar noise floors at the highest gain setting, both are in the 10 $\mathrm{nV}/\sqrt{\mathrm{Hz}}$ range. Secondly, the bandwidth of the closed-loop system is dominated by the intensity control hardware (i.e. the plant) with a delay of 400 ns, making the bandwidth difference of the analog servo (at 10 MHz) and that of the FPGA servo (2.5 MHz) less consequential.

4.2.1 Motivation

The shot-to-shot variation between experimental cycles in a scientific investigation often directly affects the quality of the result. The laser intensity is a factor that can affect a cold-atom experiment in many stages, including changing the the population of the initial atomic sample, changing the AC Stark shift relevant for spectroscopy and changing the exposure intensity of atom cloud during absorption imaging [42]. The dependence of cold-atom experiments on this quality makes active stabilization of intensity a viable route in making the experiments robust.

The hardware developed under the umbrella of intensity stabilization has another use and that is intensity modulation. Wide bandwidth modulation of intensity also has use in cold atom experiments. Example applications include dynamic formation of dipole trap [43] and transformation of the internal quantum state population [44].

4.2.2 A Note About Noise Units

In previous sections, the electronic noise level of the servo and the associated electronics is evaluated in $\mathrm{nV}/\sqrt{\mathrm{Hz}}$. In this section, the intensity stability of the laser is evaluated in

Relative Intensity Noise (RIN), with the following relationship.

$$RIN = \frac{\left\langle \Delta P(t)^2 \right\rangle}{P_0^2} \tag{4.3}$$

Here, P is the laser power and the square of which (intensity) is proportional to the detected electrical power, P_E, at a photodetector [45]. This gives rise to the expression for RIN as the following.

$$RIN = \frac{\Delta P_E}{P_{E0}} = \frac{(V_{AC})^2}{(V_{DC})^2} \tag{4.4}$$

Here, the direct observables V_{AC} and V_{DC} can be obtained from an Electrical Spectrum Analyzer (ESA) and an multimeter respectively.

Here, the RIN is an evaluation of the closed-loop intensity stabilization system rather than the just the servo alone. This means that the conversion from servo noise floor to RIN of the laser highly depends on the noise floor of the detector and the amount of light being detected. The implementation detail of this intensity stabilization system is covered in the following section.

4.2.3 Implementation

Intensity stabilization is achieved with an AOM driven by a Radio Frequency (RF) source with a variable power and a Si biased amplified photodetector. The AOM allows manipulation of laser intensity by splitting the beam with a variable diffraction efficiency controlled by the RF power sent to the AOM. The latency of the AOM depends on the proximity of the laser beam with respect to the piezo element that creates the moving diffraction pattern in the crystal and is typically in the 200-400 ns range. The Si detectors used are the Thorlabs DET10A photodetector and the Hamamatsu S5971 Si photodiode biased with well regulated 5 V. The DET10A and the S5971 detectors respectively have the Noise-Equivalent Power (NEP) of 1.2×10^{-13} W/$\sqrt{\text{Hz}}$ and 7.4×10^{-15} W/$\sqrt{\text{Hz}}$ and bandwidth of 350 MHz (calculated from the rise-time of 1ns) and 100 MHz.

For a more thorough study, additional intensity noise is injected into the system via a second AOM whose RF power is modulated by a VGA driven by a DS345 function generator that is configured to generate white noise with bandwidth of 30 MHz. The system that artificially injects intensity noise resembles a laser with an inherent noise of the same level. It may be counter-intuitive at first, but the latency through the noise injection AOM and the location where the noise is injected (before or after the correction AOM) does not affect the noise suppressing capability of the system.

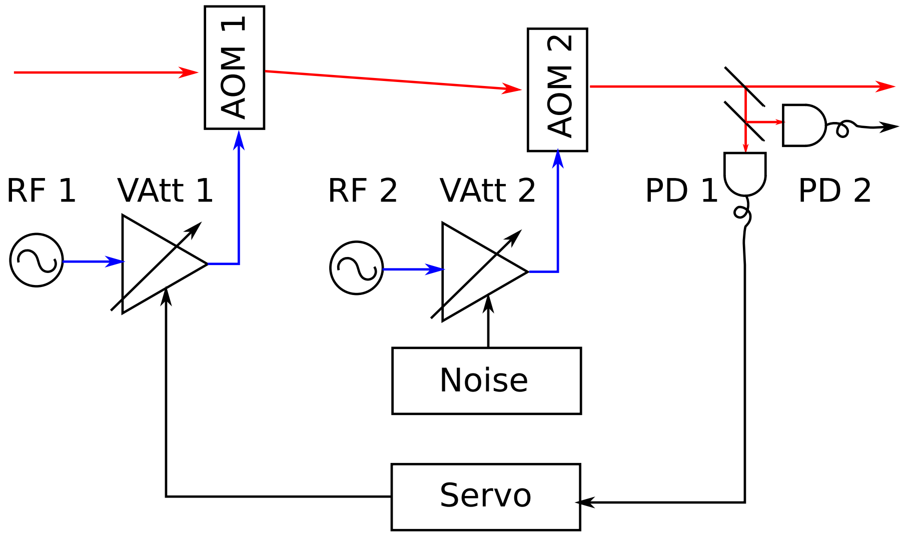


Figure 4.8: The opto-electrical system used to study intensity stabilization. The intensity stabilization consists of a RF source (RF1), a variable attenuator (VAtt1), an AOM (AOM1), a photo-detector (PD1) and an analog or an FPGA servo. To study the performance of the intensity stabilization system on various level and type of intensity noise, a noise modulation system is constructed from a RF source (RF2), a variable attenuator (VAtt2), an AOM (AOM2) and a noise generator. Finally, the out-of-loop detection of intensity noise of the system is accomplished via a second photo-detector (PD2).

Actuator

The AOM is a diffraction device with a diffraction grating dynamically formed inside an acousto-optical material by a compression wave. The compression wave is generated by a piezo mounted at one end of the crystal and travels at the speed of sound. The piezo is often driven from an RF source through an Inductor-Capacitor (LC) resonator. At different RF power levels, the compression wave forms diffraction grating of different contrast in the crystal.

The RF power is controlled using a fast analog switch based on the MAAVSS0006, which is only available as a single IC. For this work we designed a PCB with coaxial connectors as the interface to the IC. The core schematic of the design is shown in Figure 4.9 with additional details in Appendix B.

Figure 4.10 shows the relationship between the control voltage and the transmission efficiency. This analog RF switch can only accept a rather low input power (10 dBm), so it is often the first component in the RF amplification chain. A few observations can be made from the data in Figure 4.10. For example, the leakage when the switch is off, typically in the -60 dB to -80 dB range, suggests the need for a mechanical shutter when the light level must be fully off. In addition, we observe variation between devices, suggesting that calibration maybe needed for each device. Finally, we note that the transition between on and off is not linear but appears to be roughly linear in dB in the transition region. This means that the closed-loop system using this intensity modulator may not maintain a constant gain for all set points and open-loop use of this modulator may need special consideration on the nonlinearity of the modulator.

In this design of an analog RF switch, speed is a major consideration. The variable attenuator is chosen for the fast rise and fall time specified to be 10 ns in the datasheet [46]. Since the variable attenuator is to be used as part of an intensity modulator, its transfer function is evaluated. The transfer function shown in Figure 4.10 shows the performance of the analog RF switch together with an AOM. In this particular case, the transfer function has modulation bandwidth of 3 MHz, and a latency of about 400 ns observed from the phase response plot. In the case of the latency, the time it takes the compression wave to travel to the laser beam is the dominant source of delay with every 1 mm of distance between the laser beam to the piezo driver costing about 230 ns (calculated from the speed of sound in Tellurium Dioxide (TeO_2) [47]). The latency time causes the closed-loop bandwidth of an intensity stabilization scheme with AOM to be limited to about 1 MHz, although the modulation (open-loop) bandwidth is much higher. The modulation bandwidth of 3 MHz shown in this transfer function data is likely the result of the finite beam size coupled with the speed of sound in the crystal. We note that these data were taken with an OP37 opamp in the which was subsequently replaced with AD829 with a faster response. The AD829 was observed to have a bandwidth of 90 MHz. It would be interesting to see if the modulation bandwidth increases from this change.

With use in closed-loop, some optimization is done to reduce the total loop latency. Besides placing the focus of the laser beam close to the piezo driver, the BNC cable lengths are minimized and this led to a small reduction in delay.

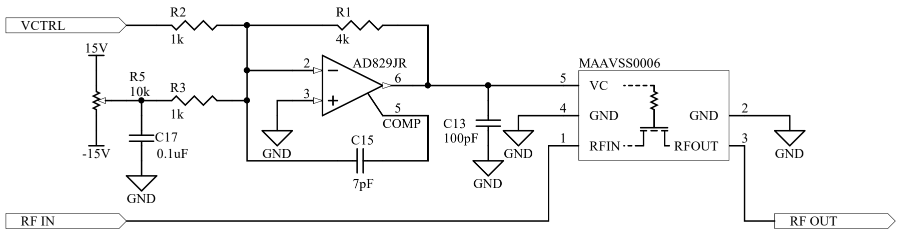


Figure 4.9: Diagram of the variable attenuator circuit with the AD829 configured in current compensation mode. A snapshot of the layout and additional details can be found in Appendix B.

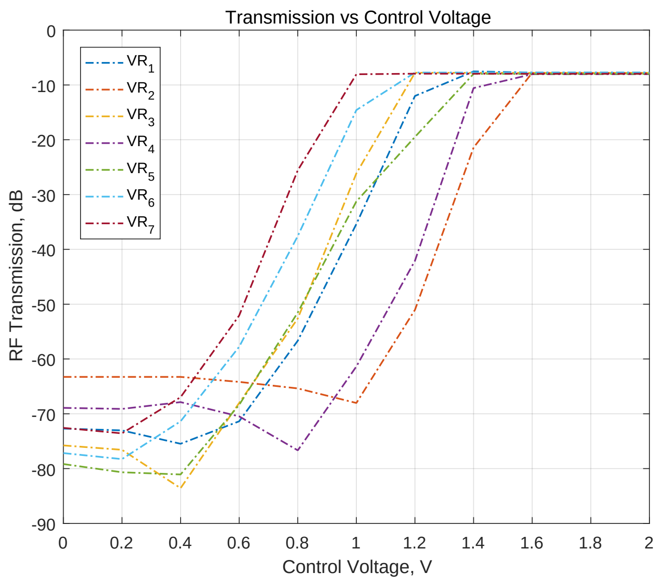


Figure 4.10: Control voltage versus transmission through the variable attenuator. It is important to note the variation in the response to input voltage and in maximum RF suppression.

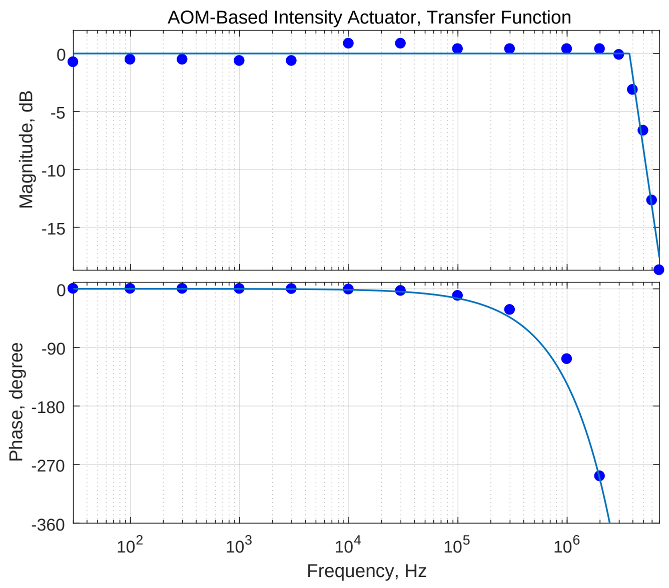


Figure 4.11: Transfer function of the variable attenuator + AOM + PD system. The loop delay is about 400 ns. As the delay is dominated by the distance between the laser beam and the ultrasonic source in the AOM, this delay is modified at various point of the intensity stabilization study due to small positional changes but is kept to a range of 200-400 ns. The modulation bandwidth is in excess of 3 MHz. This opens up the use of this system in high speed intensity modulation.

Detector

As the interpretation of the closed-loop transfer function in Section 4.1.2 shows, the noise floor of the closed-loop system, especially at low frequencies is highly dependent on the noise level of the first stage. This means that the noise level of the detector sets the lower bound on the closed-loop performance. The intensity stability achieved by this work is the result of optimizations made to reduce the noise level of the detector as much as possible.

The major contributor of noises at the detector is the electronic/photon shot noise, the thermal noise of the charge carriers, the op-amp flicker noise often referred to as the 1/f or pink noise, and the easily over-looked power supply noise that often couples into electronic circuits. The shot noise expressed for electricity

$$\langle i^2 \rangle = 2eI\Delta f \tag{4.5}$$

where e is elementary charge and Δf is bandwidth, is hard to work around without increasing the intensity of indecent light onto a photo-detector, as the shot-noise is the by-product of the particle-like property of both the electronics and photons and is proportional to square root of the number of particles. Once the intensity is fixed, the impedance that is driven by the photodetector (modelled as a variable current source) and the gain after the photodetector are both free design parameters that affect the SNR of the detector. An excellent source on design principles can be found here [48]. In the work conducted here, the noise floor of the detector is much lower than the noise floor of the servo, with detector noise floor at the 1 $\mathrm{nV}/\sqrt{\mathrm{Hz}}$ level (assuming 1 mA average photocurrent and 50 Ω impedance). This suggest that additional gain at or after the detector can improve the overall performance, by a factor of 100 until the photon/electronic shot noise becomes the limiting factor.

Another technique for identifying noise sources is to apply the characteristic shape of the noise spectrum of various noise type. This is a useful technique when the only direct observable is the spectrum of the noise from an ESA. For example, both shot noise and thermal noise are constant in frequencies[49]. The op-amp flicker noise or the 1/f has a slope of -20 dB/decade at low frequency [50, p.84]. Noise with any other characteristic shapes whether is it is in the form of spikes or a broad spectrum are likely a result of poor practice in the form of improper ground, noisy power supply or inadequate shielding of signals.

In the implementation scheme shown in Figure 4.8, intensity is monitored by two photodetectors with only one detector in the feedback loop of the system. This technique is often referred to as out-of-loop detection, and it is helpful for detecting whether the noise floor of the detector is higher than the servo. When adding the out-of-loop detection, we focus the laser beam onto both detectors and keep the path difference minimal. The in-loop spectrum looks identical to the out-of-loop spectrum, but to eliminate differences at low frequencies, we found it necessary to reduce the air circulation near the detectors.

4.2.4 PII Comparison

In comparing the performance of the FPGA servo against analog servos, we use a well-reputed commercial analog servo by Vescent as a benchmark (D2-125). For this comparison, the FPGA servo needed to work within the constraints set by the analog servo. The D2-125 model by Vescent, implements a Proportional double-Integral and Derivative (PIID) with configurable corner frequencies.

In the interest of fairness in comparing the three servos, the total loop gain of each closed-loop system is kept the same after we determined the optimal loop gain. The corner frequencies of the servos are chosen according to the loop delay of each system. In the case of the Vescent servo, with the higher servo bandwidth of 10 MHz, a higher corner frequency at 100 kHz is possible without worsening the oscillation at the loop delay resonance. In the FPGA servo, with a lower servo bandwidth of 2.5 MHz, a corner frequency of 70 kHz is chosen due to the higher loop latency. As it can be observed in Figure 4.12, the width of the suppression band is not substantially different between the systems formed by different servos, since the latency of the AOM is the limiting factor to the closed-loop performance.

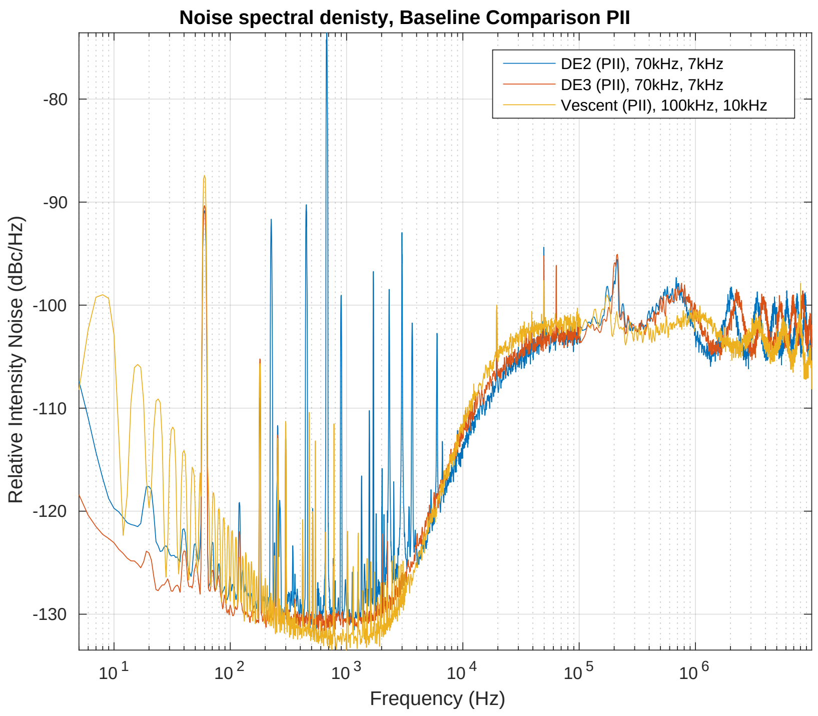


Figure 4.12: Alternative baseline comparison of DE2, DE3 and Vescent servo for the intensity stabilization study. All servos realize a PII transfer function to match Vescent. The gain and corner frequencies of all servos are tuned to optimize the noise suppression. The origin of the spikes in frequencies between 60 Hz and 40 kHz visible on the DE2 spectrum is covered in Section 2.4.5.

To complete the controller's transfer function, the corner frequency for the second integrator is chosen to be a decade below the corner frequency of the first PI. This was done because when two zeros are too close in frequency, the closed-loop system pushes the zeros away from the real axis in the s-domain (this can be observed in a root locus diagram) and this can result in oscillation. The contribution of the derivative component is not studied here and and thus was not enabled.

The spectrum shows that the FPGA and the Vescent servo have similar noise floors of roughly 10 $\mathrm{nV}/\sqrt{\mathrm{Hz}}$ at 1 kHz. With the exception of the small difference in the frequencies of the loop resonances, the noise spectrum capability of the two types of servo is nearly identical. This is not surprising as similar transfer function are implemented in the servos.

4.2.5 PI^3 Improvement

As a natural extension of the PII servo, a PI^3 (3 integrator in cascade) was attempted on the intensity stabilization system. This is accomplished by cascading three IIR filters in the FPGA servo, all configured as integrators. The difference between the PII filter and the PI^3, shown in Figure 4.13 is that it widens the suppression band without adding oscillation at the loop delay resonance. This is an example of a filter that cannot be easily implemented in an analog servo without additional hardware.

4.2.6 PII + Lag-Lead Improvement

The lag-lead filter is another example of a filter that can not be easily implemented in an analog servo. Unlike the PI^3 filter, which can still be constructed by chaining multiple existing analog servos, the lag-lead filter is impractical to implement as an analog circuit due to its large number of tunable poles and zeros.

The lag-lead filter is effectively a notch filter composed of two poles and two zeros. Similar to the lead-lag filter (a bandpass filter), the lag-lead filter can be introduced into a transfer function to modify the transfer function locally, as illustrated by their transfer functions shown in Figure 4.14. The lag-lead filter can be used to suppress loop resonance when implemented with a center frequency that is the same as the frequency of the loop resonance. Because the lag-lead filter reduces the gain locally at the loop resonance, the amplitude of the loop oscillation is reduced. Another benefit of a notch filter (the lag-lead filter) at loop resonance is to allow the loop gain to be increased and thus results in a wider suppression band for noise. Both

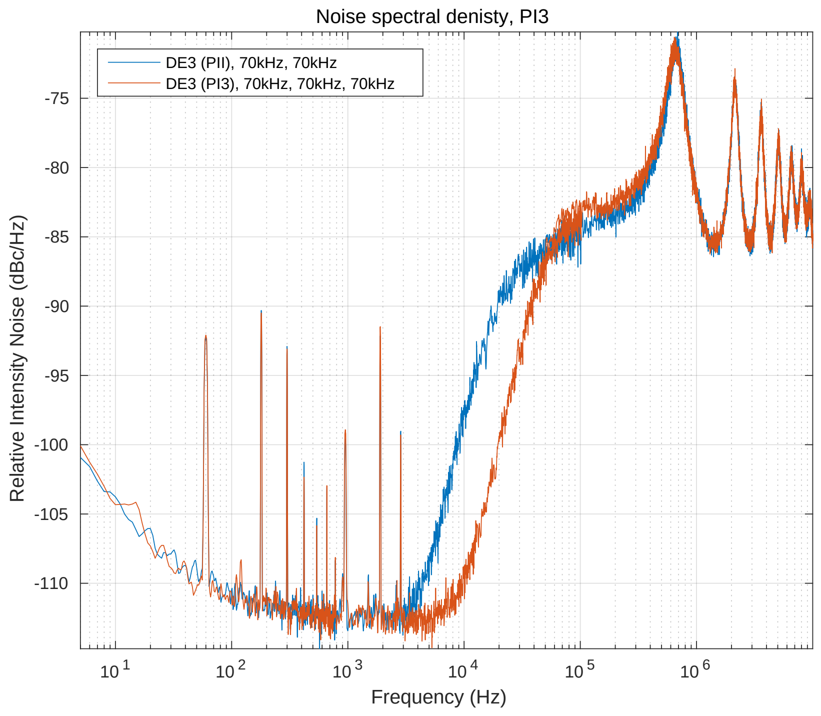


Figure 4.13: Demonstration of the DE3's ability to implement an additional integrator and its effect on noise suppression. The corner frequency of 70 kHz is used for all 3 poles. The suppression slope changes from 40 dB/decade to 60 dB/decade as expected.

effects are illustrated in Figure 4.14, where the DE3 prototype servo with 32-bit IIR coefficient implements a PII + lag-lead filter with total latency of about 200 ns.

The lag-lead filter and the PI^3 are merely examples of advanced servo transfer function that an FPGA servo can implement without costing hardware development time or servo performance. This also means that the use of FPGA servo can lower the barrier for exploring the use of advanced servo transfer function for the use in laser control. Despite of the added flexibility in implementing servo transfer functions, it is still important to keep in mind at additional servo gain is only beneficial at frequencies where the noise floor of the servo is not a limiting factor. That is to say that noise suppression is still fundamentally limited by the input noise of the servo.

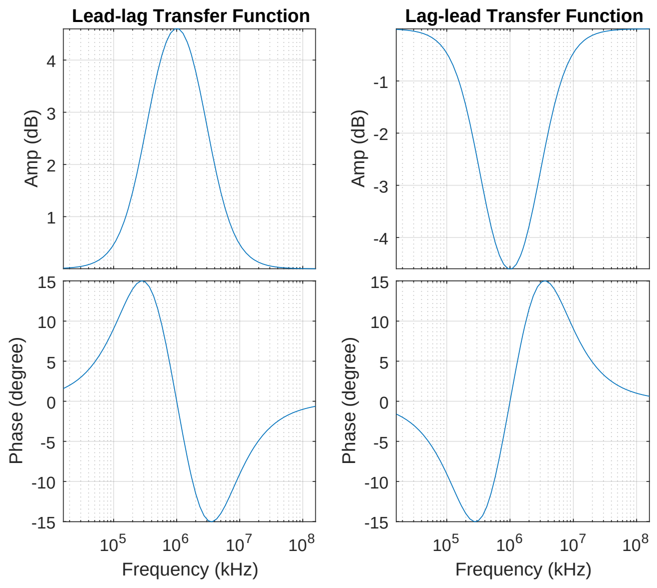


Figure 4.14: The transfer function of the lead-lag and the lag-lead filer with center frequency at 1 MHz and a bandwidth of 1.5 MHz. The lead-lag filter is composed of zeros at 250 kHz and 4 MHz and poles at 500 kHz and 2 MHz. The lag-lead filter is composed of zeros at 500 kHz and 2 MHz and poles are 250 kHz and 4 MHz.

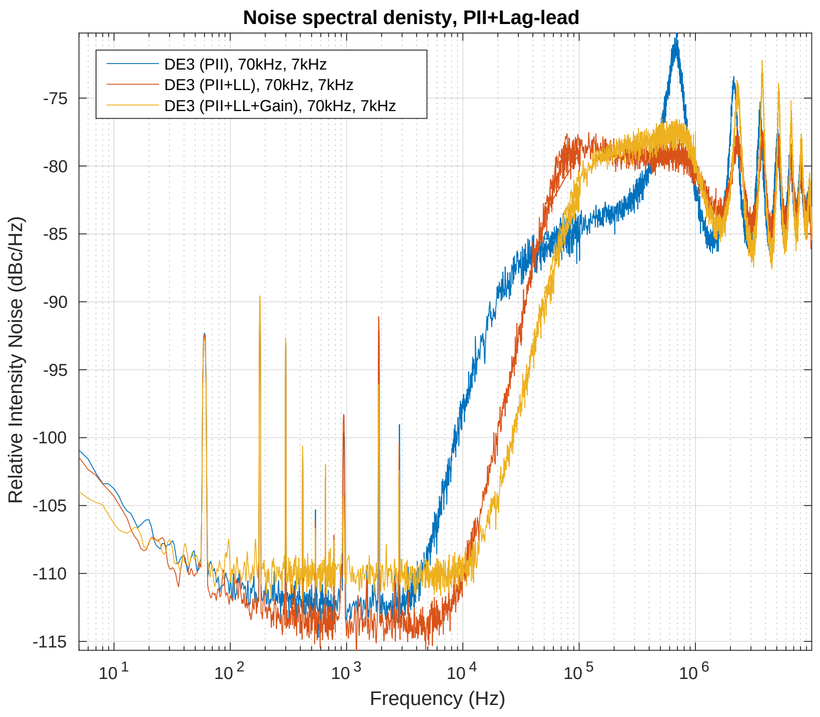


Figure 4.15: Demonstration of the DE3's ability to implement lag-lead filter at center frequency of 700 kHz. The lag-lead filter has two effects 1) suppression of loop resonance of the servo system 2) allowing increase in gain and thus widening of the noise suppression band

4.3 Conclusion

The most intriguing conclusion drawn from the study covered in this chapter is the similar in performance of the analog servo and the FPGA servos when configured in closed-loop (Section 4.2.4). Specifically, when the analog servo and the FPGA servo are configured with similar transfer functions, both as PII filters, the closed-loop performance does not show substantial differences in either noise suppression or the frequencies of the loop resonance. The similarity in loop resonance is a result of the plant being a substantial source of delay in the closed-loop system and the similarity in noise floor is a result of carefully design circuits with detailed consideration in noise reduction already covered in Chapter 2.

The work by this chapter demonstrates the versatility of an FPGA servo and its ability to form advanced transfer functions. The implementation of advanced transfer function an FPGA servo is suggested by a few published work as a potential benefit this new technology [14], but this work is the first to demonstrate the use of advanced transfer functions in the form of PI^3 and PII+lag-lead in an FPGA servo (Section 4.2.5 - 4.2.6). We are able to show the benefit of these advanced transfer functions in widening the noise suppression band beyond what was demonstrated with PII, which is considered a benchmark in this study.

Another contribution covered in this chapter is the summary of the tuning techniques needed to work with an FPGA and how the techniques differ from working with an analog servo (Section 4.1). The study shows that tuning parameters like loop gain and frequencies of the poles and zeros affect the closed-loop transfer function of the FPGA servo similarly to its effect on the anlog servo. In comparison to the analog servo, the FPGA servo has a stronger dependence on gain distribution as the system is limited by the noise floor at the ADC and the output range of the DAC. The work summarize a strategy for the optimal gain distribution when using an FPGA servo (Section 4.1.6), where the user must be careful to avoid the signals from exceeding the range of the ADC or the DAC at any time.

Chapter 5

Case Study: Evaporation in a Dipole Trap

Previous chapters describe the design of a high-performance FPGA servo and the evaluation of its performance in closed-loop. As a natural extension to the work described so far, we look into using the FPGA-based system to improve existing experiments. In this case study, we investigate in the relationship between the active stabilization of the laser intensity and the evaporation efficiency in a dipole trap. This case study demonstrates the use of an FPGA servo in the intensity stabilization of lasers and shows that its use is not limited to frequency or phase stabilization (as demonstrated in various published work [8], [14]). This chapter starts by motivating the importance of laser intensity in a dipole trap and eventually reach the conclusion of whether active intensity stabilization is a suitable solution to improving evaporation efficiency.

5.1 Motivation and Background

Optical-only traps with wavelength far detuned from the cooling transitions of Alkali atoms have long been used to trap and cool neutral atoms to study the dynamics of atomic clouds at temperatures under the micro Kelvin level [51]. At the Quantum Degenerate Gas (QDG) laboratory, an dipole trap is used to perform a second cooling stage to create ultra-cold Li or Li+Rb atomic mixtures, and it is preceded by loading and cooling of neutral Li and/or Rb atoms in a Magneto-Optical Trap (MOT) [52], [53]. The dipole trap is generated by focusing the output of a 1090 nm, 100 W fiber laser onto the atoms, and it is responsible for evaporatively cooling the atom cloud further and confining the cloud for scientific investigation. The final stage of each experiment is the absorption imaging of the atomic cloud with a small amount of on resonance light, which captures the atom number and the physical distribution of the atoms and provides clues as to what happened during the experiment.

The purpose of this investigation of evaporative cooling of Li in a dipole trap is to improve the evaporation efficiency since this will increase the number of atoms available for the experiments. A higher atom population has many benefits. Examples include a higher quality absorption image due to a higher signal to noise and a better evaporation efficiency of an atomic mixture (Li and Rb for example) where Li acts as the sympathetic coolant [54]. Prior to this study, the atoms undergo a transfer from a D_2 MOT to a high power dipole trap formed by a 100 W multimode fiber laser (SPI SP-100C-0013), and then another transfer to a low power dipole trap formed by a 20 W single mode fiber laser (IPG YLR-20-1064-LP-SF). The transfer is necessary because the 100 W SPI laser lacks the intensity stability to cool the Li cloud below temperatures of 1 uK, and the 20 W IPG laser is not powerful enough to allow a direct transfer to it from the D_2 MOT. This study investigates the stability issues in the 100 W SPI laser and aims to minimize the atom losses due to transfer between the two dipole traps.

5.2 Dipole Trap and Trap Frequency

Improving the evaporation efficiency starts with a basic understanding of dipole traps. Although detailed reviews of dipole traps can be found in various published work [55], this thesis provides a condensed overview that covers the basic concepts needed for investigating issues related to evaporation efficiency. One concept is the approximation of the profile of a dipole trap as a harmonic potential, which has constant spacing between adjacent energy levels. This approximation is good for atoms with very low energy as they only explore the bottom of the Gaussian shaped potential which is very nearly harmonic. Here the energy spacing is characterized by the harmonic trap frequency, and the atoms are most sensitive to perturbations at this frequency.

The trapping potential produced by a focused laser beam is proportional to the intensity. The profile of a focused Gaussian beam is illustrated in Figure 5.1 (left) and it has the following expression.

$$I = I_0 e^{-\frac{2x^2}{w(z)^2}} \tag{5.1}$$

Here I_0 is the peak intensity, z is the position on the propagating axis of the laser and x is the position on an axis perpendicular to the propagation axis of the laser. The parameter $w(z)$ is the width of the Gaussian profile as a function of z position with the following expression.

$$w(z) = w_0\sqrt{1+\left(\frac{z}{z_\mathrm{R}}\right)^2} \tag{5.2}$$

The interaction between the light field and neutral atoms depends on the wave length of the light field and the atomic transitions available in the atoms. Fine structures aside, the single valence electron in an ^{6}Li atom can make transition between the ground state $1\mathrm{s}^2 2\mathrm{s}^1$ and the excited state $1\mathrm{s}^2 2\mathrm{p}^1$ with light at 671 nm. Both the SPI laser and IPG laser with respective central wavelengths of 1091 nm and 1065 nm are far-red-detuned from the transition and form attractive potentials for the atoms.

The attractive energy potential has the profile of the following

$$U \propto I \sim U_0 e^{-\frac{2x^2}{w(z)^2}},\ w(z) = w_0\sqrt{1+\left(\frac{z}{z_\mathrm{R}}\right)^2} \tag{5.3}$$

When only considering radial motion at the beam waist, this expression can be converted into the following by applying Taylor expansion.

$$U \sim U_0(1-\frac{2x^2}{w_0^2}+\frac{4x^4}{w_0^4}) \tag{5.4}$$

After dropping the higher order terms in the Taylor expansion, the energy potential can be approximated as a harmonic potential with the following expression,

$$U = U_0\frac{2x^2}{w_0^2} \tag{5.5}$$

which is harmonic and therefore has evenly spaced energy levels. This approximation is illustrated in Figure 5.1 (middle). This means that trapped atoms can be excited by a perturbation (acting on their positional degree of freedom) at this frequency and move up the ladder of energies leading to heating of the ensemble and eventually loss of particles from the trap. The spacing between energy level has the following expression,

$$\Delta U = \hbar\sqrt{\frac{4U_0}{mw_0}} \tag{5.6}$$

which can be characterized by the following

$$\omega_r = \sqrt{\frac{4U_0}{mw_0}} \tag{5.7}$$

referred to as the trap frequency. Here, the subscript "r" refers to the radial axis of the laser.

The same second order approximation can be applied to the propagating axis, z, with the following expression,

$$\omega_a = \sqrt{\frac{4U_0}{mz_R}} \tag{5.8}$$

where the characteristic length scale, rather than the width of the beam waist, is the Rayleigh length. The ensemble temperature is sensitive to perturbations at or near these frequencies. In our experiments, we also use a crossed beam geometry, as illustrated in Figure 5.1 where the trap frequencies are determined by the combined potential of the two beams.

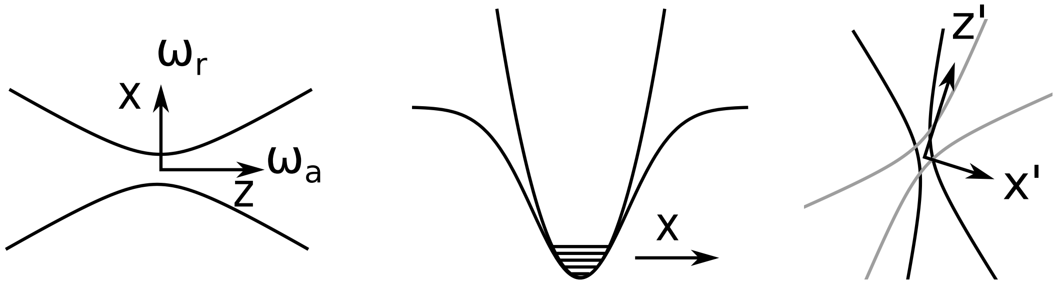


Figure 5.1: Side view of a focused Gaussian beam (left), the Gaussian-shaped potential, in the cross section (middle) often approximated as a harmonic potential in the deep trap limit and the top view of crossed focused beam (right).

5.3 Loss Mechanisms

A commonly used cooling technique in an optical dipole trap is so-called forced evaporative cooling where the potential of the trap is lowered slowly by decreasing the beam intensity forcing the hottest atoms to escape and leaving behind a cooler sample after the ensemble has re-thermalized. In this process, not all atoms that started in the dipole trap remain. As shown by OHara et al [56], [57], when the re-thermalization timescale is very rapid, the evaporation is efficient and the ensemble temperature closely tracks the trap depth. In this case, the ratio of the trap depth to the temperature $U/kT = \eta$ is typically of the order $\eta = 10$ and the number of atoms in the trap follows the following scaling law.

$$\frac{N}{N_i} = (\frac{U}{U_i})^{3/[2(\eta-3)]} = (\frac{U}{U_i})^{0.22} \tag{5.9}$$

Here, U_i and U are the depth of the initial and the final trap potential and the N_i and N are respectively the initial and final population. A detailed discussion of the efficiency of the force evaporative cooling is provided by the work of OHara at el [56] [57].

With the knowledge of the achievable efficiency of a force evaporation routine, it was surprising to see the evaporation efficiency of ^{6}Li in the 100 W SPI laser drastically deviate from the scaling law in trap less than 10 W in power. Some detective work went into debugging this problem. The investigation is based on an understanding of various loss mechanisms that can lead to a loss of atoms in a dipole trap.

5.3.1 Background Scattering

Trap loss can be caused by collisions with residual particles in the vacuum. It is somewhat surprising that collisions with these room temperature particles can sometimes be soft enough to leave the trapped particles in the trap but with a higher kinetic energy. Both loss and heating of the ensemble reduce the efficiency of evaporation. However, in our case, we observed efficient evaporation in the 20 W IPG laser. To completely rule out background collisions as the problem, we measured the loss rates in both dipole traps at different laser powers to confirm that the poor SPI evaporation was not the result of a poor vacuum in our experiment.

5.3.2 Loss Due to Resonant Scattering

With background scattering eliminated as a possible cause, the next possible cause evaluated was the loss of atom due to spectral components close to Li_6 D_2 line in the SPI. Off resonant scattering was ruled out since the evaporation in the IPG laser at similar powers was better and its operation wavelength is actually closer to the D2 line at 671nm. The SPI laser was found to have no significant spectral power near 671 nm on an optical spectrum analyzer Ando Q6315. These observations lower the likelihood of resonant scattering as a possible cause. These observations lower the likelihood of optical pumping as a possible cause.

5.3.3 Parametric Heating

With the first two candidates deemed unlikely causes, the final candidate is the stability of the laser itself. The work by Savard at el categorized stability of a dipole trap into two types, the pointing stability which affects the physical location of the trap and the intensity stability which affects the depth of the trap and the trap frequencies [58]. Both types of instability, when oscillating on similar time scales as the trap frequency, are able to excite trapped atoms to higher energies leading to ensemble heating and, eventually, to trap loss. In our experiment, the pointing stability of the SPI is carefully handled with the use of a CNC-machined fiber-mounts from solid stainless steel, weighted mechanical posts and stable commercial mirror mounts. This leaves the intensity noise of the SPI laser the only remaining and a highly likely cause of the evaporation issue.

5.4 Study

As a starting point for this study, we note that it was shown by Savard et al [58] that the heating rate of a trapped atomic ensemble is related to the intensity noise of the trap laser by the following expression.

$$\Gamma_\epsilon = \pi^2 \upsilon_{trap}^2 S_\epsilon(2\upsilon_{trap}) \tag{5.10}$$

Here Γ_ϵ is the e-folding time of temperature in the trap due to intensity noise in the laser. It is related to the value of the trap frequency and the one-sided power spectrum of the relative intensity noise at twice of the trap frequency. This relationship allows us to make the connection between intensity noise of the laser (in RIN) and heating rate.

5.4.1 Heating Rate of Existing Trap

The heating rate of the trapped ensemble can be characterized by measuring the temperature after various hold times. The ensemble temperature is measured using a Time-of-Flight (TOF) method, where the atom cloud is allowed to expand in free space for some time, and the cloud size after expansion (i.e. the speed of the expansion) provides a measure of the average kinetic energy at the beginning of the expansion and in turn the average temperature [42]. The heating rate measurement is therefore a series of TOF measurements after the atom cloud is held in the dipole trap for different times. The heating rate measurement is therefore more time-consuming than a loss rate measurement, but the heating rate measurement gives more insight into the evolution of the ensemble over time.

As shown in Figure 5.2, the temperature of a ^{6}Li cloud is shown for different hold times in the SPI laser trap at 10 W, corresponding to the power where the evaporation efficiency was observed to be poor. The data is consistent with an exponential increase in the cloud temperature with an e-folding time of 1.9 s. With the knowledge of the e-folding time, and the relationship between the laser intensity noise and heating rate in Equation 5.10, it is possible to infer the level of intensity noise that would be responsible for the heating rate. Figure 5.2 shows a measurement of the intensity noise of the SPI laser at a similar output power, and the noise level is consistent with an e-folding time of 1.9 s. The intensity noise of the laser at similar power can be seen in Figure 5.2, to be in similar level as what would be responsible for a e-folding time of 1.9 s. The trap frequency is estimated to be 5 kHz in the axial direction and 1 kHz in the radial direction.

While this agreement is evidence that the problem is related to the intensity noise, we need to confirm that reducing the intensity noise of the SPI actually leads to a lower heating and loss rates.

5.4.2 The Effect of Intensity Stabilization

In order to check if a reduction in the intensity noise reduces the atom loss during evaporation due to heating, we used the active intensity stabilization system presented in Chapter 4 to reduce the intensity noise. This involved using an AOM together with the FPGA servo. The

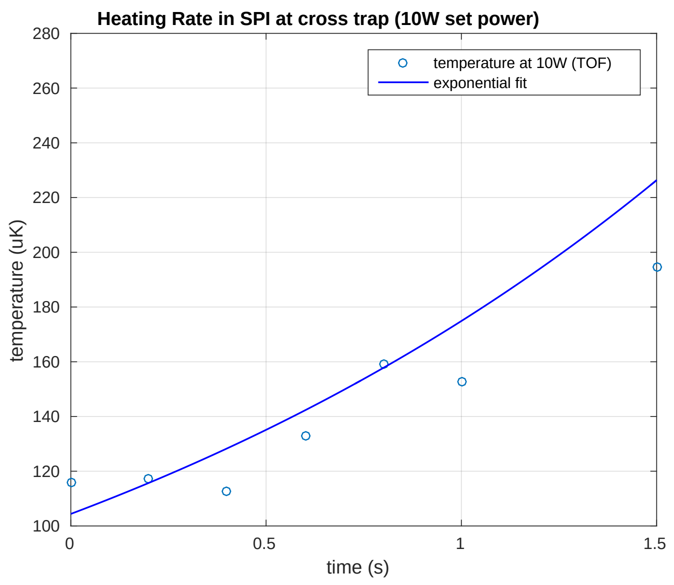


Figure 5.2: heating rate of the atom cloud when SPI held at 10W. The heating rate correspond to a e-folding time of 1.9 s. this is also a screenshot not a proper plot

analog signal previously used to control the SPI power output was used as the setpoint for the FPGA servo.

The effect of the active stabilization on the intensity noise can be seen in Figure 5.3. The active intensity stabilization has a finite bandwidth, but it is able to suppress the intensity noise at the trap frequency to a much lower level. To investigate how the evaporation efficiency changes as a result, a forced evaporation routine was developed to compare the evaporation with and without active stabilization. The evaporation efficiency both with and without active stabilization was independently optimized by choosing the durations of a set of piecewise linear ramps that optimized each linear stage in which the laser power dropped by a factor of 2. This crude optimization method was employed to perform a quick check to determine if active stabilization had any effect on the evaporation efficiency.

The evaporation efficiency of the free-running SPI laser and the SPI laser with noise suppression is shown in Figure 5.4. The predicted number versus final power (given in Eqn. 5.9) for an efficient evaporation ($\eta = 10$) is included in this plot for reference. The active stabilization is observed to provide a slight improvement in atom numbers down to 12 W, but the efficiency drops sharply compared to the unstabilized case for powers below 10 W. This drop was not further investigated but may have been due to an instability in the servo at lower setpoints or

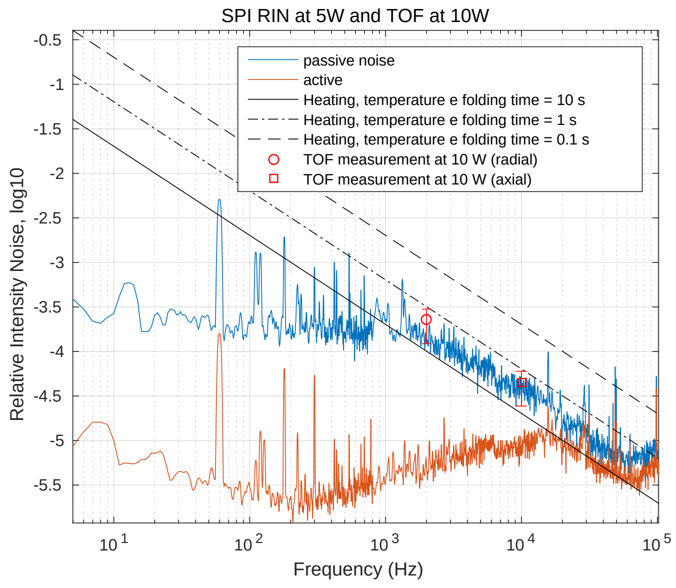


Figure 5.3: Side by side comparison of RIN level responsible for measured heating rate and RIN of laser. The constant heating rate line show 20 dB/decade slope. It appears that active intensity stabilization should reduce heating rate.

due to noise features near to the trap frequency below 10 W not being evident in Figure 5.3 since this was an in-loop rather than an out-of-loop measurement

We are not able to show whether active intensity stabilization with AOM is is sufficient at improvement evaporation efficiency. because the study opt for a simpler method to reduce intensity noise once the method is revealed. We speculate that the effect of active intensity stabilization is not necessarily thoroughly monitored during the experiment as the noise spectrum of the laser is taken at a single power with an in-loop measurement rather than an out-of-loop measurement at multipler power setting.

5.4.3 Passive Stability

In investigating the intensity stability of the 100 W SPI laser, we found that the passive stability of the SPI laser is better at high power setpoints. With the use of the SR780 VNA and the DET10A photodetector, the intensity noise measurement of the SPI laser is taken at various power settings between 1 W and 100 W. The results are shown in Figure 5.5 and reveal that this laser exhibits a relatively low RIN at high power setpoints and a much higher RIN at low powers. The data revealed a much more simple remedy to fix the evaporation efficiency. The

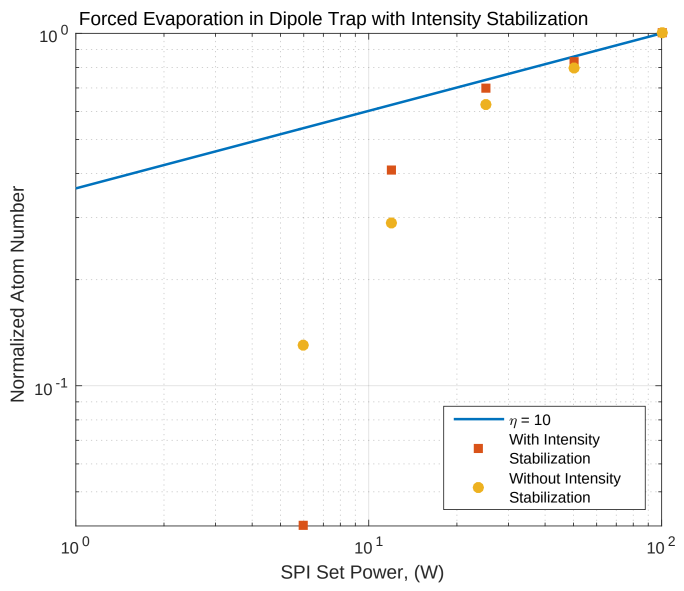


Figure 5.4: Evaporation efficiency of the sample without (yellow) and with (red) intensity stabilization. It is possible to see a small improvement in efficiency for end trap power of 50, 25 and 12 W, but the evaporation still cannot reach the desired trap depth.

solution was to operate the laser at a high intensity setpoint and to turn the power down using an external AOM. This behavior also provides a rather clear explanation as to the drastic performance difference between our two dipole trap lasers. By driving forced evaporation through lowering the power setpoint of the SPI laser, the RIN and ensemble heating would increase counteracting the cooling of the sample and leading to a rather inefficient evaporation.

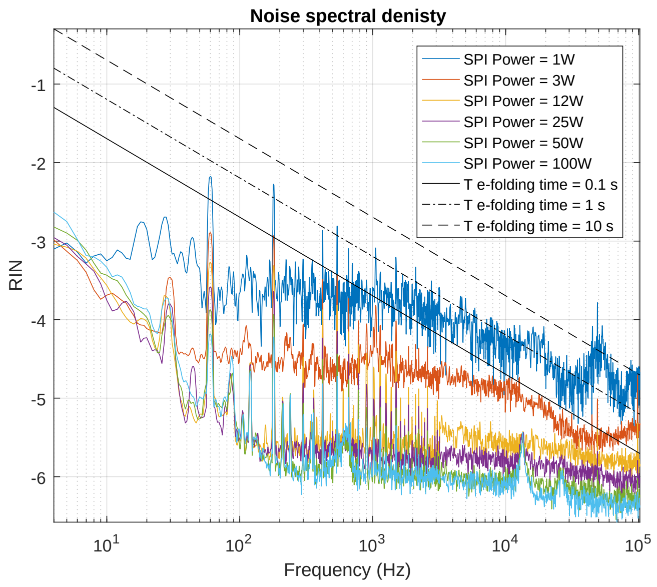


Figure 5.5: passive stability of fiber laser at various power settings in RIN.

5.4.4 Improved Evaporation

With a better knowledge of the RIN of the SPI laser, we were able to depreciate the use of the power setting on the SPI and instead use an AOM and a Direct Digital Synthesizer (DDS) with variable RF output to modify the depth of the trap. Indeed, the equipment was already in place due to the previous work on active intensity stabilization. As shown in Figure 5.6, this method produced a highly efficient evaporation in the SPI down to very low trap depths. This one change yielded a factor of 4 improvement in atom number and eliminated the need to transfer the atoms to the IPG for the final evaporation stages.

5.5 Future Work

Since the primary objective of this particular study was to improve the total atom number at the lowest temperatures by improving the evaporation in the SPI and avoiding the need to transfer between the two dipole traps, the study ended once this objective was fulfilled. While some evidence was obtained that active intensity stabilization improved the evaporation, further investigation was unnecessary.

Although the observed improvement in evaporation was only marginal, this study does not

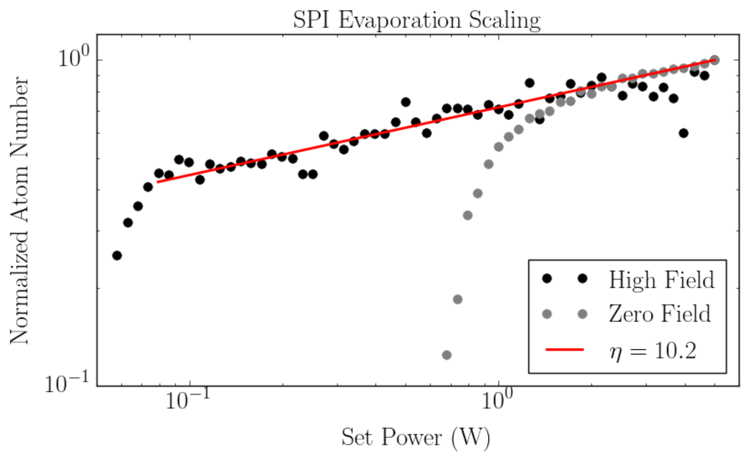


Figure 5.6: The atom number versus final power in the evaporation ramp using the SPI at near its maximum power setup (where the passive stability of the laser is relatively good) and an AOM to control the power delivered to the atoms showed a highly efficient evaporation ($\eta = 10$). The atom number shows an apparent drop when the ensemble is imaged at zero magnetic field since Feshbach molecules are forming at powers below 1 W and are not imaged at low magnetic fields. The images taken at high magnetic field show that the ensemble is still present and still evaporating very efficiently down to powers below 100 mW.

undermine the usefulness of active stabilization. One likely use of the IPG laser that was freed up from the new evaporation routine is in lattice-forming. As it was shown in the work by Blatt et al, active intensity stabilization can improve the lifetime in a lattice trap where the passive stability of the laser, at 106 level in RIN, was improved to the 107 level with the use of an AOM-based noise-eater [59]. It should be noted that the active intensity stabilization discussed in the work have to undergo a redesign, particularly in the detection and front end section, to achieve a similar noise floor as the one presented in the work by Blatt el at.

Chapter 6

Conclusion

In the quest to understand the theoretical and practical limitations of digital servos realized by FPGAs, this work involved the design and construction of FPGA servos based on two different FPGA development platforms (Chapter 2): the DE2 and the DE3 by Altera. Both servos are shown to be capable of reaching loop delay of less than 200 ns (including conversion and computation delay) and are competitive against the published open-source design by NIST with a total delay of 320 ns [14] and the Digilock product by Toptica with a total delay of 200 ns [17]. The design based on the DE2 platform is consolidated onto a single daughtercard that is compatible with most of the development platforms provided by Altera, such as the DE3 through an adapter and the new FPGA platforms that have built-in HPS processors like the DE1-SOC.

On the firmware side, this work included servo logic designs in the form of the IIR filter and other useful instrumentation tools like the AWG (Chapter 3). This work also explored the practical aspects of developing high speed signal processing modules in an FPGA with an emphasis on the verification of the each part of the implementation. Contributions are made to the development of digital servos by optimizing each IIR filter to a single clock delay with a master clock speed of 50-75 MHz. This feature reduces the computation delay through the FPGA servo and reduces the overall latency through the FPGA servo. In addition, this work explores an interesting and important limitation of discrete-time filters that is highly relevant to the present discussion. We explore the discrete frequency resolution of poles and zeros implemented by an IIR filter, and an expression is derived for the frequency resolution in a first order IIR filter.

This work also reports on a side-by-side comparison between an FPGA servo and a high-speed analog servo in closed-loop in both a self-lock configuration and an intensity stabilization configuration (Chapter 4). In the comparisons, it is clear that the noise floor and the bandwidth of the servos play an important role in the closed-loop performance of the system. In our design of the FPGA servos, the overall noise floor of the servo has a best case scenario of 10 nV/$\sqrt{\text{Hz}}$ at 1 kHz, when the input gain (gain between the servo input and ADC) is set to the maximum. This performance is similar to that of the analog servo. However, it is important to note that the noise floor of a FPGA servo is limited by the ADC at 177 nV/$\sqrt{\text{Hz}}$ which means that a large gain is needed between the servo input and the ADC. Also, the distribution of gain in the FPGA servo loop has to be carefully tuned to produce the 10 nV/$\sqrt{\text{Hz}}$ noise performance.

The analog servo and the FPGA servo are also different in signal latency. Broadly speaking, a high-speed analog servo is faster than the FPGA servo designed in this work by >4 times. In this work, we benchmarked a fast commercial analog servo with closed-loop bandwidth of of 10 MHz. In comparison, the FPGA servos produced by this thesis operate with closed-loop bandwidth of 2-2.5 MHz. In the present design, the signal conversion from the analog to the digital domain and back again to the analog domain from the digital domain is substantial. The conversion delay is 80-120 ns and accounts for 40-60% of the total delay in the FPGA servo. The computation delay takes up a smaller fraction as a result of the speed optimization in designing the IIR logic, with the cascade of two third order IIR filters costing between 23 ns (DE2 servo) and 40 ns (DE3 servo). This account for 13 - 20% of the total delay. The delay in the analog signal conditioning stage in this design is only 30 ns (15% of the total) and other propagation delay, such as the time it takes for the digital signals to travel from the ADC to the FPGA and then back to the DAC), makes up for the rest. We note that the total delay through an analog servo should be similar in magnitude to the delay through the analog front-end of this FPGA servo with the use of similar op-amp technology. The delay in signal conversion and, to a lesser extent, the computational delay are the culprits in the additional delay in an FPGA servo. The longer total delay of the FPGA servo limits its use to closed-loop system with a target closed loop bandwidth of <1 MHz. However, even with this constraint, the FPGA servo is still able to satisfy a large number of closed-loop applications.

This work also explores the use of a digital servo design and the quantization error that results. Some published works has already argued for and demonstrated that the quantization error of the ADC and the DAC does not degrade the performance of an FPGA servo [14],

[18]. The work done in thesis agree with these findings and found the noise level of the ADC in $\mathrm{nV}/\sqrt{\mathrm{Hz}}$ almost more important. In this work, we observed two other quantization effects. The first is the quantization of the z-domain due the finite computation precision that the IIR can be implemented in (Section 3.2.5). The second is the effect of computation precision on the noise floor of the overall servo system. To the best of our knowledge, these effects have not been explicitly investigated previously.

One application of an FPGA servo explored in this work is the intensity stabilization of a laser for atom trapping. In particular, we examined the evaporation efficiency of atoms from an optical dipole trap with and without intensity stabilization (Chapter 5). The work found preliminary evidence suggesting that intensity stabilization improved the evaporation efficiency; however, because a different solution was found further work to quantify this effect was not performed. In particular, this study found that the passive intensity stability of the laser is sufficient when the laser operates at a high power setting and thus forced evaporation was performed using an external AOM to modify the beam intensity. The study does not undermine the usefulness of active intensity stabilization in scenaios where the passive stability of the laser is not sufficient [60]; nor does it undermine the usefulness of FPGA servos in AMO experiments as demonstrated in many published works the use in active stabilization of laser frequency and phase [5], [8], [14].

The development of fast FPGA servo hardware and logic are only the the first steps towards wide spread use of digital servos in closed-loop laser control applications. This work argues that a well-designed FPGA servo is comparable with a high-performance analog servo as long as the target closed-loop bandwidth of the laser system is less than 1 MHz. Although the performance of the FPGA servo is convincing, it still takes effort to optimize the servo in different laser systems that may require more tuning techniques and even different servo transfer functions. So far, the closed-loop control application that have been demonstrated to work with FPGA servo are intensity stabilization (this work), frequency stabilization (NIST) [14]. Existing open-source FPGA servo designs, like in the one published by NIST can be very helpful in accelerating the wide-spread use of FPGA servos.

Bibliography

[1] Russell Tessier and Wayne Burleson. Reconfigurable computing for digital signal processing: A survey. *The Journal of VLSI Signal Processing*, 28(1):7–27, 2001.

[2] Ian Kuon, Russell Tessier, and Jonathan Rose. FPGA architecture: Survey and challenges. *Foundations and Trends in Electronic Design Automation*, 2(2):135–253, 2008.

[3] RWP Drever, John L Hall, FV Kowalski, J_ Hough, GM Ford, AJ Munley, and H Ward. Laser phase and frequency stabilization using an optical resonator. *Applied Physics B*, 31 (2):97–105, 1983.

[4] David J Jones, Eric O Potma, Ji-xin Cheng, Berndt Burfeindt, Yang Pang, Jun Ye, and X Sunney Xie. Synchronization of two passively mode-locked, picosecond lasers within 20 fs for coherent anti-Stokes Raman scattering microscopy. *Review of Scientific Instruments*, 73(8):2843–2848, 2002.

[5] John Stockton, Michael Armen, and Hideo Mabuchi. Programmable logic devices in experimental quantum optics. *JOSA B*, 19(12):3019–3027, 2002.

[6] Arne Schwettmann, Jonathon Sedlacek, and James P Shaffer. Field-programmable gate array based locking circuit for external cavity diode laser frequency stabilization. *Review of Scientific Instruments*, 82(10):103103, 2011.

[7] A. Walther, F. Ziesel, T. Ruster, S. T. Dawkins, K. Ott, M. Hettrich, K. Singer, F. Schmidt-Kaler, and U. Poschinger. Controlling fast transport of cold trapped ions. *Physical Review Letters*, 109:080501, August 2012.

[8] Jaroslaw Labaziewicz, Philip Richerme, Kenneth R Brown, Isaac L Chuang, and Kazuhiro

Hayasaka. Compact, filtered diode laser system for precision spectroscopy. *Optics Letters*, 32(5):572–574, 2007.

[9] Nils B Jørgensen, Danny Birkmose, Kristian Trelborg, Lars Wacker, Nils Winter, Andrew J Hilliard, Mark G Bason, and Jan J Arlt. A simple laser locking system based on a field-programmable gate array. *Review of Scientific Instruments*, 87(7):073106, 2016.

[10] Zhouxiang Xu, Xian Zhang, Kaikai Huang, and Xuanhui Lu. A digital optical phase-locked loop for diode lasers based on field programmable gate array. *Review of Scientific Instruments*, 83(9):093104, 2012.

[11] Wang Xiao-Long, Tao Tian-Jiong, Cheng Bing, Wu Bin, Xu Yun-Fei, Wang Zhao-Ying, and Lin Qiang. A digital phase lock loop for an external cavity diode laser. *Chinese Physics Letters*, 28(8):084214, 2011.

[12] Sébastien Merlet, Laurent Volodimer, Michel Lours, and F Pereira Dos Santos. A simple laser system for atom interferometry. *Applied Physics B*, 117(2):749–754, 2014.

[13] RWM Smith, IL Freeston, BH Brown, and AM Sinton. Design of a phase-sensitive detector to maximize signal-to-noise ratio in the presence of Gaussian wideband noise. *Measurement Science and Technology*, 3(11):1054, 1992.

[14] D. R. Leibrandt and J. Heidecker. An open source digital servo for atomic, molecular, and optical physics experiments. *Review of Scientific Instruments*, 86(12):123115, 2015.

[15] BM Sparkes, HM Chrzanowski, DP Parrain, BC Buchler, PK Lam, and T Symul. A scalable, self-analyzing digital locking system for use on quantum optics experiments. *Review of Scientific Instruments*, 82(7):075113, 2011.

[16] G Yang, JF Barry, ES Shuman, MH Steinecker, and D DeMille. A low-cost, FPGA-based servo controller with lock-in amplifier. *Journal of Instrumentation*, 7(10):P10026, 2012.

[17] Laser locking & laser driving - electronic control for passive stability and active locking. Technical report, Toptica Photonics. [Online; accessed 9-April-2017].

[18] RW Stewart and E Pfann. Oversampling and sigma-delta strategies for data conversion. *Electronics & Communication Engineering Journal*, 10(1):37–47, 1998.

[19] *D2 Series Laser Servo.* Vescent Photonics, November 2012.

[20] *DE3 Development and Education Board User Manual.* Altera Corporation, November 2009.

[21] *DE2 Development and Education Board User Manual.* Altera Corporation, 2006.

[22] Todd P. Meyrath and Florian Schreck. Octal 16-bit DAC. Technical report, University of Texas at Austin, 2004.

[23] *Data Conversion HSMC Reference Manual.* Altera Corporation, March 2008.

[24] *THDB-ADA User Manual: High-Speed A/D and D/A Development Kit.* Terasic Technolgies Inc, September 2014.

[25] How do I program an EPCS device with a SOF file and Nios II ELF file using the Quartus II Programmer, 2012. URL `https://www.altera.com/support/support-resources/knowledge-base/solutions/rd04112006_450.html`. [Online; accessed 9-April-2017].

[26] Variable gain amplifier selection table. October 2006. URL `http://www.analog.com/static/imported-files/product_selection_guide/VGA_1_pg.pdf`. [Online; accessed 9-April-2017].

[27] *Dual, 14-BIT 275 MSPS Digital-to-analog Converter.* Texas Instruments, November 2004.

[28] Graham C. Goodwin, Stefan F. Graebe, and Mario E. Salgado. *Control System Design.* Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition, 2000. ISBN 0139586539.

[29] Karl Johan Aström and Richard M Murray. *Feedback Systems: an Introduction for Scientists and Engineers.* Princeton University Press, 2010.

[30] Paul Katz. *Digital Control Using Microprocessors.* Prentice Hall, 1981.

[31] Robert Eugene Bogner and Anthony George Constantinides. *Introduction to Digital Filtering.* John Wiley & Sons, 1975.

[32] Belle A Shenoi. *Introduction to Digital Signal Processing and Filter Design.* John Wiley & Sons, 2005.

[33] KB MacAdam, A Steinbach, and Carl Wieman. A narrow-band tunable diode laser system with grating feedback, and a saturated absorption spectrometer for Cs and Rb. *American Journal of Physics*, 60:1098–1098, 1992.

[34] Alan V. Oppenheim and R.W. Schafer. *Digital Signal Processing.* MIT video course. Prentice-Hall, 1975. ISBN 9780132146357.

[35] *Cyclone II Device Handbook, Volume 1.* Altera Corporation, February 2008.

[36] Steve Kilts. *Advanced FPGA Design: Architecture, Implementation, and Optimization.* John Wiley & Sons, 2007.

[37] Willis J Tompkins. Biomedical digital signal processing. *Editorial Prentice Hall*, 1993.

[38] *IEEE Standard Verilog Hardware Description Language IEEE Std 1364-2001.* IEEE, September 2001.

[39] Eric D Black. An introduction to Pound–Drever–Hall laser frequency stabilization. *American Journal of Physics*, 69(1):79–87, 2001.

[40] Maximiliano Osvaldo Sonnaillon and Fabian Jose Bonetto. A low-cost, high-performance, digital signal processor-based lock-in amplifier capable of measuring multiple frequency sweeps simultaneously. *Review of Scientific Instruments*, 76(2):024703, 2005.

[41] Principles of lock-in detection and the state of the art. Technical report, Zurich Intruments, September 2016.

[42] Gregory-Orfeus Konstantinidis. *Diffractive dark-ground imaging of ultra-low atom-numbers in a MOT.* PhD thesis, *Πανεπιστήμιο Κρήτης. Σχολή Θετικών Επιστημών. Τμήμα Φυσικής*, 2012.

[43] Kevin Henderson, Changhyun Ryu, Calum MacCormick, and MG Boshier. Experimental demonstration of painting arbitrary and dynamic potentials for Bose–Einstein condensates. *New Journal of Physics*, 11(4):043030, 2009.

[44] Mariusz Semczuk, Xuan Li, Will Gunton, Magnus Haw, Nikesh S. Dattani, Julien Witz, Arthur K. Mills, David J. Jones, and Kirk W. Madison. High-resolution photoassociation spectroscopy of the $^6\mathrm{Li}_2$ $1^3\Sigma_g^+$ state. *Physical Review A*, 87:052505, May 2013.

[45] Relative intensity noise of distributed feedback laser. Technical report, Eagleyard, 2004.

[46] *3 Volt Voltage Variable Attenuator*. M/A-COM Technology Solutions. Rev. 1. [Online; accessed 9-April-2017].

[47] *Model ATM Series Acoustic-optical modulator*. IntraAction Corp. [Online; accessed 9-April-2017].

[48] Ryan P Scott, Carsten Langrock, and Brian H Kolner. High-dynamic-range laser amplitude and phase noise measurement techniques. *IEEE Journal of Selected Topics in Quantum Electronics*, 7(4):641–655, 2001.

[49] Thermal noise. `http://isites.harvard.edu/fs/docs/icb.topic86897.files/October_13_Temperature/ThermalNoise.pdf`. [Online; accessed 9-April-2017].

[50] John WM Rogers and Calvin Plett. *Radio Frequency Integrated Circuit Design*. Artech House, 2010.

[51] Charles S Adams, Heun Jin Lee, Nir Davidson, Mark Kasevich, and Steven Chu. Evaporative cooling in a crossed dipole trap. *Physical Review Letters*, 74(18):3577, 1995.

[52] William Bowden, Will Gunton, Mariusz Semczuk, Kahan Dare, and Kirk W Madison. An adaptable dual species effusive source and Zeeman slower design demonstrated with Rb and Li. *Review of Scientific Instruments*, 87(4):043111, 2016.

[53] Will Gunton. *Photoassociation and Feshbach resonance studies in ultra-cold gases of ^{6}Li and Rb atoms*. PhD thesis, University of British Columbia, 2016.

[54] M Mudrich, S Kraft, K Singer, R Grimm, A Mosk, and M Weidemüller. Sympathetic cooling with two atomic species in an optical trap. *Physical Review Letters*, 88(25):253001, 2002.

[55] Rudolf Grimm, Matthias Weidemüller, and Yurii B Ovchinnikov. Optical dipole traps for neutral atoms. *Advances in Atomic, Molecular, and Optical Physics*, 42:95–170, 2000.

[56] KM Ohara, ME Gehm, SR Granade, and JE Thomas. Scaling laws for evaporative cooling in time-dependent optical traps. *Physical Review A*, 64(5):051403, 2001.

[57] Kenneth Martin O'Hara. *Optical trapping and evaporative cooling of fermionic atoms*. PhD thesis, Duke University, 2000.

[58] TA Savard, KM Ohara, and JE Thomas. Laser-noise-induced heating in far-off resonance optical traps. *Physical Review A*, 56(2):R1095, 1997.

[59] S. Blatt, A. Mazurenko, M. F. Parsons, C. S. Chiu, F. Huber, and M. Greiner. Low-noise optical lattices for ultracold ^{6}Li. *Physical Review A*, 92:021402, August 2015.

[60] Jameson Rollins, David Ottaway, Michael Zucker, Rainer Weiss, and Richard Abbott. Solid-state laser intensity stabilization at the 10^{-8} level. *Optics Letters*, 29(16):1876–1878, 2004.

Appendix A

The Servo Daughter Card

The servo daughter card is designed to be compatible with the GPIO header used in many Altera development boards, like the DE2 etc. When attached to a compatible FPGA development board, the servo daughter card produces a dual channel FPGA servo, with built-in analog frontend. The design rationale and the performance of the FPGA servo is described in detail in the hardware chapter in Chapter 2. This section covers the rest of technical details including schematics, circuit layout, bill of material and a few mistakes on the board that needs to be worked around and be eliminated in future revisions.

A.1 Power Supply

Operation of the servo daughter requires additional power supply besides the power supply on the FPGA development board. A quiet $\pm 13 \sim 15V$ voltage is needed to power the analog frontend and the current consumption depends on whether the signal conversion section, the ADC and the DAC, is powered through the GPIO header on the FPGA development board or through the power supply input on the servo daughter card. The ADC and DAC on the servo daughter card consumes up to 600 mA of current, whereas the rest of the daughter card circuit consumed up to 200 mA. When the ADC and the DAC is powered through the power supply input on the daughter card via a chain of linear regulators, the high current consumption of the ADC and the DAC cause the linear regulators to operate too close to their thermal limit so this mode of operation is not recommended without additional thermal relief around the linear regulators. Powering the ADC and DAC via the 3.3 V supply on the GPIO header is more convenient, but can not guarantee noise performance.

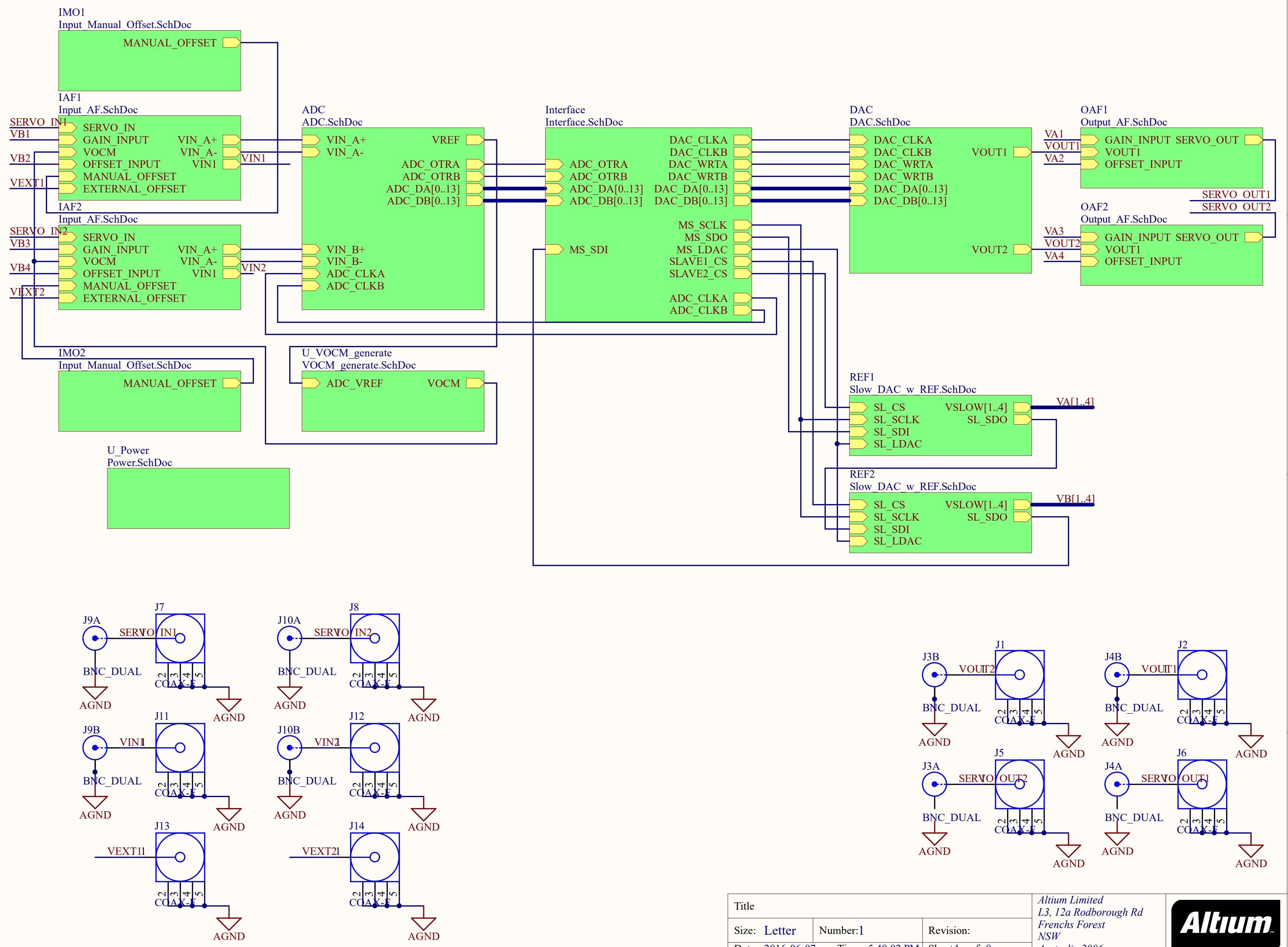
A.2 Pending Design Changes

The next few pages provides the schematics, the layout and the bill of material for the servo daughter card and what is shown here is the very first iteration of the PCB design. All major parts of the design works as intended, but a few quirks exist at the schematics level and needs to be worked around in the assembly process. The first is the polarity of the LDAC signal on the slow DAC IC, DAC8734. The pull up resistor R504(A/B) on the LDAC signal should be modified into a pull down with a near by ground pad on the PCB to enable latching the DAC output by default. The second is a missing input signal to the DACMODE pin on the fast DAC IC, AD8967. This can be fixed with a solder bridge between the DACMODE pin (48) to the adjacent A3V3 pin (47) which selects the independent access mode of the two DAC channel within the IC. The third mistake is the missing resistive divider between the slow DAC output and some of the offset input of the analog frontend. The missing divider need to be inserted in place of the resistor R309(A/B), likely in the form of a ”dead-bug”

The above problems, including the thermal limitation around the linear regulators are to be fixed in future revisions, but there is one other quirk of this design that is to stay. A careful comparison between the ADA card provided by Altera and the daughter servo card designed in this work should reveal that the pin mapping on the GPIO header of the two daughter card are not the same. Because the pin mapping of the fast DAC is different between the two board, change of pin mapping is needed to port FPGA design for one daughter card to the other. To match the servo daughter card to the ADA card means significant redesign of the PCB layout and offer little benefit, this quirk of the design is to stay.

A.3 Design Files

The following page shows the schematics, layout and bill of material of the servo daughter card. The design is meant to be self-documenting, with a hierarchical schematics structure and device annotation that are meant to inform the schematics sub-page that the device is from.



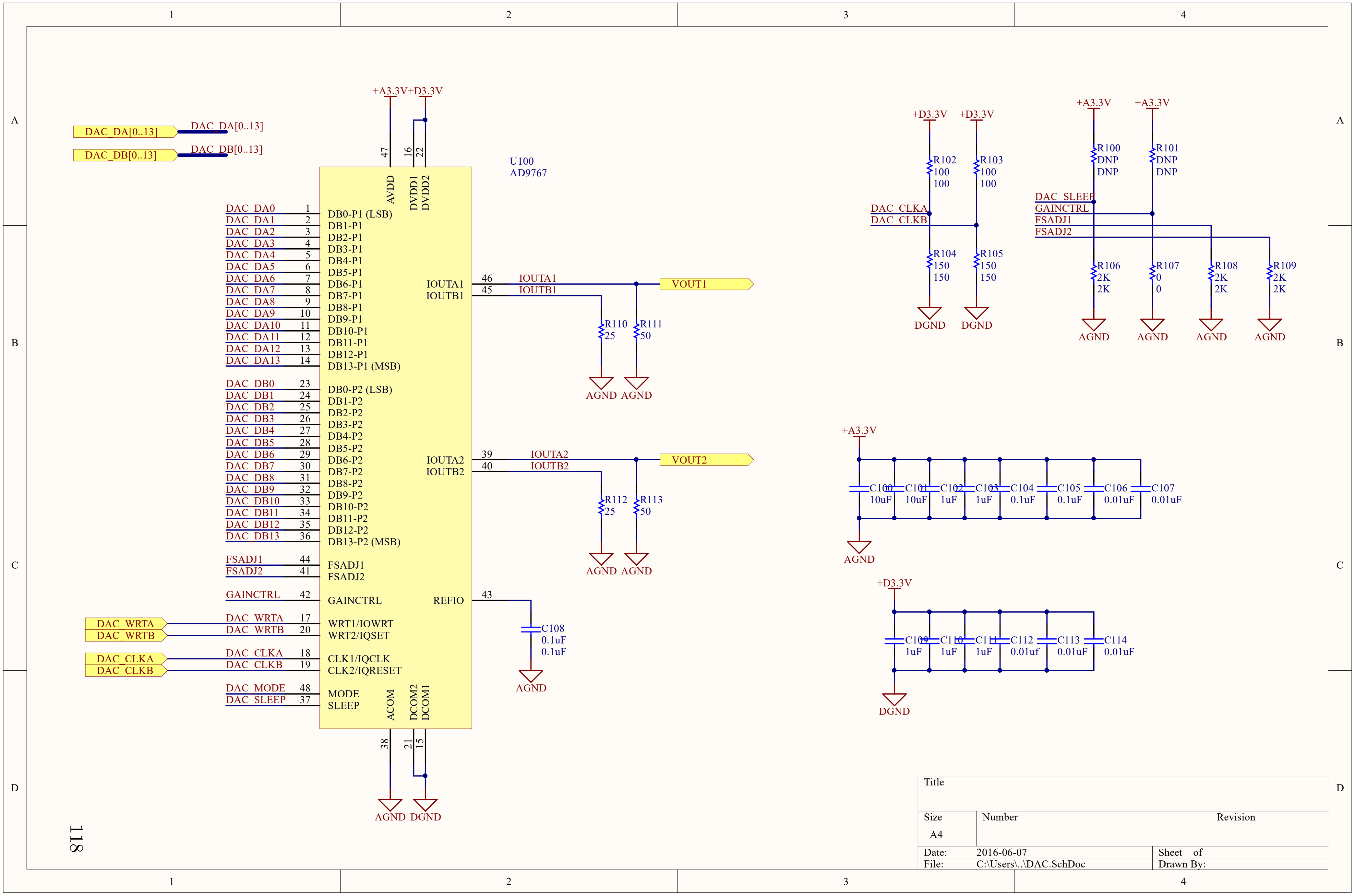
Title

Size: Letter Number: 1 Revision:

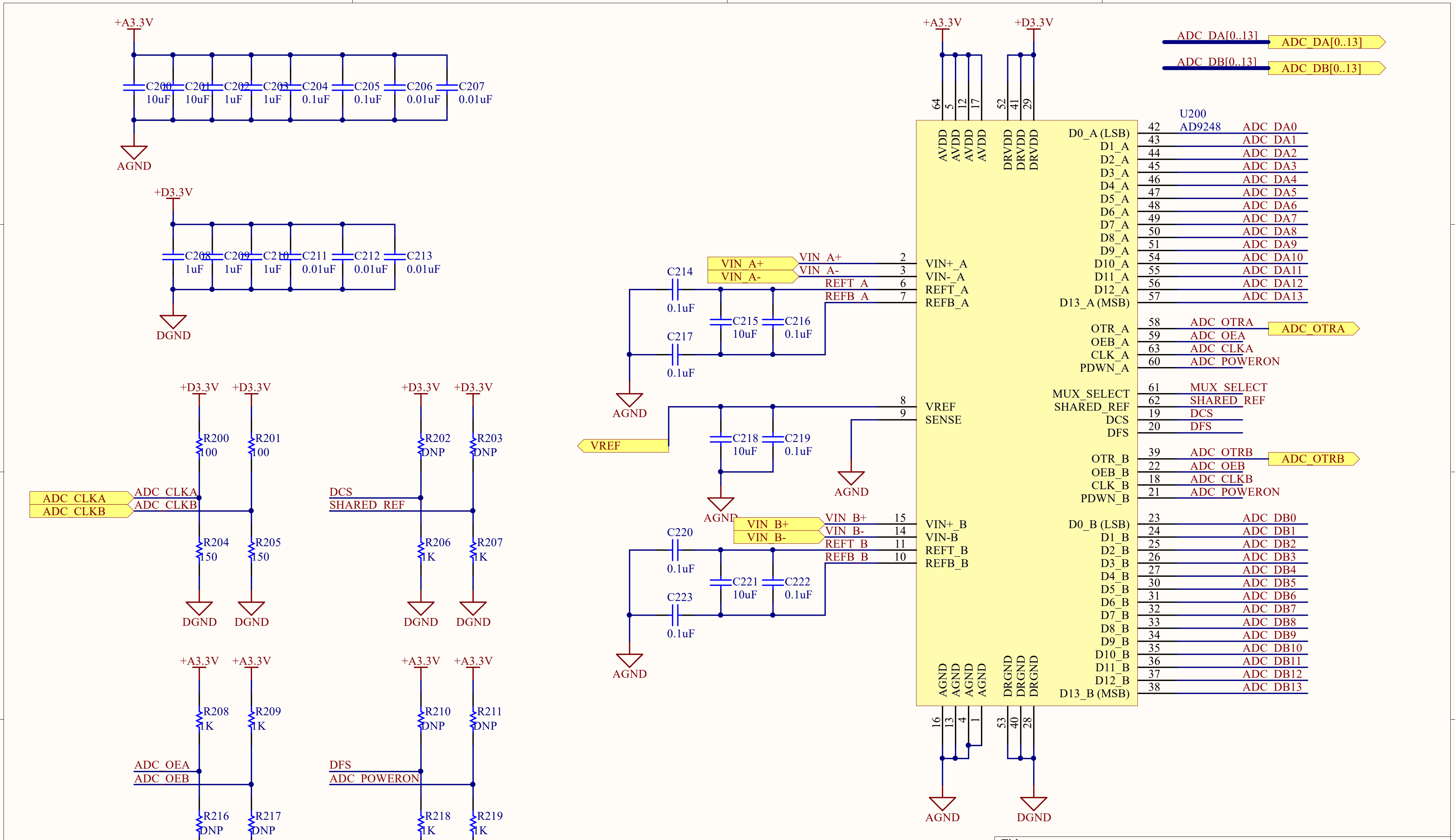
Date: 2016-06-07 Time: 5:40:02 PM Sheet 1 of 8

File: C:\Users\Public\Documents\qdg_projects_repo\trunk\PRJ_servo_daughter_card\top.SchDoc

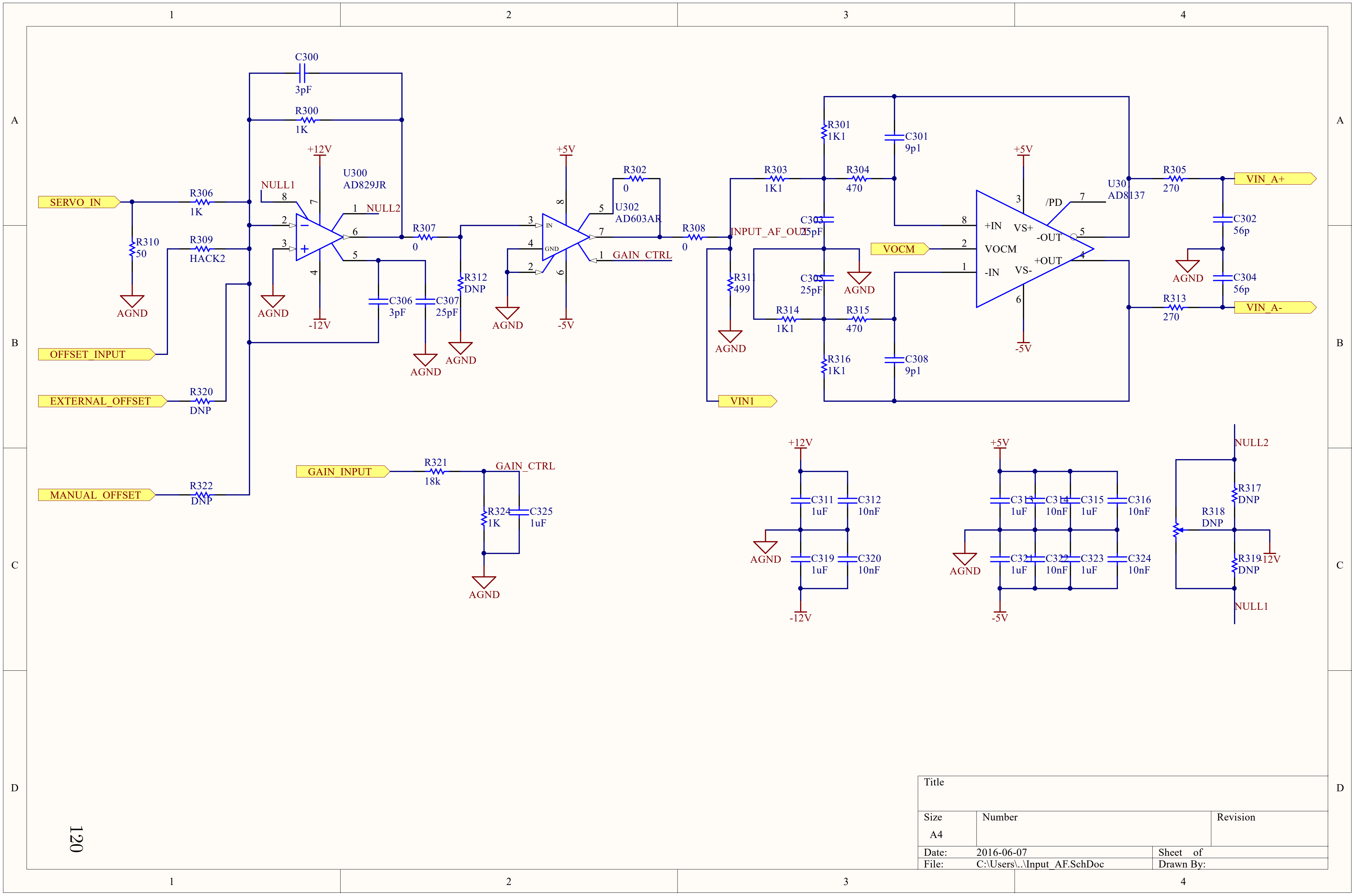
Altium Limited
L3, 12a Rodborough Rd
Frenchs Forest
NSW
Australia 2086

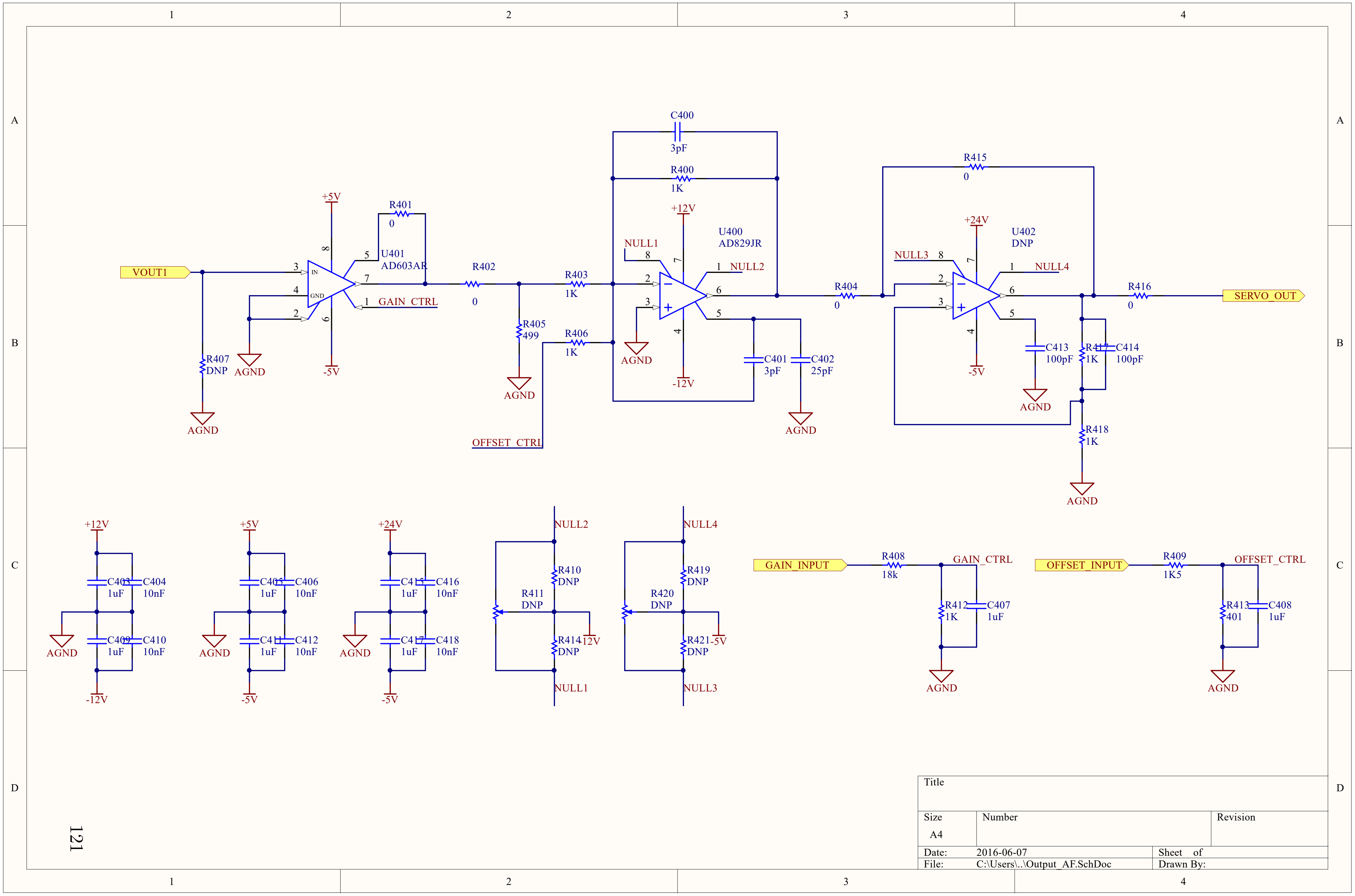


Title		
Size A4	Number	Revision
Date: 2016-06-07		Sheet of
File: C:\Users\..\DAC.SchDoc		Drawn By:

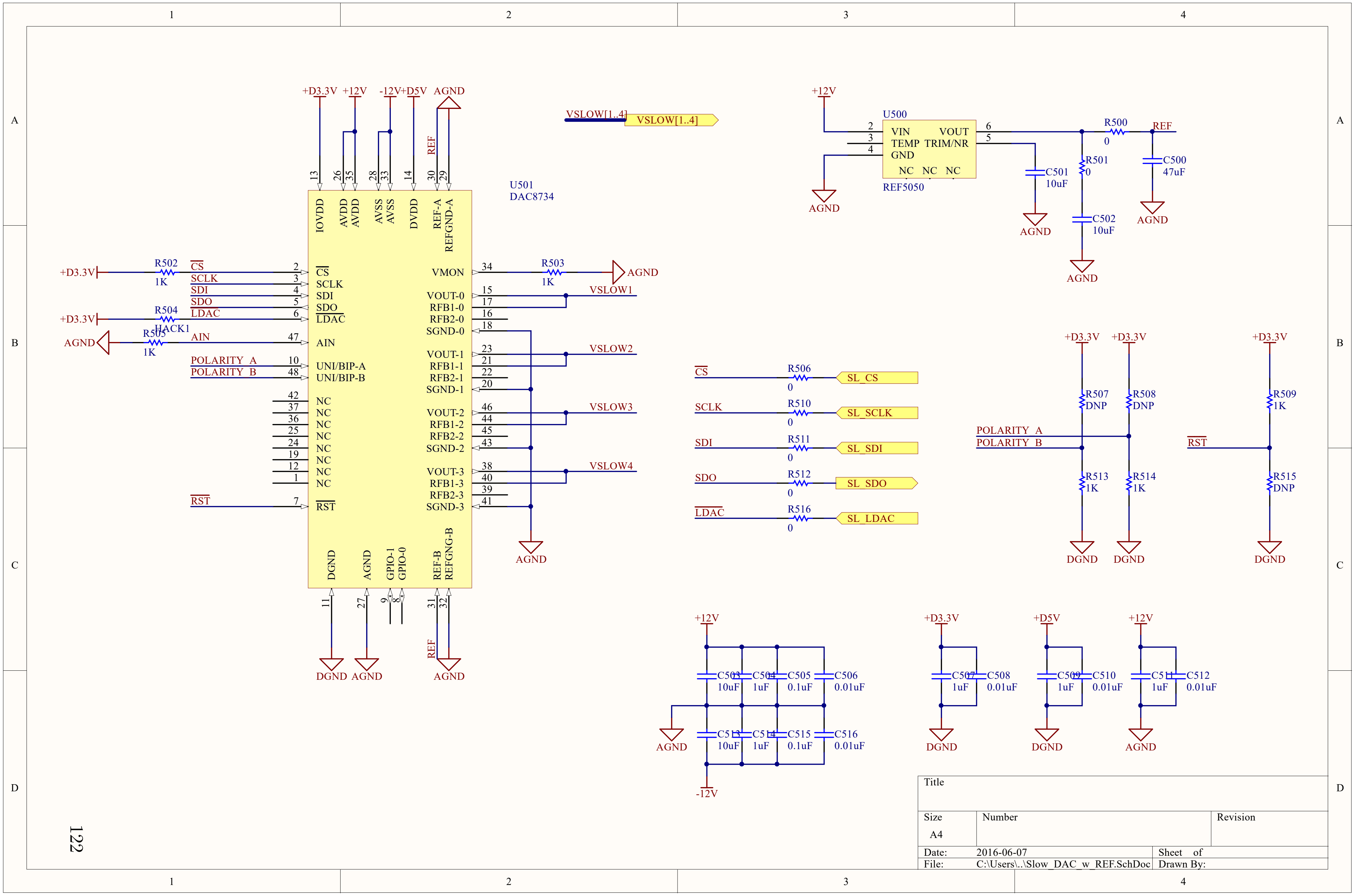


Title		
Size A4	Number	Revision
Date: 2016-06-07		Sheet of
File: C:\Users\..\ADC.SchDoc		Drawn By:

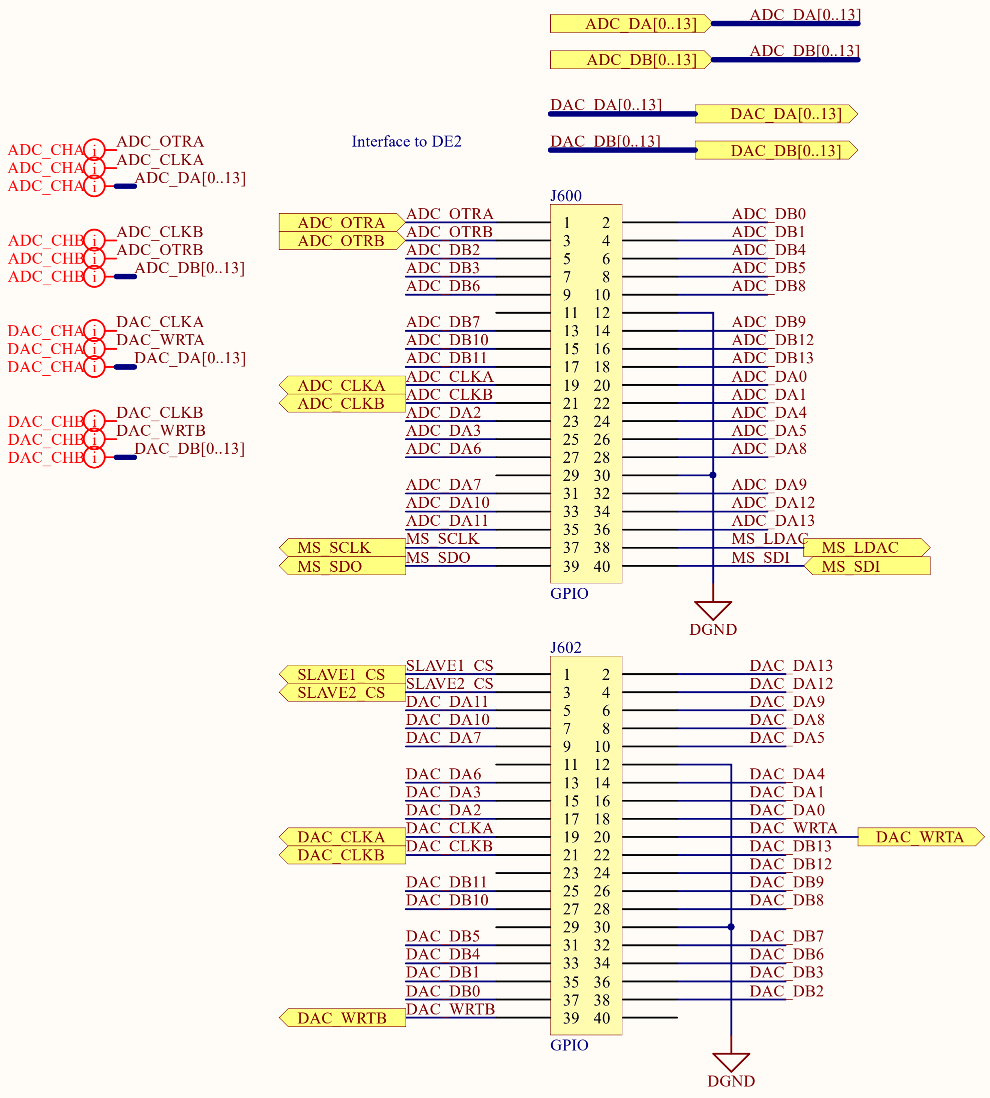

SERVO_IN
OFFSET_INPUT
EXTERNAL_OFFSET
MANUAL_OFFSET
GAIN_INPUT
R306 1K
R310 50
R309 HACK2
R320 DNP
R322 DNP
C300 3pF
R300 1K
U300 AD829JR
NULL1
NULL2
+12V
-12V
R307 0
C306 3pF
C307 25pF
R312 DNP
U302 AD603AR
+5V
-5V
R302 0
GAIN_CTRL
R308 0
INPUT_AF_OUT
R311 499
VIN1
R303 1K1
R301 1K1
C301 9p1
R304 470
C303 25pF
C305 25pF
R314 1K1
R315 470
R316 1K1
C308 9p1
VOCM
U301 AD8137
+IN
-IN
VS+
VS-
/PD
-OUT
+OUT
R305 270
R313 270
C302 56p
C304 56p
VIN_A+
VIN_A-
R321 18k
R324 1K
C325 1uF
C311 1uF
C312 10nF
C319 1uF
C320 10nF
C313 1uF
C314 10nF
C315 1uF
C316 10nF
C321 1uF
C322 10nF
C323 1uF
C324 10nF
R317 DNP
R318 DNP
R319 DNP
AGND
Title
Size
A4
Number
Revision
Date: 2016-06-07
Sheet of
File: C:\Users\..\Input_AF.SchDoc
Drawn By:



Title		
Size A4	Number	Revision
Date: 2016-06-07		Sheet of
File: C:\Users\..\Output_AF.SchDoc		Drawn By:

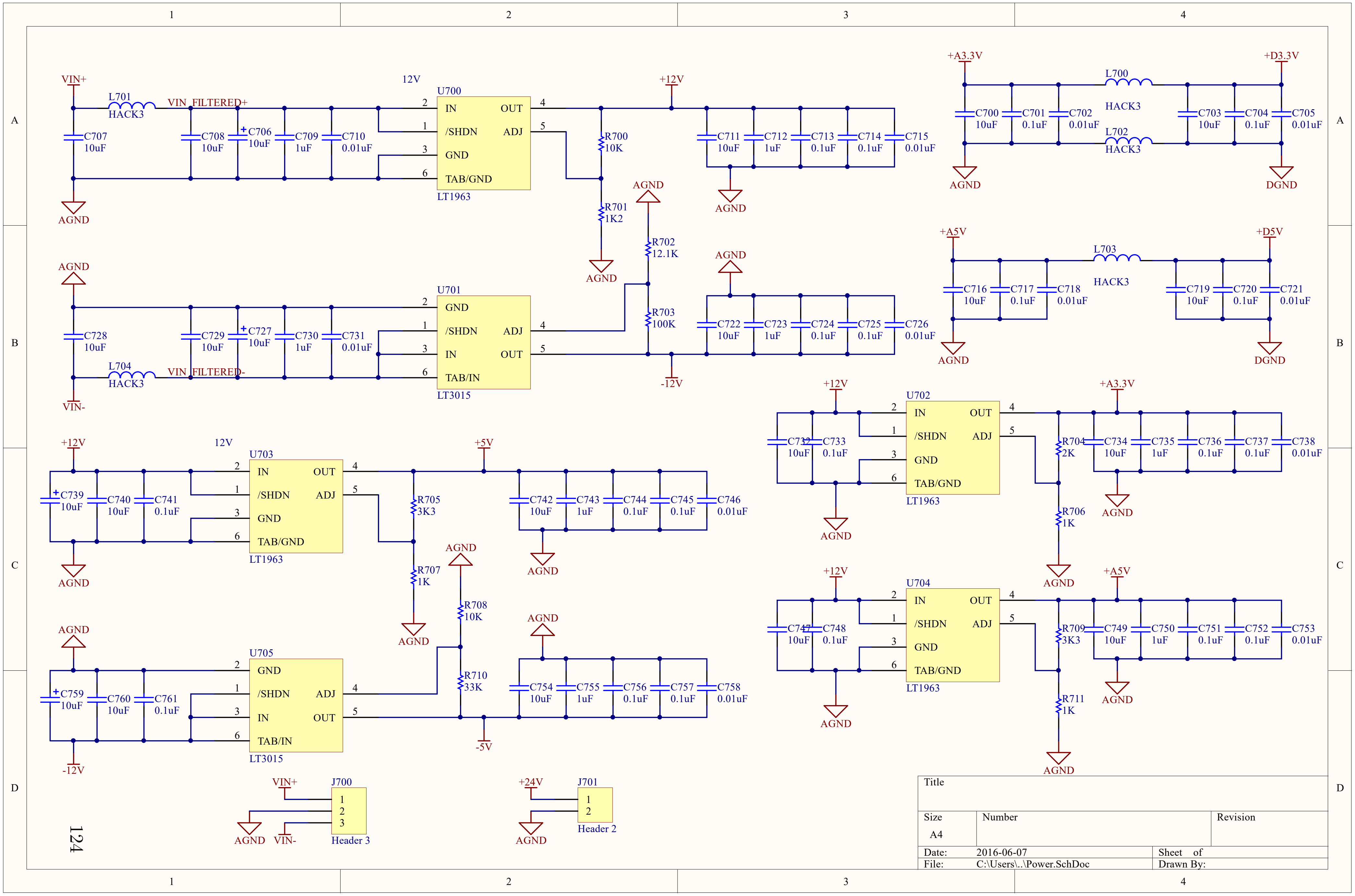


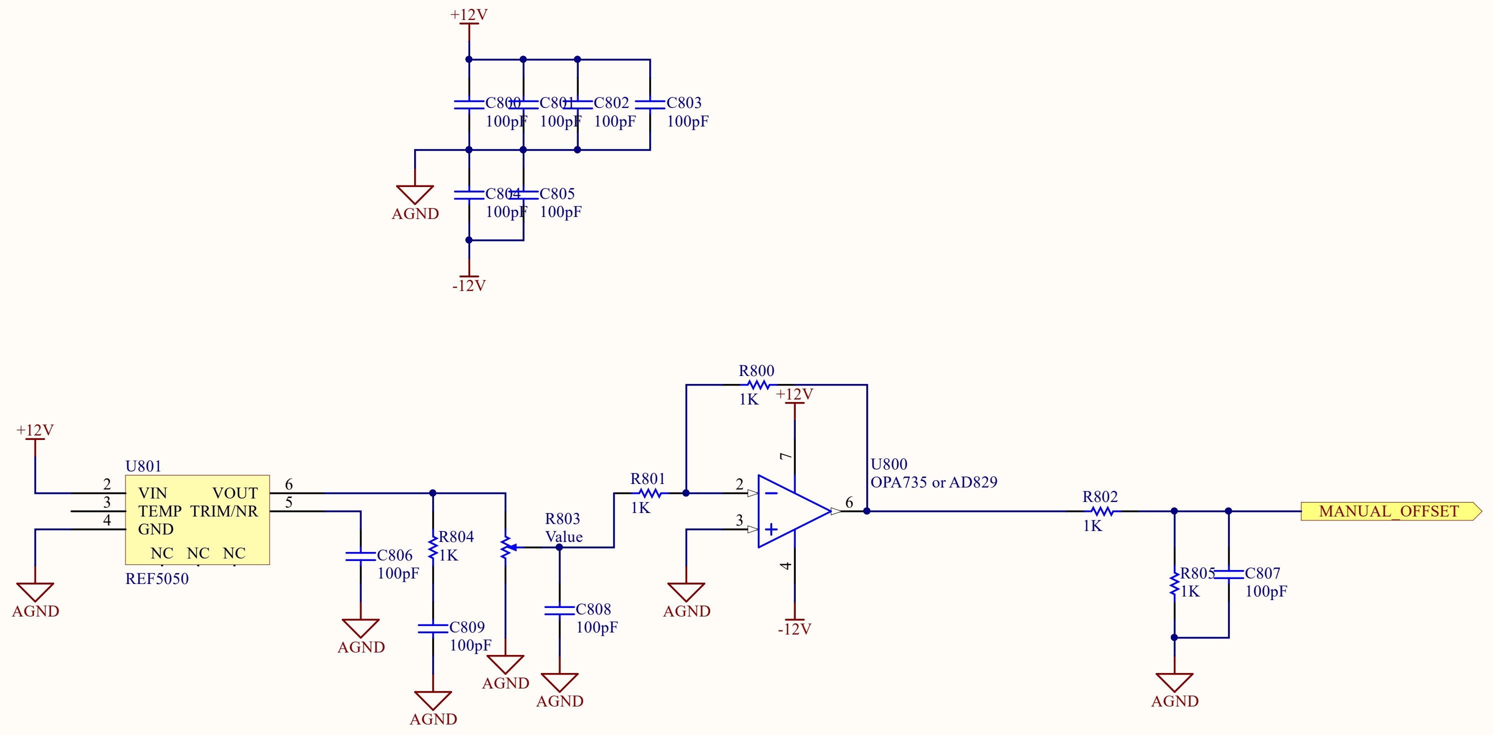
Title		
Size A4	Number	Revision
Date: 2016-06-07	Sheet of	
File: C:\Users\..\Slow_DAC_w_REF.SchDoc	Drawn By:	



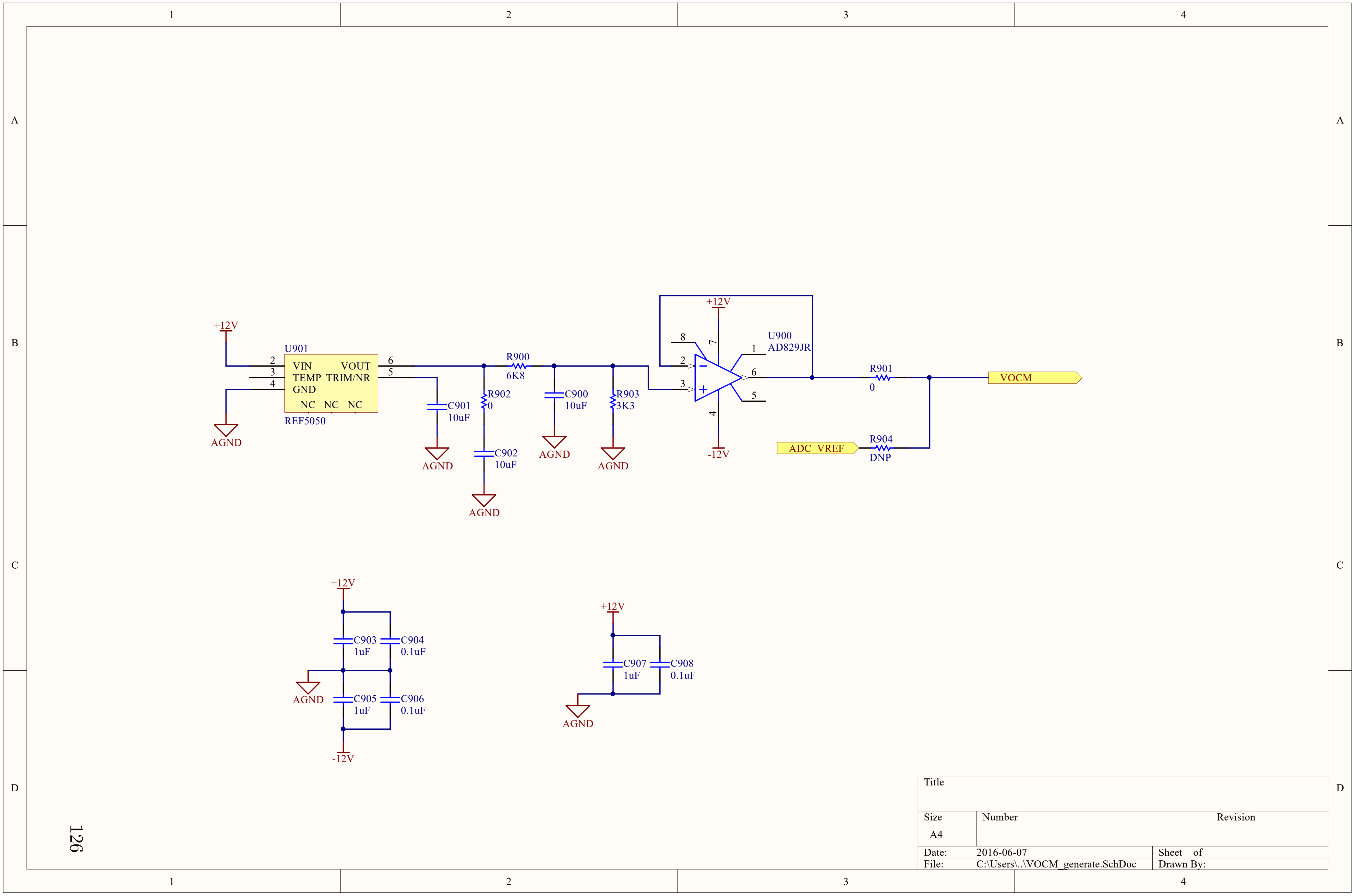
123

Title		
Size A4	Number	Revision
Date: 2016-06-07		Sheet of
File: C:\Users\..\Interface.SchDoc		Drawn By:


VIN+
L701 HACK3
VIN_FILTERED+
12V
U700 LT1963
IN OUT /SHDN ADJ GND TAB/GND
+12V
C707 10uF
C708 10uF
C706 10uF
C709 1uF
C710 0.01uF
R700 10K
R701 1K2
C711 10uF
C712 1uF
C713 0.1uF
C714 0.1uF
C715 0.01uF
AGND
+A3.3V
+D3.3V
L700 HACK3
L702 HACK3
C700 10uF
C701 0.1uF
C702 0.01uF
C703 10uF
C704 0.1uF
C705 0.01uF
DGND
R702 12.1K
R703 100K
U701 LT3015
GND /SHDN ADJ IN OUT TAB/IN
C728 10uF
C729 10uF
C727 10uF
C730 1uF
C731 0.01uF
L704 HACK3
VIN_FILTERED-
VIN-
-12V
C722 10uF
C723 1uF
C724 0.1uF
C725 0.1uF
C726 0.01uF
+A5V
+D5V
L703 HACK3
C716 10uF
C717 0.1uF
C718 0.01uF
C719 10uF
C720 0.1uF
C721 0.01uF
U702 LT1963
C732 10uF
C733 0.1uF
R704 2K
R706 1K
C734 10uF
C735 1uF
C736 0.1uF
C737 0.1uF
C738 0.01uF
U703 LT1963
+5V
C739 10uF
C740 10uF
C741 0.1uF
R705 3K3
R707 1K
C742 10uF
C743 1uF
C744 0.1uF
C745 0.1uF
C746 0.01uF
U704 LT1963
C747 10uF
C748 0.1uF
R709 3K3
R711 1K
C749 10uF
C750 1uF
C751 0.1uF
C752 0.1uF
C753 0.01uF
U705 LT3015
R708 10K
R710 33K
C759 10uF
C760 10uF
C761 0.1uF
C754 10uF
C755 1uF
C756 0.1uF
C757 0.1uF
C758 0.01uF
-5V
J700 Header 3
+24V
J701 Header 2
Title
Size A4
Number
Revision
Date: 2016-06-07
Sheet of
File: C:\Users\..\Power.SchDoc
Drawn By:



Title		
Size A4	Number	Revision
Date: 2016-06-07		Sheet of
File: C:\Users\..\Input_Manual_Offset.SchDoc		Drawn By:



Title		
Size A4	Number	Revision
Date: 2016-06-07		Sheet of
File: C:\Users\..\VOCM_generate.SchDoc		Drawn By:

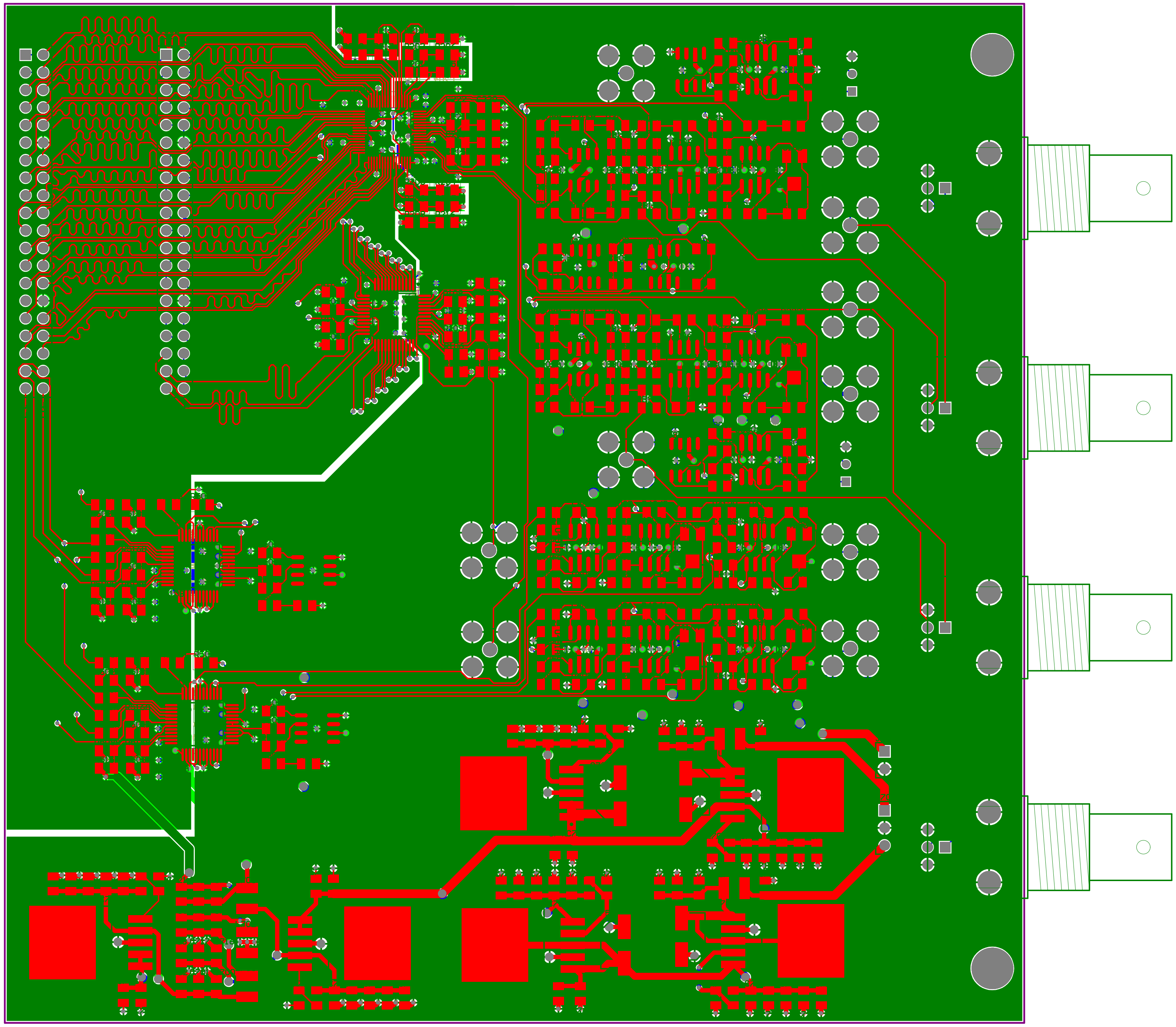


Table A.1: The bill of material of the servo daughter card

Designator	Values	Digikey Number	Quantity
C100, C101, C200, C201, C215, C218, C221, C500A, C500B, C501A, C501B, C502A, C502B, C503A, C503B, C513A, C513B, C700, C703, C707, C708, C711, C716, C719, C722, C728, C729, C732, C734, C740, C742, C747, C749, C754, C760, C900, C901, C902	10uF	311-1459-1-ND	38

Continued on next page

Table A.1 – continued from previous page

Designator	Values	Digikey Number	Quantity
C102, C103, C109, C110, C111, C202, C203, C208, C209, C210, C311A, C311B, C313A, C313B, C315A, C315B, C319A, C319B, C321A, C321B, C323A, C323B, C325A, C325B, C403A, C403B, C405A, C405B, C407A, C407B, C408A, C408B, C409A, C409B, C411A, C411B, C415A, C415B, C417A, C417B, C504A, C504B, C507A, C507B, C509A, C509B, C511A, C511B, C514A, C514B, C709, C712, C723, C730, C735, C743, C750, C755	1 uF	399-1284-1-ND	58

Continued on next page

Table A.1 – continued from previous page

Designator	Values	Digikey Number	Quantity
C104, C105, C108, C204, C205, C214, C216, C217, C219, C220, C222, C223, C505A, C505B, C515A, C515B, C701, C704, C713, C714, C717, C720, C724, C725, C733, C736, C737, C741, C744, C745, C748, C751, C752, C756, C757, C761, C904, C906, C908	0.1 uF	1276-1003-1-ND	39
C106, C107, C112, C113, C114, C206, C207, C211, C212, C213, C506A, C506B, C508A, C508B, C510A, C510B, C512A, C512B, C516A, C516B, C702, C705, C710, C715, C718, C721, C726, C731, C738, C746, C753, C758	0.01uF	311-1136-1-ND	32
C300A, C300B, C306A, C306B, C400A, C400B, C401A, C401B	3 pF	399-1107-1-ND	8
C301A, C301B, C308A, C308B	9.1 pF	311-1098-1-ND	4
C302A, C302B, C304A, C304B	56 pF	1276-1833-1-ND	4

Continued on next page

Table A.1 – continued from previous page

Designator	Values	Digikey Number	Quantity
C303A, C303B, C305A, C305B, C307A, C307B, C402A, C402B	25 pF	1276-2621-1-ND	8
C312A, C312B, C314A, C314B, C316A, C316B, C320A, C320B, C322A, C322B, C324A, C324B, C404A, C404B, C406A, C406B, C410A, C410B, C412A, C412B, C416A, C416B, C418A, C418B	10 nF	311-1136-1-ND	24
C413A, C413B, C414A, C414B, C800A, C800B, C801A, C801B, C802A, C802B, C803A, C803B, C804A, C804B, C805A, C805B, C806A, C806B, C807A, C807B, C808A, C808B, C809A, C809B	100 pF	399-1122-1-ND	30
C706, C727, C739, C759	10 uF	PCE3914CT-ND	4
J3, J4, J9, J10	BNC_DUAL	ACX1655-ND	4
J600, J602	GPIO	OR1153-ND	2
R102, R103, R200, R201	100 Ω	311-100CRCT-ND	4
R104, R105, R204, R205	150 Ω	311-150CRCT-ND	4
R106, R108, R109, R704	2 kΩ	311-2.00KCRCT-ND	4

Continued on next page

Table A.1 – continued from previous page

Designator	Values	Digikey Number	Quantity
R107, R302A, R302B, R307A, R307B, R308A, R308B, R401A, R401B, R402A, R402B, R404A, R404B, R415A, R415B, R416A, R416B, R500A, R500B, R501A, R501B, R506A, R506B, R510A, R510B, R511A, R511B, R512A, R512B, R516A, R516B, R901, R902	0	311-0.0ARCT-ND	33 Ω
R110, R112	25 Ω	311-24.9CRCT-ND	2
R111, R113, R310A, R310B	50 Ω	P49.9CCT-ND	4

Continued on next page

Table A.1 – continued from previous page

Designator	Values	Digikey Number	Quantity
R206, R207, R208, R209, R218, R219, R300A, R300B, R306A, R306B, R324A, R324B, R400A, R400B, R403A, R403B, R406A, R406B, R412A, R412B, R417A, R417B, R418A, R418B, R502A, R502B, R503A, R503B, R504A, R504B, R505A, R505B, R509A, R509B, R513A, R513B, R514A, R514B, R706, R707, R711, R800A, R800B, R801A, R801B, R802A, R802B, R804A, R804B, R805A, R805B	$1k\ \Omega$	311-1.00KCRCT-ND	50
R301A, R301B, R303A, R303B, R314A, R314B, R316A, R316B	1.1 kΩ	311-1.10KCRCT-ND	8 Ω
R304A, R304B, R315A, R315B	470 Ω	311-470CRCT-ND	4
R305A, R305B, R313A, R313B	270 Ω	311-270CRCT-ND	4
R311A, R311B, R405A, R405B	499 Ω	311-499CRCT-ND	4

Continued on next page

Table A.1 – continued from previous page

Designator	Values	Digikey Number	Quantity
R321A, R321B, R408A, R408B	18 kΩ	311-18.0KCRCT-ND	4
R409A, R409B	1.5 kΩ	311-1.50KCRCT-ND	2
R413A, R413B	401 Ω	311-402CRCT-ND	2
R700, R708	10 kΩ	311-10.0KCRCT-ND	2
R701	1.2 kΩ	311-1.20KCRCT-ND	1
R702	12.1 kΩ	311-12.1KCRCT-ND	1
R703	100 kΩ	311-100KCRCT-ND	1
R705, R709, R903	3.3 kΩ	311-3.30KCRCT-ND	3
R900	6.8 kΩ	311-6.8KARCT-ND	1
R710	33 kΩ	311-33.0KCRCT-ND	1
U300A, U300B, U400A, U400B, U800A, U800B, U900	AD829JR	AD829JRZ-ND	7
U301A, U301B	AD8137	AD8137YRZ-REEL7CT-ND	2
U302A, U302B, U401A, U401B	AD603AR	AD603ARZ-REEL7CT-ND	4
U500A, U500B, U801A, U801B, U901	REF5050	296-22211-5-ND	5
U700, U702, U703, U704	LT1963	LT1963AEQ#TRPBFCT-ND	4
U701, U705	LT3015	LT3015EQ#PBF-ND	2
U100	AD9767		1
U200	AD9248		1
U501A, U501B	DAC8734		2
J1, J2, J5, J6, J7, J8, J11, J12, J13, J14	COAX-F		10
J700	Header 3		1

Continued on next page

Table A.1 – continued from previous page

Designator	Values	Digikey Number	Quantity
J701	Header 2		1

Appendix B

The Variable Attenuator

The variable RF attenuator used for intensity control in Chapter 4 and Chapter 5 is constructed for general use in the lab. The characteristics of this variable attenuator such as the speed of the variable attenuator and the relationship between input voltage and RF transmission is covered in Section 4.2.3. This section covers the finer technical details, such as special assembly instructions and pending design changes.

B.1 Design Details and Pending Changes

The variable attenuator is designed around the MAAVSS0006 IC. Because the IC is not available commercially in a package that is convenient to use, a PCB and enclosure is designed to host the IC. This allows the variable attenuator to have a BNC interface. In this design, two variable attenuating channels are designed into each package and they are mean to be used in cascade mode to achieve modulation range of 60-70 dB. The device need a quiet $\pm 13 - 15$ V power supply, supplied through a 5-pin mini-XLR connector. The pin-out of this connector is compatible with the standard power supply standard in the QDG lab.

The design document presented here is the second PCB version of the variable attenuator design. It fixed various problem in the original PCB design although some problem still remains.

One mistakes at the schematics level needs to be corrected in future revision and it is the compensation scheme of the AD829. The compensation capacitor, C15 and C16 in their respective channels are incorrectly attached to the Pin 8 of the AD829, rather, it should be attached to Pin 5. The mistake is patched by soldering C15 and C16 between Pin 5 of the respective op-amp and a nearby via on the ground plane, but it should be fixed as the patch add additional processing step to the design. While fixing this mistake, it is also a good chance to try out the current compensation scheme, as the work found the current compensation scheme to be typically faster than the voltage compensation scheme. Section 4.2.3 shows evidence that the circuit around AD829 is the bottleneck in speed so such a change has the potential to improve modulation speed. The change is very simple and should not disturb the layout else where on the PCB.

B.2 Calibration

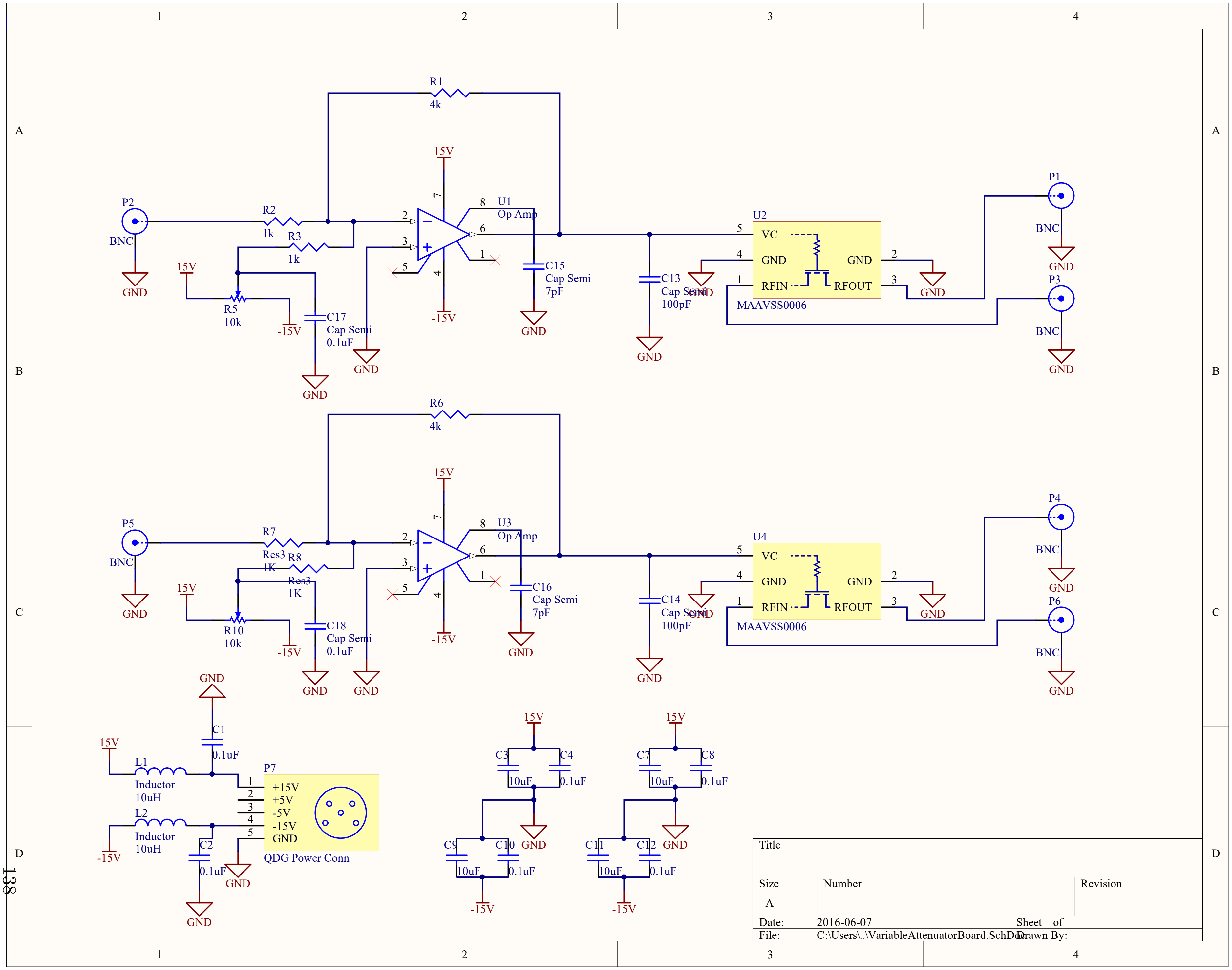
Each variable attenuator has a different relationship between its input control voltage and the RF transmission. This can be observed in Figure 4.10 and we did not look into the cause of this variable. Thus, each variable attenuator needs to be calibrated. It may help to use the potentiometer R5 and R10 to add an offset to the input voltage so that the variable attenuator transition from “on” to “off” between 0 to 0.5 V.

B.3 Assembly

In the second version of the design (design files are included in later part of this section), all the connectors are soldered to the PCB and can be fastened to the removable panel on the enclosure for extra security. The panels can be machined from a flat sheet of metal in a water jet machine. Once the PCB is soldered and calibrated, the assembly is complete by sliding the PCB into the existing card guide in the commercial enclosure and attaching the panels. The easiest way to do this is to compress the enclosure on a vise to prevent the PCB from sticking to card guide.

B.4 Design Files

The design documents presented here, the schematics, PCB layout, the bill of material and the panel design is sufficient to reproduce the variable attenuator design.



R1 4k
15V
P2 BNC GND
R2 1k
R3 1k
U1 Op Amp
C15 Cap Semi 7pF
C13 Cap Semi 100pF
U2 MAAVSS0006
VC GND RFIN GND RFOUT
P1 BNC GND
P3 BNC GND
R5 10k
-15V
C17 Cap Semi 0.1uF
R6 4k
P5 BNC GND
R7 Res3 1K
R8 Res3 1K
U3 Op Amp
C16 Cap Semi 7pF
C14 Cap Semi 100pF
U4 MAAVSS0006
P4 BNC GND
P6 BNC GND
R10 10k
C18 Cap Semi 0.1uF
C1 0.1uF
L1 Inductor 10uH
L2 Inductor 10uH
C2 0.1uF
P7 QDG Power Conn
+15V +5V -5V -15V GND
C3 10uF C4 0.1uF
C7 10uF C8 0.1uF
C9 10uF C10 0.1uF
C11 10uF C12 0.1uF
Title
Size A
Number
Revision
Date: 2016-06-07
Sheet of
File: C:\Users\..\VariableAttenuatorBoard.SchDoc
Drawn By:

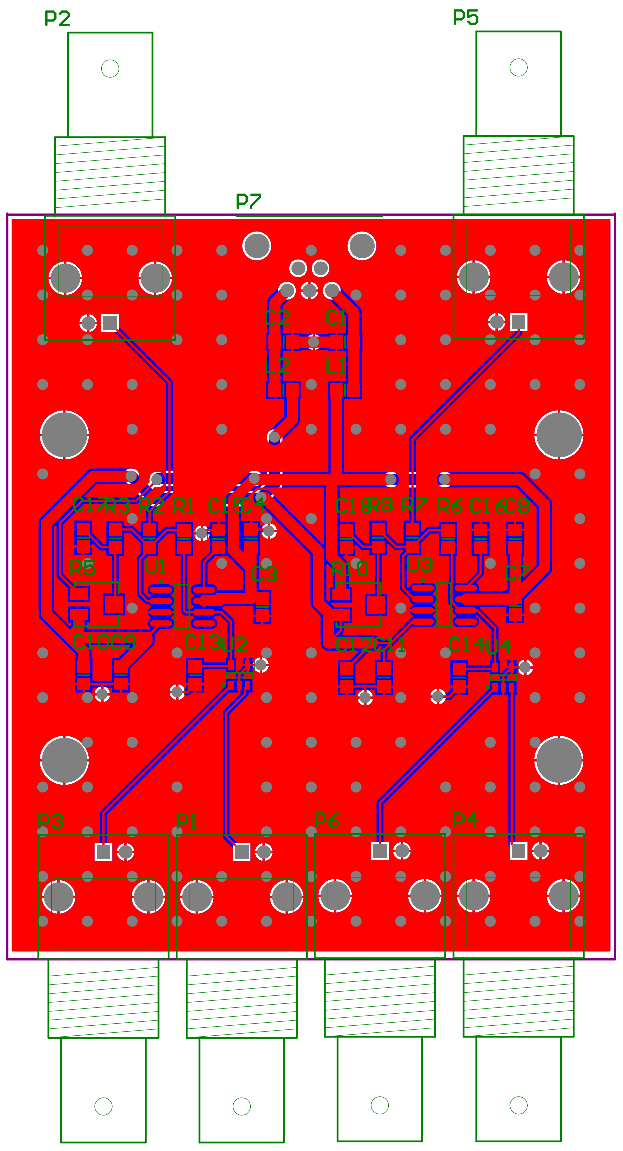

P2
P5
P7
C2
C1
L2
L1
C17
R3
R2
R1
C5
C4
C18
R8
R7
R6
C16
C6
R5
U1
C3
R10
U3
C7
C10
C9
C13
U2
C12
C11
C14
U4
P3
P1
P6
P4

Table B.1: The bill of material of the variable attenuator unit.

Electrical			
Designator	Values	Manufacturer Part Number	Quantity
U2, U4	Variable attenuator	MAAVSS0006TR-3000	2
U1, U3	AD829	AD829JRZ	2
R2, R3, R7, R8	1k	RC0805FR-071KL	4
R1, R6	4.02k	RC0402FR-074K02L	2
R5, R10	10k	SM-42TW103	2
P1, P2, P3, P4, P5, P6	BNC	1-1337543-0	6
P7	Mini-XLR	TRAPC5MX	1
C13, C14	100pF	C0805C101J5GACTU	2
C15, C16	7pF	CL21C070CBANNNC	2
C3, C9, C7, C11	10uF	GRM21BR61C106KE15K	4
C1, C2, C4, C10, C8, C12, C17, C18	10uF	CL21B104KBCNNNC	8
L1, L2	100uH	LBR2012T100K	2
Mechanical			
Enclosure	-	EX-4500	1

Appendix C

Register Address of FPGA Peripheral

The SOC design FPGA servo is composed of an MCU embedded in the FPGA and a few specialized modules for data processing. The MCU specializes in communication tasks such as processing the commands between the PC and the servo and the coordination between the modules. The specialized modules handle high-speed signal processing tasks that require the flexibility of the FPGA to implement different logical structures.

When generating an SOC, each module instantiated in the SOC are given a unique address. These addresses are used by the MCU to access the modules and each parameter in a specialized module can be accessed at a fixed offset from the base address. The offset addresses of these paremeters are listed in Table C.1 - C.5. In an SOC that contains multiple IIR filters, each filter is given a unique base address.

The advantage of this design scheme is flexibility and portability. Each specialized module can be designed and tested as a unit. To expand an FPGA servo design from a single IIR filter per channel to 3 IIR filter configured in cascade, the work only involves editing the configuration of the SOC to instantiate more IIR filters and editing the command interface in the MCU to make sure that the PC can control the newly added IIR modules.

It is important to note that although the Serial Peripheral Interface (SPI) module used to control the slow DAC is a very general structure generated from Altera toolbox, the design in MCU needs to take into account the command structure of the slow DAC (each command to the slow DAC is 3 Byte long). Because two ICs are connected in the daisy-chain fashion, in order to talk the second IC in the chain, the MCU must issue 3 bytes carrying the command intended for the second IC and the immediately issue another 3 Bytes (could be just a No

Operation (NOP) command) in order to prompt the first IC in the chain to relay the first 3 bytes to the second IC.

Table C.1: The address at which the IIR coefficients can be accessed in the general purpose IIR filter tested on the DE2 FPGA. The coefficients are signed.

Register Address	Register Function	Format
0	B_0	Q5.10
1	B_1	Q5.10
2	B_2 (If used)	Q5.10
3	B_3 (If used)	Q5.10
4	A_1	Q5.10
5	A_2 (If used)	Q5.10
6	A_3 (If used)	Q5.10

Table C.2: The address at which the IIR coefficients can be accessed in the IIR filter designed for the DE3 FPGA.

Register Address	Register Function	Format
0	B_0	Q3.28
1	B_1	Q3.28
2	B_2 (If used)	Q3.28
3	B_3 (If used)	Q3.28
4	A_1	Q3.28
5	A_2 (If used)	Q3.28
6	A_3 (If used)	Q3.28

Table C.3: The register address of the FIR filter.

Register Address	Register Function	Format
0	B_0	Q5.10
1	B_1	Q5.10
2	B_2 (If used)	Q5.10
3	B_3 (If used)	Q5.10
4	B_4 (If used)	Q5.10
5	B_5 (If used)	Q5.10
6	B_6 (If used)	Q5.10

Table C.4: The register address for the SPI module that is used to control the slow DACs in the FPGA servo based on the DE2 and the custom-made servo daughtercard. In the daughtercard, two ICs are connected in a daisy chain fashion. To communicate to the second IC in the chain, data has to be sent out in multiples of 3 bytes. The SPI module is generated from Altera's toolbox.

Register Address	Register Function	Format
0	CPU Read	U8
1	CPU Write	U8
2	SPI Status	1-bit
3	SPI Control	1-bit
4	-	-
5	Slave Select	1-bit
6	End of Packet	1-bit

Table C.5: The register address of the AWG module

Register Address	Register Name	Format
0	Ramp_Inc	U32
1	Ramp_Max	U32
2	Ramp_Min	U32
3	Ramp_Prescaler	U16
4	A_0	Q5.10
5	A_1	Q5.10
6	A_2	Q5.10
7	A_3	Q5.10
8	A_4	Q5.10

Appendix D

PC-Side Software

The PC-side software is responsible for generating the IIR coefficients, AWG coefficients and providing a Graphical User Interface (GUI) for the FPGA servo user. The majority of the PC-side software including the GUI is developed in MATLAB. However, the design can be ported into a different environment as the commands to the FPGA is well defined and non-proprietary.

D.1 Command Format

The command interface of the FPGA exposed control to modules in the SOC as setting that can be accessed at unique addresses. The address to the setting is defined in the code that run in the MCU where the commands are interpreted. The setting from different module are typically assigned a unique offset. To access the setting the keywords “set”, “get” and “print” are used, with the format like “keyword address (value)”. For example, a dual channel FPGA servo with a single IIR filter maybe need a command “set 0 1024” and “set 16 1024” to set the B_0 of the IIR filter for the first servo channel and the second servo channel respectively.

D.2 IIR Coefficients and Screening of Transfer Functions

Because the IIR filter has limited resolution, the PC-side software find the nearest transfer function that the FPGA servo can implement and compares it to the transfer function that the user requested. The software leaves it up to the user to make sure that the transfer function implemented by the hardware does not cause instability in the loop.

For the PC-side software to make this comparison correctly a few parameters need to be recorded correctly in PC-side software. The important parameters are the clock speed of the IIR logic (not to be confused with clock speed of the ADC or the DAC which can be different), the resolution of the IIR coefficients that is implemented in the FPGA servo (the common formats are Q5.10 and Q3.28) and how the IIR filters are cascade. The PC side program also needs to know the format of the commands and the address that each coefficient can be accessed at (this is not be confused with the location where these coefficients are stored in the MCU's memory space).

Currently, there is no way of knowing the exact configuration of IIR filters that is programmed in the FPGA without accessing the source code that describes the FPGA circuitry. This can be a source of confusion when FPGA servos with different IIR configurations are present. We currently combat this problem with the use of unique system ID which is a peripheral generated by the Altera toolbox which receives a unique timestamp every time the SOC is regenerated. The system ID peripheral is also given a device ID to differentiate IIR filter designed for different FPGAs.

This method still requires access to the source code to make sure that the program on the PC and inside the MCU in the FPGA are compatible with the FPGA design. If the developer make sure to only release one servo design for general use then only one PC side program needs to be released and no compatibility issue would surface. In the case where multiple servo designs are needed, a comprehensive way of fixing this is to add special command in the MCU to allow the FPGA servo to inform any potential PC host its underlying IIR filter structure. The PC side program are then required to query this information from the FPGA servo and adapt its tuning interface accordingly.

To illustrate the importance of this screening process, an example output of the tuning interface is shown in Figure D.1. Here the tuning interface is "hard-coded" to interface with an FPGA servo where the underlying FPGA logic is a single third order IIR filter. The output based two IIR filters are overlay to illustrate the effect of computation precision in IIR filter. It can be observed in Figure D.1 that the IIR filter with 32-bit coefficients can implement the shape of a PID control although it has trouble addressing the high frequency pole precisely. In comparison, the IIR filter with 16-bit coefficients fails to implement the integrator (at low frequency) and the resulting transfer function is unusable in closed-loop.

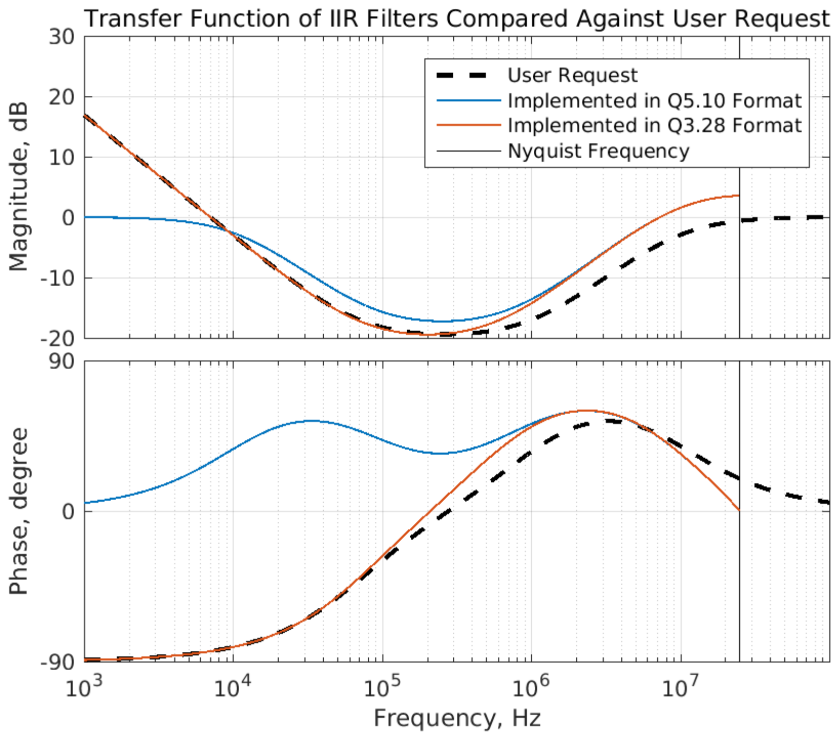


Figure D.1: Example output of the PC software. In this case, the user input are zeros at -70 kHz and -1 MHz and poles at 0 and -10 MHz and the underlying servo logic is implemented as a single IIR filter. The input forms an PID controller, but IIR filter with different coefficient resolution emulate the transfer function with different accuracy. The resulting transfer function from the IIR filter with 16-bit coefficient (Q5.10) fails to implement the integrator.

D.3 Slow DAC Calibration Table

The slow DAC calibration table is needed for the variable gain circuits in the FPGA servo because often in a servo loop the gain has to be adjusted for optimal performance. The VGA used in this FPGA servo is linear-in-dB. Instead of approximating the gain as a exponential function, this work approximate the gain with a calibration table with value in the slow DAC and gain. Any intermediate gain is interpolated linearly from the closest two entries in the calibration table.

Each offset circuit is calibrated with a free parameter that represents the linear relationship between the slow DAC value and the offset voltage.